BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Towards Secure Decentralized Applications and Consensus Protocols in Blockchains

Habilitation Thesis

Ing. Ivan Homoliak Ph.D. Brno 2023

Abstract

With the rise of cryptocurrencies, many new applications and approaches leveraging the decentralization of blockchains have emerged. Blockchains are full-stack distributed systems in which multiple sub-systems interact together. Although most of the deployed blockchains and decentralized applications running on them need better scalability and performance, their security is undoubtedly another critical factor for their success. Due to the complexity of blockchains and many decentralized applications, their security assessment and analysis require a more holistic view than in the case of traditional distributed or centralized systems.

In this thesis, we summarize our contributions to the security of blockchains and a few types of decentralized applications. In detail, we contribute to the standardization of vulnerability/threat analysis by proposing a security reference architecture for blockchains. Then, we contribute to the security of consensus protocols in single-chain Proof-of-Work blockchains and their resistance to selfish mining attacks, undercutting attacks as well as greedy transaction selection attacks on blockchains with Direct Acyclic Graphs. Next, we contribute to cryptocurrency wallets by proposing a new classification of authentication schemes as well as a novel approach to two-factor authentication based on One-Time Passwords. Next, we contribute to the area of e-voting by proposing a practical boardroom voting protocol that we later extend to its scalable version supporting millions of participants, while maintaining its security and privacy properties. In the area of e-voting, we also propose a novel repetitive voting framework, enabling vote changes in between elections while avoiding peak-end effects. Finally, we contribute to secure logging with blockchains and trusted computing by proposing a new approach to a centralized ledger that guarantees non-equivocation, integrity, censorship evidence, and other features. In the follow-up contribution to secure logging, we built on top of our centralized ledger and propose an interoperability protocol for central bank digital currencies, which provides atomicity of transfer operations.

Keywords

Security, privacy, blockchains, distributed ledgers, threat modeling, standardization, decentralized applications, consensus protocols, proof-of-work, selfish mining attacks, undercutting attacks, incentive attacks, DAG-based blockchains, cryptocurrency wallets, two-factor authentication, 2FA, electronic voting, e-voting, public bulletin board, secure logging, data provenance, interoperability, Central Bank Digital Currency, CBDC, Trusted Execution Environment, TEE, cross-chain protocol.

Acknowledgment

First, I would like to thank my supervisor and colleague from SUTD – Pawel Szalachowski – for collaborating and igniting many interesting research ideas as well as explaining to me that ideas are important but cheap in contrast to realization. Second, I would like to thank all my co-authors, especially Sarad Venugopalan, Pieter Hartel, Daniel Reijsbergen, Federico Matteo Benčić, Fran Casino, and Martin Hrubý. Third, I would like to thank Tomas Vojnar, Pavel Zemcik, Pavel Smrz, and Ales Smrcka for their support in funding my research at FIT BUT. Next, I would like to thank all Ph.D./MSc/BSc students I have had the chance to work with, especially Martin Peresini, Ivana Stančíková, and Dominik Breitenbacher / Tomas Hladky, Jakub Handzus, Jakub Kubik, and Martin Ersek / and Rastislav Budinský. Also, I would like to thank Milan Ceska (and Tomas Vojnar) for inspiring me in the habilitation process and helping me to maximize cross-domain transfer learning in this process. Finally, I would like to thank around 20% of anonymous reviewers (mostly from security venues) for providing useful and constructive feedback.

Over the time, I have been supported by number of projects, in particular, by EU ECSEL projects, EU HORIZON EUROPE projects, National Research Foundation – Prime Minister's Office in Singapore, Technology Agency of the Czech Republic, and the Czech IT4Innovations Centre of Excellence project.

Contents

Part I

COMMENTED RESEARCH

Chapter 1

Introduction

The popularity of blockchain systems has rapidly increased in recent years, mainly due to the decentralization of control that they aim to provide. Blockchains are full-stack distributed systems in which multiple layers, (sub)systems, and dynamics interact together. Hence, they should leverage a secure and resilient networking architecture, a robust consensus protocol, and a safe environment for building higher-level applications. Although most of the deployed blockchains need better scalability and well-aligned incentives to unleash their full potential, their security is undoubtedly a critical factor for their success. As these systems are actively being developed and deployed, it is often challenging to understand how secure they are, or what security implications are introduced by some specific components they consist of. Moreover, due to their complexity and novelty (e.g., built-in protocol incentives), their security assessment and analysis require a more holistic view than in the case of traditional distributed systems.

In this work, we first present our contributions to the standardization of vulnerability/threat analysis and modeling in blockchains, and then we present our contributions to particular areas in blockchains' consensus protocols, cryptocurrency wallets, electronic voting, and secure logging with the focus on security and/or privacy aspects. In the following, we introduce these areas and outline our contributions.

1.1 Standardization in Threat Modeling

Although some standardization efforts have already been undertaken in the field of blockchains and distributed ledgers, they are either specific to a particular platform [Ent19] or still under development [ISO19b, ISO19a]. Hence, there is a lack of platform-agnostic standards in blockchain implementation, interoperability, services, and applications, as well as the analysis of its security threats [Goa19, Bar18]. All of these areas are challenging, and it might take years until they are standardized and agreed upon across a diverse spectrum of stakeholders.

We believe that it is critical to provide blockchain stakeholders (developers, users, standardization bodies, regulators, etc.) with a comprehensive systematization of knowledge about the security and privacy aspects of today's blockchain systems. We aim to achieve this goal, with a particular focus on system design and architectural aspects. We do not limit our work to an enumeration of security issues, but additionally, discuss the origins of those issues while listing possible countermeasures and mitigation techniques together with their potential implications. In sum, we propose the security reference architecture (SRA) for blockchains, which is based on models that demonstrate the stacked hierarchy of different threat categories (similar to the ISO/OSI hierarchy [Zim80]) and is inspired by security modeling performed in the cloud computing [LTM$^+$11, XX13]. As our next contribution in this direction, we enrich the threat-risk assessment standard ISO/IEC 15408 [Com17] to fit the blockchain infrastructure. We achieve this by embedding the stacked model into this standard. More details in this direction of our research are elaborated in Chapter 3.

1.2 Consensus Protocols

While the previous area of the thesis was theoretical and analytical, in the current area of consensus protocols we aim to investigate practical security aspects of blockchains, and their consensus protocols in particular. Consensus protocols represent a means to provide naturally incentivized decentralization, immutability, and other features of blockchains (see Section 2.1.1). Therefore, modeling and simulation of consensus protocols in terms of security and incentives is an important research direction. There exist several principally different categories of consensus protocols such as Proof-of-Resource (PoR), Proof-of-Stake, and Byzantine-Fault-Tolerant protocols (see Section 3.4), each of them potentially vulnerable to different types of threats. Nevertheless, in this research area, we focus on PoR protocols and Proof-of-Work (PoW) protocols in particular.

As our first contribution, we design StrongChain [SRHS19] consensus protocol that improves the resistance of Nakamoto consensus [Nak08] to selfish mining by rewarding partial partial PoW puzzle solutions and incorporating them to the total "weight" of the chain. While the idea of rewarding partial puzzle solutions is not novel [ZSS$^+$18, PS17, Riz16], StrongChain achieves resistance to selfish mining in a space-efficient manner that does not create a new vulnerability (such as selfish mining on a subchain in [ZSS$^+$18]). At the same time, StrongChain improves on accuracy of distributed time and decreases the reward variance of miners, and thus it creates better conditions for more decentralized mining.

Our second contribution is in the area of consensus protocols that utilize Directed Acyclic Graphs (DAGs) to solve the limited processing throughput of traditional single-chain Proof-of-Work (PoW) blockchains. Many such protocols (e.g., Inclusive [LSZ15], GHOSTDAG [SWZ21], PHANTOM [SWZ21], SPECTRE [SLZ16], Prism [BKT$^+$19]) utilize a random transaction selection (RTS) strategy to avoid transaction duplicates across parallel blocks in DAG and thus maximize the network throughput. However, these works have not rigorously examined incentive-oriented greedy behaviors when transaction selection deviates from the protocol, which motivated our research. Therefore, we first perform a generic game-theoretic analysis abstracting several DAG-based blockchain protocols that use the RTS strategy [PBH+], and we prove that such a strategy does not constitute a Nash equilibrium. Then, we design a simulator [PHMH24] and perform experiments confirming that greedy actors who do not follow the RTS strategy can profit more than honest miners and harm the processing throughput of the protocol [PBH+]. We show that this effect is indirectly proportional to the network propagation delay. Finally, we show that greedy miners are incentivized to form a shared mining pool to increase their profits, which undermines decentralization and degrades the design of the protocols in question. Finally, we elaborate on a few techniques to mitigate such incentive attacks.

In our last contribution, we mainly focus on the undercutting attacks in the transaction-fee-based regime (i.e., without block rewards) of PoW blockchains with the longest chain fork-choice rule. Note that such a regime is expected to occur in Bitcoin's consensus protocol around the year 2140. Additionally, we focus on two closely related problems: (1) fluctuations in mining revenue and (2) the mining gap – i.e., a situation, in which the immediate reward from transaction fees does not cover miners' expenditures. To mitigate these issues, we propose a solution [BHS23] that splits transaction fees from a mined block into two parts – (1) an instant reward for the miner of a block and (2) a deposit sent to one or more fee-redistribution smart contracts ($\mathcal{FRSC}$s) that are part of the consensus protocol. At the same time, these $\mathcal{FRSC}$s reward the miner of a block with a certain fraction of the accumulated funds over a predefined time. This setting enables us to achieve several interesting properties that improve the incentive stability and security of the protocol, which is beneficial for honest miners. With our solution, the fraction of DEFAULT-COMPLIANT miners who strictly do not execute undercutting attacks is lowered from the state-of-the-art [CKWN16a] result of 66% to 30%. More details in this direction of our research are presented in Chapter 4.

1.3 Cryptocurrency Wallets

With the recent rise in the popularity of cryptocurrencies, the security and management of crypto-tokens have become critical. We have witnessed many attacks on users and wallet providers, which have resulted in significant financial losses. To remedy these issues, several wallet solutions have been proposed. According to the previous work [ECBS18, BMC+15], there are a few categories of common (single-factor) key management approaches, such as password-protected/password-derived wallets, hardware wallets, and server-side/client-side hosted wallets. Each category has its respective drawbacks and vulnerabilities.

To increase the security of former wallet categories, multi-factor authentication (MFA) is often used, which enables spending crypto-tokens only when several secrets are used together. However, we emphasize that different security implications stem from the multi-factor authentication executed *against a centralized party* (e.g., username/password or Google Authenticator) and *against the blockchain* itself. In the former, the authentication factor is only as secure as the centralized party, while the latter provides stronger security that depends on the assumption of an honest majority of decentralized consensus nodes (i.e., miners) and the security of cryptographic primitives used.

In our first contribution in this direction, we propose a classification scheme [HBH+20b] for cryptocurrency wallets that distinguishes between the authentication factors validated against the blockchain and a centralized party (or a device). We apply this classification to several existing wallets that we also compare in terms of various security features.

In our second contribution, we focus on the security vs. usability of wallets using MFA against the blockchain, provided by the wallets from a split control category [ECBS18]. MFA in these wallets can be constructed by threshold cryptography wallets [GGK$^+$15, Myc19], multi-signatures [Arm16, Ele18, Tru19, Cop19], and state-aware smart-contracts [Unc18, Tec18, Con19a]. The last class of wallets is of our concern, as spending rules and security features can be encoded in a smart contract. Although there are several smart-contract wallets using MFA against the blockchain [Unc18, Con19a], to the best of our knowledge, none of them provides an air-gapped authentication in the form of short OTPs similar to Google Authenticator. Therefore, we propose SmartOTPs [HBH+20a], a framework for smart-contract cryptocurrency wallets, which provides 2FA against data stored on the blockchain. The first factor is represented by the user's private key and the second factor by OTPs. To produce OTPs, the authenticator device of SmartOTPs utilizes hash-based cryptographic constructs, namely a pseudo-random function, a Merkle tree, and hash chains. We propose a novel combination of these elements that minimizes the amount of data transferred from the authenticator to the client, which enables us to implement the authenticator in a fully air-gapped setting. SmartOTPs provide protection against three exclusively occurring attackers: the attacker who possesses the user's private key *or* the attacker who possesses the user's authenticator *or* the attacker that tampers with the client. More details in this direction of our research are presented in Chapter 5.

1.4 Electronic Voting

Voting is an integral part of democratic governance, where eligible participants can cast a vote for their representative choice (e.g., candidate or policy) through a secret ballot. Electronic voting (e-voting) is usually centralized and suffers from a single point of failure that can be manifested in censorship, tampering, and issues with the availability of a service. To improve some features of e-voting, decentralized blockchain-based solutions can be employed, where the blockchain represents a public bulletin board that in contrast to a centralized bulletin board provides extremely high availability, censorship resistance, and correct code execution. A blockchain ensures that all entities in the voting system have the same view of the actions made by others due to its immutability and append-only features. A few blockchain-based e-voting solutions have been proposed in recent years, mostly focusing on boardroom voting [MSH17, PR18, LSY$^+$20, YLS$^+$18] or small-scale voting [SGY20, DMMM18, LSY$^+$20].

Decentralization was a desired property of e-voting even before the invention of blockchains. For example, (partially) decentralized e-voting that uses the homomorphic properties of El-Gamal encryption was introduced by Cramer et al. [CGS97]. It assumes a threshold number of honest election authorities to provide the privacy of vote. However, when this threshold is adversarial, it does not protect from computing partial tallies, making statistical inferences about it, or even worse – revealing the vote choices of participants. A solution that removed trust in tallying authorities was for the first time proposed by Kiayias and Yung [KY02] in their privacy-preserving self-tallying boardroom voting protocol. A similar protocol was later proposed by Hao et al. [HRZ10], which was later extended to a blockchain environment by McCorry et al. [MSH17] in their Open Vote Network (OVN). An interesting property of OVN is that it requires only a single honest voting participant to maintain the privacy of the votes. However, OVN supports only two vote choices (based on [HRZ10]), assumes no stalling participants, requires expensive on-chain tally computation, and does not scale in the number of participants. The scalability of OVN was partially improved by Seifelnasr et al. [SGY20], but retaining the limitation of 2 choices and missing robustness.

In our first contribution within blockchain-based electronic voting, we introduce BBB-Voting [HLS23], a similar blockchain-based approach for decentralized voting such as OVN, but in contrast to OVN, BBB-Voting supports 1-out-of-k choices and provides robustness that enables recovery from stalling participants. We make a cost-optimized implementation using an Ethereum-based environment, which we compare with OVN and show that our work decreases the costs for voters by 13.5% in normalized gas consumption. Finally, we show how BBB-Voting can be extended to support the number of participants limited only by the expenses paid by the semi-trusted[1] authority and the computing power to obtain the tally.

In our second contribution, we introduce SBvote [SH23] (as an extension of BBB-Voting), a blockchain-based self-tallying voting protocol that is scalable in the number of voters, and therefore suitable for large-scale elections. The evaluation of our proof-of-concept implementation shows that the protocol's scalability is limited only by the underlying blockchain platform. Despite the limitations imposed by the throughput of the blockchain platforms, SBvote can accommodate elections with millions of voters. We evaluated the scalability of SBvote on two public smart contract platforms – Gnosis and Harmony.

In our last contribution, we propose Always on Voting (AoV) [VSH23] – a repetitive blockchain-based voting framework that allows participants to continuously vote and change elected candidates or policies without waiting for the next elections. Participants are permitted to privately change their vote at any point in time, while the effect of their change is manifested at the end of each epoch, whose duration is shorter than the time between two main elections. To thwart the problem of peak-end effect in epochs, the ends of epochs are randomized and made unpredictable, while preserved within soft bounds. In AoV, we make the synergy between a Bitcoin puzzle oracle, verifiable delay function, and smart contract properties to achieve these goals. AoV can be integrated with various existing blockchain-based e-voting solutions. More details in this direction of our research are presented in Chapter 6.

[1] The authority is only trusted to do identity management of participants honestly, which is an equivalent trust model as in OVN.

1.5 Secure Logging

Centralized ledger systems designed for secure logging are append-only databases providing immutability (i.e., tamper resistance) as a core property. To facilitate their append-only feature, cryptographic constructions, such as hash chains or hash trees, are usually deployed. Traditionally, public ledger systems are centralized, and controlled by a single entity that acts as a trusted party. In such a setting, ledgers are being deployed in various applications, including payments, logging, timestamping services, repositories, or public logs of various artifacts (e.g., keys [MBB$^+$15, CDGM19], certificates issued by authorities [LLK13], and binaries [FDP$^+$14]). Unfortunately, centralized ledgers have also several drawbacks, like a lack of efficient verifiability or a higher risk of censorship and equivocation.

In our first contribution to secure logging, we propose Aquareum [HS20], a framework for centralized ledgers mitigating their main limitations. Aquareum employs a trusted execution environment (TEE) and a public smart contract platform to provide verifiability, non-equivocation, and mitigation of censorship. In Aquareum, a ledger operator deploys a pre-defined TEE enclave code, which verifies the consistency and correctness of the ledger for every ledger update. Then, proof produced by the enclave is published at an existing public smart contract platform, guaranteeing that the given snapshot of the ledger is verified and no alternative snapshot of this ledger exists. Furthermore, whenever a client suspects that her query (or transaction) is censored, she can (confidentially) request a resolution of the query via the smart contract platform. The ledger operator noticing the query is obligated to handle it by passing the query to the enclave that creates a public proof of query resolution and publishes it using the smart contract platform. With such a censorship-evident design, an operator is publicly visible when misbehaving, thus the clients can take appropriate actions (e.g., sue the operator) or encode some automated service-level agreements into their smart contracts. Since Aquareum is integrated with a Turing-complete virtual machine, it allows arbitrary transaction processing logic, including tokens or client-specified smart contracts.

In our second contribution, we present CBDC-AquaSphere, a protocol that uses a combination of a trusted execution environment (TEE) and a public blockchain to enable interoperability over independent centralized CBDC ledgers (based on Aquareum). Our interoperability protocol uses a custom adaptation of atomic swap protocol and is executed by any pair of CBDC instances to realize a one-way transfer. It ensures features such as atomicity, verifiability, correctness, censorship resistance, and privacy while offering high scalability in terms of the number of CBDC instances. Our approach enables two possible deployment scenarios that can be combined: (1) CBDC instances represent central banks of multiple countries, and (2) CBDC instances represent the set of retail banks and a paramount central bank of a single country. More details in this direction of our research are presented in Chapter 7.

Paper	**Topic**	**Approach**	**Proofs**	**Implementation**	**Experiments**	**Security Analysis**	**Literature Review**	**Writing**
[BHS23]	▒	▒	×		▒	■	■	■
[DHV$^+$23]	■	■	×	■	■	■	▒	■
[HHR19]		▒	×	×	■	×	▒	■
[HHMH19]		▒	×	×	×	▒	▒	■
[HBH+20a]		■	▒	■	■	■	■	■
[HPH$^+$23]	■	■	▒	▒	▒	■	■	■
[HS20]		■	▒	■	■	■	■	■
[HVHS19]		■	×	×	×	■	■	■
[HVR+20]		■	×	×	×	■	■	■
[HTT$^+$20]			×		■	■	▒	■
[PBMH21]	■	■	×		■	■	▒	■
[PHMH23]		▒	×		▒	■	▒	■
[PBH+]	■	■	×		■	■	■	■
[SH23]	■	■	×			▒	▒	▒
[SRHS19b]		▒		■	■	■	■	
[HLS23]		▒	▒	■	■	■	■	■
[VSH23]		▒				■	■	■

Table 1.1: The author's contributions to the selected papers related to this thesis. Essential contribution is depicted in black, partial (still important) contribution is depicted in gray, minor or no contribution is depicted in white, and non-applicable field is depicted by a cross. The papers highlighted in bold are attached to this thesis.

1.6 Author's Contribution

In Table 1.1, we describe the author's contributions to the papers contained in this thesis.[2] Since there is no standard metric assessing the qualitative and quantitative contributions, the table describes the author's contribution to common parts in the process of creating a paper in computer science.

1.7 Organization of the Thesis

The rest of the thesis is organized as follows. In Chapter 2, we describe preliminaries and background related to this thesis. Next, in Chapter 3, we describe our contributions to the standardization of threat modeling for blockchains, and we introduce the security reference architecture as a layered model. Chapter 4 summarizes our contributions to the security of the consensus protocol in blockchains – in particular, we focus on Proof-of-

[2] Note that the table contains alphabetic ordering of papers by the author names.

Work (PoW) protocols: we describe StrongChain, transaction selection (incentive) attacks on DAG-based blockchains, and undercutting attacks on PoW blockchains. In Chapter 5, we deal with cryptocurrency wallets, where we describe our proposed classification of authentication schemes for such wallets as well as SmartOTPs, our contribution to the two-factor authentication on blockchains, and its security. Then, in Chapter 6, we focus on electronic voting using blockchains as an instance of a public bulletin board, and we describe our proposals BBB-Voting and SBvote as well as the always-on-voting framework for repetitive voting, accompanied with the analysis of their security and privacy aspects. Chapter 7 focuses on secure logging, where we present Aquareum, a centralized ledger based on blockchain and trusted computing; later in this chapter, we build on Aquareum and propose an interoperability protocol for central bank digital currencies called CBDC-AquaSphere. Chapter 8 concludes the paper and outlines our future research directions.

Chapter 2

Background

In this chapter, we summarize the background and preliminaries of the thesis. The reader familiar with the topics of blockchain, trusted computing, and integrity-preserving data structures can skip this chapter and proceed to Chapter 3. This chapter is based on the papers [HVHS19, HVR+20, SRHS19, VSH23, HS20, HPH$^+$23].

The blockchain is a data structure representing an append-only distributed ledger that consists of entries (a.k.a., transactions) aggregated within ordered blocks. The order of the blocks is agreed upon by mutually untrusting participants running a consensus protocol – these participants are also referred to as nodes. The blockchain is resistant against modifications by design since blocks are linked using a cryptographic hash function, and each new block has to be agreed upon by nodes running a consensus protocol.

A transaction is an elementary data entry that may contain arbitrary data, e.g., an order to transfer native cryptocurrency (i.e., crypto-tokens), a piece of application code (i.e., smart contract), the execution orders of such application code, etc. Transactions sent to a blockchain are validated by all nodes that maintain a replicated state of the blockchain.

2.1 Features of Blockchains

Blockchains were initially introduced as a means of coping with the centralization of monetary assets management, resulting in their most popular application – a decentralized cryptocurrency with a native crypto-token. Nevertheless, other blockchain applications have emerged, benefiting from features other than decentralization, e.g., privacy, energy efficiency, throughput, etc. We split the features of blockchains into inherent and non-inherent ones, where the former involves "traditional" features that were aimed to provide by all blockchains while the latter involves features specific to particular blockchain types. These features are summarized in the following.

2.1.1 Inherent Features

Decentralization: is achieved by a distributed consensus protocol – the protocol ensures that each modification of the ledger is a result of interaction among participants. In the consensus protocol, participants are equal, i.e., no single entity is designed as an authority. An important result of decentralization is resilience to node failures.

Censorship Resistance: is achieved due to decentralization, and it ensures that each valid transaction is processed and included in the blockchain.

Immutability: means that the history of the ledger cannot be easily modified – it requires a significant quorum of colluding nodes. The immutability of history is achieved by a cryptographic one-way function (i.e., a hash function) that creates integrity-preserving links between the previous record (i.e., block) and the current one. In this way, integrity-preserving chains (e.g., blockchains) or graphs (e.g., direct acyclic graphs [SLZ16, Tea18, Pop16] or trees [SZ13]) are built in an append-only fashion. However, the immutability of new blocks is not immediate and depends on the time to the finality of a particular consensus protocol (see Section 2.4).

Availability: although distributed ledgers are highly redundant in terms of data storage (i.e., full nodes store replicated data), the main advantage of such redundancy is paid off by the extremely high availability of the system. This feature may be of special interest to applications that cannot tolerate outages.

Auditability: correctness of each transaction and block recorded in the blockchain can be validated by any participating node, which is possible due to the publicly-known rules of a consensus protocol.

Transparency: the transactions stored in the blockchain as well as the actions of protocol participants are visible to other participants and in most cases even to the public.

2.1.2 Non-Inherent Features

Additionally to the inherent features, blockchains may be equipped with other features that aim to achieve extra goals. Below we list a few examples of such non-inherent features.

Energy Efficiency: running an open distributed ledger often means that scarce resources are wasted (e.g., Proof-of-Work). However, there are available consensus protocols that do not waste scarce resources, but instead emulate the consumption of scarce resources (i.e., Proof-of-Burn), or the interest rate on an investment (i.e., Proof-of-Stake). See examples of these protocols in Section 3.4.

Scalability: describes how the consensus protocol scales when the number of participants increases. Protocols whose behavior is not negatively affected by an increasing number of participants have high scalability.

Throughput: represents the number of transactions that can be processed per unit of time. Some consensus protocols have only a small throughput (e.g., Proof-of-Work), while others are designed with the intention to maximize throughput (e.g., Byzantine Fault Tolerant (BFT) protocols with a small number of participants).

Privacy & Anonymity: by design, data recorded on a public blockchain is visible to all nodes or public, which may lead to privacy and anonymity issues. Therefore, multiple solutions increasing anonymity (e.g., ring signatures [RST01] in Monero) and privacy (e.g., zk-SNARKs [BSCTV14] in Zcash) were proposed in the context of cryptocurrencies, while other efforts have been made in privacy-preserving smart

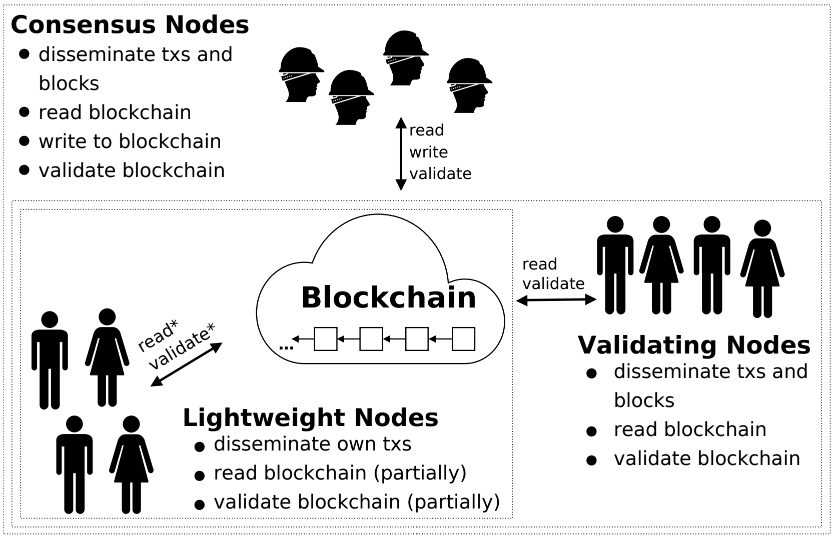


Figure 2.1: Involved parties with their interactions and hierarchy.

contract platforms [KMS+16, CZK+19a].

Accountability and Non-Repudiation: if blockchains or applications running on top of them are designed in such a way that identities of nodes (or application users) are known and verified, accountability and non-repudiation of actions performed can be provided too.

2.2 Involved Parties

Blockchains usually involve three native types of parties that can be organized into a hierarchy, according to the actions that they perform (see Figure 2.1):

(1) Consensus nodes (a.k.a., *miners* in Proof-of-Resource protocols) actively participate in the underlying consensus protocol. These nodes can read the blockchain and write to it by appending new transactions. Additionally, they can validate the blockchain and thus check whether writes of other consensus nodes are correct. Consensus nodes can prevent malicious behaviors (e.g., by not appending invalid transactions, or ignoring an incorrect chain).

(2) Validating nodes read the entire blockchain, validate it, and disseminate transactions. Unlike consensus nodes, validating nodes cannot write to the blockchain, and thus they cannot prevent malicious behaviors. On the other hand, they can detect malicious behavior since they possess copies of the entire blockchain.

(3) Lightweight nodes (a.k.a., clients or Simplified Payment Verification (SPV) clients) benefit from most of the blockchain functionalities, but they are equipped only with limited information about the blockchain. These nodes can read only fragments of the blockchain (usually block headers) and validate only a small number of transactions that concern them, while they rely on consensus and validating nodes. Therefore, they can detect only a limited set of attacks, pertaining to their own transactions.

Additional Involved Parties

Note that besides native types of involved parties, many applications using or running on the blockchain introduce their own (centralized) components.

2.3 Types of Blockchains

Based on how a new node enters a consensus protocol, we distinguish the following blockchain types:

Permissionless blockchains allow anyone to join the consensus protocol without permission. To prevent Sybil attacks, this type of blockchains usually requires consensus nodes to establish their identities by running a Proof-of-Resource protocol, where the consensus power of a node is proportional to its resources allocated.

Permissioned blockchains require a consensus node to obtain permission to join the consensus protocol from a centralized or federated authority(ies), while nodes usually have equal consensus power (i.e., one vote per node).

Semi-Permissionless blockchains require a consensus node to obtain some form of permission (i.e., stake) before joining the protocol; however, such permission can be given by any consensus node. The consensus power of a node is proportional to the stake that it has.

2.4 Design Goals of Consensus Protocols

2.4.1 Standard Design Goals – Liveness and Safety

The standard design goals of consensus protocols are *liveness* and *safety*. To meet these goals, an *eventual-synchrony* network model [DLS88] is usually assumed due to its simplicity. In this model, upper bounds are put on an asynchronous delivery of each message, hence each message is eventually/synchronously delivered. **Liveness** ensures that all valid transactions are eventually processed – i.e., if a transaction is received by a single honest node, it will eventually be delivered to all honest nodes. **Safety** ensures that if an honest node accepts (or rejects) a transaction, then all other honest nodes make the same decision. Usually, consensus protocols satisfy safety and liveness only under certain assumptions: the minimal fraction of honest consensus power or the maximal fraction of adversarial consensus power. With regard to safety, literature often uses the term *finality* and *time to finality*. **Finality** represents the sequence of the blocks from the genesis block up to the block B, where it can be assumed that this sequence of blocks is infeasible to overturn. To reach finality up to the block B, several successive blocks need to be appended after B – the number of such blocks is referred to as *the number of confirmations*.

2.4.2 Specific Design Goals

As a result of this study, we learned that standard design goals of the consensus protocol should be amended by specific design goals related to the type of the blockchain. In permissionless type, *elimination of Sybil entities*, *a fresh and fair leader/committee election*, and *non-interactive verification of the consensus result* is required to meet. In contrast, the (semi)-permissionless types do not require the elimination of Sybil entities.

2.4.3 Means to Achieve Design Goals

Simulation of the Verifiable Random Function (VRF). To ensure a fresh and fair leader/committee election, all consensus nodes should contribute to the pseudo-randomness generation that determines the fresh result of the election. This can be captured by the concept of the VRF [MRV99], which ensures the unpredictability and fairness of the election process. Therefore, the leader/committee election process can be viewed as a simulation of VRF [WHH$^+$19]. Due to the properties of VRF, the correctness of the election result can be verified non-interactively after the election took place.

Incentive and Rewarding Schemes. An important aspect for protocol designers is to include a rewarding/incentive scheme that motivates consensus nodes to participate honestly in the protocol. In the context of public (permissionless) blockchains that introduce their native crypto-tokens, this is achieved by block creation rewards as well as transaction fees, and optionally penalties for misbehavior. Transaction fees and block creation rewards are attributed to the consensus node(s) that create a valid block (e.g., [Nak08]), although alternative incentive schemes rewarding more consensus nodes at the same time are also possible (e.g., [SRHS19]). While transaction fees are included in a particular transaction, the block reward is usually part of the first transaction in the block (a.k.a., *coinbase* transaction).

2.5 Basis of Consensus Protocols

Lottery and *voting* are two marginal techniques that deal with the establishment of a consensus [Hyp17]. However, in addition to them, their combinations have become popular.

Lottery-Based Protocols. These protocols provide consensus by running a lottery that elects a leader/committee, who produces the block. The advantages of lottery-based approaches are a small network traffic overheads and high scalability (in the number of consensus nodes) since the process is usually non-interactive (e.g., [Nak08], [CXS$^+$17], [KRDO17]). However, a disadvantage of this approach is the possibility of multiple "winners" being elected, who propose conflicting blocks, which naturally leads to inconsistencies called *forks*. Forks are resolved by fork-choice rules, which compute the difficulty of each branch and select the one. For the *longest chain rule*, the chain with the largest number of blocks is selected in the case of a conflict, while for the *strongest* *chain rule*, the selection criteria involve the quality of each block in the chain (e.g., [SLZ16, SZ13, Tan17, ZSS$^+$18, SRHS19]). Note that the possibility of forks in this category of protocols causes an increase of the time to finality, which in turn might enable some attacks such as double-spending.

Voting-Based Protocols. In this group of protocols, the agreement on transactions is reached through the votes of all participants. Examples include Byzantine Fault Tolerant (BFT) protocols – which require the consensus of a majority quorum (usually $\frac{2}{3}$) of all consensus nodes (e.g., [CL$^+$99, AMQ13, BKM18, KR18, DMPZ14]). The advantage of this category is a low-latency finality due to a negligible likelihood of forks. The protocols from this group suffer from low scalability, and thus their throughput forms a trade-off with scalability (i.e., the higher the number of nodes, the lower the throughput).

Combinations. To improve the scalability of voting-based protocols, it is desirable to shrink the number of consensus nodes participating in the voting by a lottery, so that only nodes of such a committee vote for a block (e.g., [GHM$^+$17], [DPS19], [TH18], [ZIL17], [KR18]). Another option to reduce active voting nodes is to split them into several groups (a.k.a., *shards*) that run a consensus protocol in parallel (e.g., [KJG$^+$18, ZMR18]). Such a setting further increases the throughput in contrast to the single-group option, but on the other hand, it requires a mechanism that accomplishes inter-shard transactions.

2.6 Failure Models in Distributed Consensus Protocols

The relevant literature mentions two main failure models for consensus protocols [Sch90]:

Fail-Stop Failures: A node either stops its operation or continues to operate, while obviously exposing its faulty behavior to other nodes. Hence, all other nodes are aware of the faulty state of that node (e.g., tolerated in Paxos [L$^+$98], Raft [OO14], Viewstamped Replication [OL88]).

Byzantine Failures: In this model, the failed nodes (a.k.a., Byzantine nodes) may perform arbitrary actions, including malicious behavior targeting the consensus protocol and collusions with other Byzantine nodes. Hence, the Byzantine failure model is of particular interest to security-critical applications, such as blockchains (e.g., Nakamoto's consensus [Nak08], pure BFT protocols [CL$^+$99], [AMQ13], [BKM18], [CP02], and hybrid protocols [GHM$^+$17, KJG$^+$18, ZMR18]).

2.7 Nakamoto Consensus and Bitcoin

The Nakamoto consensus protocol allows decentralized and distributed network comprised of mutually distrusting participants to reach an agreement on the state of the global distributed ledger (i.e., blockchain) [Nak08]. To resolve any *forks* of the blockchain the protocol specifies to always accept the longest chain as the current one. Bitcoin is a peer-to-peer cryptocurrency that deploys Nakamoto consensus as its core mechanism to avoid double-spending. Transactions spending bitcoins are announced to the Bitcoin network, where miners validate, serialize all non-included transactions, and try to create (mine) a block of transactions with a PoW embedded into the block header. A valid block must fulfill the condition that for a cryptographic hash function H, the hash value of the block header is less than the target T.

2.7.1 Incentive Scheme

Brute-forcing the nonce (together with some other fields) is the only way to produce the PoW, which costs computational resources of the miners. To incentivize miners, the Bitcoin protocol allows the miner who finds a block to insert a coinbase transaction minting a specified amount of new bitcoins and collecting transaction fees offered by the included transactions. Currently, every block mints 6.25 new bitcoins. This amount is halved every four years, upper-bounding the number of bitcoins that will be created to a fixed total of 21 million coins. It implies that after around the year 2140, no new coins will be created, and the transaction fees will be the only source of reward for miners. Because of its design, Bitcoin is a deflationary currency.

In the original white paper, Nakamoto heuristically argues that the consensus protocol remains secure as long as a majority ($> 50\%$) of the participants' computing power honestly follow the rules specified by the protocol, which is compatible with their own economic incentives.

2.7.2 Difficulty and Fork-Choice Rule

The overall hash rate of the Bitcoin network and the difficulty of the PoW determine how long it takes to generate a new block for the whole network (the block interval). To stabilize the block interval at about 10 minutes for the constantly changing total mining power, the Bitcoin network adjusts the target T every 2016 blocks (about two weeks, i.e., a *difficulty window*) according to the following formula

$$T_{new} = T_{old} \cdot \frac{\textit{Time of the last 2016 blocks}}{\textit{2016} \cdot \textit{10 minutes}}. \tag{2.1}$$

In simple terms, the difficulty increases if the network is finding blocks faster than every 10 minutes, and decrease otherwise. With dynamic difficulty, Nakamoto's longest chain fork-choice rule was considered as a bug,[1] as it is trivial to produce long chains that have low difficulty. The rule was replaced by the strongest-PoW chain rule where competing chains are measured in terms of PoW they aggregated. As long as there is one chain with the highest PoW, this chain is chosen as the current one.

2.7.3 UTXO Model

Bitcoin introduced and uses the *unspent transaction output* (UTXO) model. The validity of a Bitcoin transaction is verified by executing a script proving that the transaction sender

[1] `https://goo.gl/thhusi`

is authorized to redeem unspent coins. Also, the Bitcoin scripting language offers a mechanism (`OP_RETURN`) for recording data on the blockchain, which facilitates third-party applications built-on Bitcoin.

2.7.4 Light Clients and Simple Payment Verification (SPV)

Bitcoin proposes the simplified payment verification (SPV) protocol, that allows resource-limited clients to verify that a transaction is indeed included in a block provided only with the block header and a short transaction's inclusion proof. The key advantage of the protocol is that SPV clients can verify the existence of a transaction without downloading or storing the whole block. SPV clients are provided only with block headers and on-demand request from the network inclusion proofs of the transactions they are interested in.

2.8 Integrity Preserving Data Structures

2.8.1 Merkle Tree

A Merkle tree [Mer89] is a data structure based on the binary tree in which each leaf node contains a hash of a single data block, while each non-leaf node contains a hash of its concatenated children. At the top of a Merkle tree is the root hash, which provides a tamper-evident summary of the contents. A Merkle tree enables efficient verification as to whether some data are associated with a leaf node by comparing the expected root hash of a tree with the one computed from a hash of the data in the query and the remaining nodes required to reconstruct the root hash (i.e., *proof* or *authentication path*). The reconstruction of the root hash has the logarithmic time and space complexity, which makes the Merkle tree an efficient scheme for membership verification. The Merkle tree is utilized for example in Bitcoin (and other blockchains) for aggregation of transactions within a block into the root hash, providing the integrity snapshot and at the same time enabling SPV clients to download only data related to authentication path of the transaction whose inclusion in a block is to be verified.

To provide a membership verification of element x_i in the list of elements $X = \{x_i\}, i \geq 1$, the Merkle tree supports the following operations:

- $\mathbf{MkRoot(X)} \rightarrow \mathbf{Root}$: an aggregation of all elements of the list X by a Merkle tree, providing a single value $Root$.

- $\mathbf{MkProof(x_i, X)} \rightarrow \pi^{\mathbf{mk}}$: a Merkle proof generation for the ith element x_i present in the list of all elements X.

- $\pi^{\mathbf{mk}}.\mathbf{Verify(x_i, Root)} \rightarrow \{\mathbf{True}, \mathbf{False}\}$: verification of the Merkle proof π^{mk}, witnessing that x_i is included in the list X that is aggregated by the Merkle tree with the root hash $Root$.

2.8.2 History Tree

A Merkle tree has been primarily used for proving membership. However, Crosby and Wallach [CW09] extended its application for an append-only tamper-evident log, denoted as a *history tree*. A history tree is the Merkle tree, in which leaf nodes are added in an append-only fashion, and which allows to produce logarithmic proofs witnessing that arbitrary two versions of the tree are consistent (i.e., one version of the tree is an extension of another). Therefore, once added, a leaf node cannot be modified or removed.

A history tree brings a versioned computation of hashes over the Merkle tree, enabling to prove that different versions (i.e., commitments) of a log, with distinct root hashes, make consistent claims about the past. To provide a tamper-evident history system [CW09], the log represented by the history tree L supports the following operations:

- $\mathbf{L.add(x) \rightarrow C_j}$: appending of the record x to L, returning a new commitment C_j that represents the most recent value of the root hash of the history tree.

- $\mathbf{L.IncProof(C_i, C_j) \rightarrow \pi^{inc}}$: an incremental proof generation between two commitments C_i and C_j, where $i \leq j$.

- $\mathbf{L.MemProof(i, C_j) \rightarrow \pi^{mem}}$: a membership proof generation for the record x_i from the commitment C_j, where $i \leq j$.

- $\mathbf{\pi^{inc}.Verify(C_i, C_j) \rightarrow \{True, False\}}$: verification of the incremental proof π^{inc}, witnessing that the commitment C_j contains the same history of records $x_k, k \in \{0, \dots, i\}$ as the commitment C_i, where $i \leq j$.

- $\mathbf{\pi^{mem}.Verify(i, x_i, C_j) \rightarrow \{True, False\}}$: verification of the membership proof π^{mem}, witnessing that x_i is the ith record in the jth version of L, fixed by the commitment C_j, $i \leq j$.

- $\mathbf{\pi^{inc}.DeriveNewRoot() \rightarrow C_j}$: a reconstruction of the commitment C_j from the incremental proof π^{inc} that was generated by $L.IncProof(C_i, C_j)$.

- $\mathbf{\pi^{inc}.DeriveOldRoot() \rightarrow C_i}$: a reconstruction of the commitment C_i from the incremental proof π^{inc} that was generated by $L.IncProof(C_i, C_j)$.

2.8.3 Radix and Merkle-Patricia Tries

Radix trie serves as a key-value storage. In the Radix trie, every node at the l-th layer of the trie has the form of $\langle (p_0, p_1, \dots, p_n), v \rangle$, where v is a stored value and all $p_i,\ i \in \{0, 1, \dots, n\}$ represent the pointers on the nodes in the next (lower) layer $l + 1$ of the trie, which is selected by following the $(l + 1)$-th item of the key. Note that key consists of an arbitrary number of items that belong to an alphabet with n symbols (e.g., hex symbols). Hence, each node of the Radix trie has n children and to access a leaf node (i.e., data v), one must descend the trie starting from the root node while following the items of the key one-by-one. Note that Radix trie requires underlying database of key-value storage that maps pointers to nodes. However, Radix trie does not contain integrity protection, and when its key is too long (e.g., hash value), the Radix trie will be sparse, thus imposing a high overhead for storage of all the nodes on the path from the root to values.

Merkle Patricia Trie (MPT) [Woo14a, Ray19] is a combination of the Merkle tree (see Section 2.8.1) and Radix trie data structures, and similar the Radix Trie, it serves as a key-value data storage. However, in contrast to Radix trie, the pointers are replaced by a cryptographically secure hash of the data in nodes, providing integrity protection. In detail, MPT guarantees integrity by using a cryptographically secure hash of the value for the MPT key as well as for the realization of keys in the underlying database that maps the hashes of nodes to their content; therefore, the hash of the root node of the MPT represents an integrity snapshot of the whole MPT trie. Next, Merkle-Patricia trie introduces the *extension nodes*, due to which, there is no need to keep a dedicated node for each item of the path in the key. The MPT trie T supports the following operations:

$\mathbf{T}.\mathbf{root} \rightarrow \mathbf{Root}$: accessing the hash of the root node of MPT, which is stored as a key in the underlying database.

- $\mathbf{T}.\mathbf{add}(\mathbf{k}, \mathbf{x}) \rightarrow \mathbf{Root}$: adding the value x with the key k to T while obtaining the new hash value of the root node.
- $\mathbf{T}.\mathbf{get}(\mathbf{k}) \rightarrow \{\mathbf{x}, \perp\}$: fetching a value x that corresponds to key k; return $\perp$ if no such value exists.
- $\mathbf{T}.\mathbf{delete}(\mathbf{k}) \rightarrow \{\mathbf{True}, \mathbf{False}\}$: deleting the entry with key equal to k, returning $True$ upon success, $False$ otherwise.
- $\mathbf{T}.\mathbf{MptProof}(\mathbf{k}) \rightarrow \{\pi^{\mathbf{mpt}}, \pi^{\overline{\mathbf{mpt}}}\}$: a MPT (inclusion / exclusion) proof generation for the entry with key k.
- $\pi^{\mathbf{mpt}}.\mathbf{Verify}(\mathbf{k}, \mathbf{Root}) \rightarrow \{\mathbf{True}, \mathbf{False}\}$: verification of the MPT proof π^{mpt}, witnessing that entry with the key k is in the MPT whose hash of the root node is equal to $Root$.
- $\pi^{\overline{\mathbf{mpt}}}.\mathbf{VerifyNeg}(\mathbf{k}, \mathbf{Root}) \rightarrow \{\mathbf{True}, \mathbf{False}\}$: verification of the negative MPT proof, witnessing that entry with the key k is not in the MPT with the root hash equal to $Root$.

2.9 Verifiable Delay Function

The functionality of Verifiable Delay Function (VDF) [BBBF18] is similar to a time lock,[2] but in addition to it, by providing a short proof, a verifier may easily check if the prover knows the output of the VDF. The function is effectively serialized, and parallel processing does not help to speed up VDF computation. A moderate amount of sequential computation is required to compute VDF. Given a time delay t, a VDF must satisfy

[2] Time locks are computational problems that can only be solved by running a continuous computation for a given amount of time.

the following conditions: for any input x, anyone equipped with commercial hardware can find y = VDF(x, t) in t sequential steps, but an adversary with p parallel processing units must not distinguish y from a random number in significantly fewer steps. Further, given output y of VDF, the prover can supply a proof π to a verifier, who may check the output $y \ = \ $ VDF(x, t) using π in logarithmic time w.r.t. time delay t (i.e., $VDF\_Verify(y, \pi) \stackrel{?}{=} True$).

Finally, the safety factor A_{max} is defined as the time ratio that the adversary is estimated to run VDF computation faster on proprietary hardware as opposed to a benign VDF computation using commercial hardware (see Drake [Dra18]). CPU over-clocking records [SF20] indicate that $A_{max} = 10$ is a reasonable estimate.

2.10 Atomic Swap

A basic atomic swap assumes two parties $\mathbb{A}$ and $\mathbb{B}$ owning crypto-tokens in two different blockchains. $\mathbb{A}$ and $\mathbb{B}$ wish to execute cross-chain exchange atomically and thus achieve a *fairness* property, i.e., either both of the parties receive the agreed amount of crypto-tokens or neither of them. First, this process involves an agreement on the amount and exchange rate, and second, the execution of the exchange itself.

In a centralized scenario [Mic03], the approach is to utilize a trusted third party for the execution of the exchange. In contrast to the centralized scenario, blockchains allow us to execute such an exchange without a requirement of the trusted party. The atomic swap protocol [Bit18c] enables conditional redemption of the funds in the first blockchain to $\mathbb{B}$ upon revealing of the hash pre-image (i.e., secret) that redeems the funds on the second blockchain to $\mathbb{A}$. The atomic swap protocol is based on two Hashed Time-Lock Contracts (HTLC) that are deployed by both parties in both blockchains.

Although HTLCs can be implemented by Turing-incomplete smart contracts with support for hash-locks and time-locks, for clarity, we provide a description assuming Turing-complete smart contracts, requiring four transactions:

1. $\mathbb{A}$ chooses a random string x (i.e., a secret) and computes its hash $h(x)$. Using $h(x)$, $\mathbb{A}$ deploys $HTLC_{\mathbb{A}}$ on the first blockchain and sends the agreed amount to it, which later enables anybody to do a conditional transfer of that amount to $\mathbb{B}$ upon calling a particular method of $HTLC_{\mathbb{A}}$ with $x = h(x)$ as an argument (i.e., hash-lock). Moreover, $\mathbb{A}$ defines a time-lock, which, when expired, allows $\mathbb{A}$ to recover funds into her address by calling a dedicated method: this is to prevent aborting of the protocol by another party.

2. When $\mathbb{B}$ notices that $HTLC_{\mathbb{A}}$ has been already deployed, she deploys $HTLC_{\mathbb{B}}$ on the second blockchain and sends the agreed amount there, enabling a conditional transfer of that amount to $\mathbb{A}$ upon revealing the correct pre-image of $h(x)$ ($h(x)$ is visible from already deployed $HTLC_{\mathbb{A}}$). $\mathbb{B}$ also defines a time-lock in $HTLC_{\mathbb{B}}$ to handle abortion by $\mathbb{A}$.

3. Once $\mathbb{A}$ notices deployed $HTLC_{\mathbb{B}}$, she calls a method of $HTLC_{\mathbb{B}}$ with revealed x, and in turn, she obtains the funds on the second blockchain.

4. Once $\mathbb{B}$ notices that x was revealed by $\mathbb{A}$ on the second blockchain, she calls a method of $HTLC_{\mathbb{A}}$ with x as an argument, and in turn, she obtains the funds on the first blockchain.

If any of the parties aborts, the counter-party waits until the time-lock expires and redeems the funds.

2.11 Trusted Execution Environment

Trusted Execution Environment (TEE) is a hardware-based component that can securely execute arbitrary code in an isolated environment. TEE uses cryptography primitives and hardware-embedded secrets that protect data confidentiality and the integrity of computations. In particular, the adversary model of TEE usually includes privileged applications and an operating system, which may compromise unprivileged user-space applications. There are several practical instances of TEE, such as Intel Software Guard Extensions (SGX) [AGJS13, MAB$^+$13, HLP$^+$13] available at Intel's CPUs or based on RISC-V architecture such as Keystone-enclave [Enc19] and Sanctum [CLD16]. In the context of this work (i.e., Chapter 7), we built on top of Intel SGX, therefore we adopt the terminology introduced by it.

Intel SGX. Intel SGX is a set of instructions that ensures hardware-level isolation of protected user-space codes called *enclaves*. An enclave process cannot execute system calls but can read and write memory outside the enclave. Thus isolated execution in SGX may be viewed as an ideal model in which a process is guaranteed to be executed correctly with ideal confidentiality, while it might run on a potentially malicious operating system.

Intel SGX allows a local process or a remote system to securely communicate with the enclave as well as execute verification of the integrity of the enclave's code. When an enclave is created, the CPU outputs a report of its initial state, also referred to as a *measurement*, which is signed by the private key of TEE and encrypted by a public key of Intel Attestation Service (IAS). The hardware-protected signature serves as the proof that the measured code is running in an SGX-protected enclave, while the encryption by IAS public key ensures that the SGX-equipped CPU is genuine and was manufactured by Intel. This proof is also known as a *quote* or *attestation*, and it can be verified by a local process or by a remote system. The enclave process-provided public key can be used by a verifier to establish a secure channel with the enclave or to verify the signature during the attestation. We assume that a trustworthy measurement of the enclave's code is available for any client that wishes to verify an attestation.

2.12 Central Bank Digital Currency (CBDC)

CBDC is often defined as a digital liability backed and issued by a central bank that is widely available to the general public. CBDC encompasses many potential benefits such as efficiency and resiliency, flexible monetary policies, and enables enhanced control of tax evasion and money laundering [KAD$^+$20]. However, regulations, privacy and identity management issues, as well as design vulnerabilities are potential risks that are shared with cryptocurrencies. Many blockchain-based CBDC projects rely on using some sort of stable coins adapting permissioned blockchains due to their scalability and the capability to establish specific privacy policies, as compared to public blockchains [SI21, ZH21]. Therefore, the level of decentralization and coin volatility are two main differences between blockchain-based CBDCs and common cryptocurrencies. These CBDCs are often based on permissioned blockchain projects such as Corda [BCGH16], variants of Hyperledger [Hyp22], and Quorum [EKR17].

CDBC solutions are often designed as multi-layer projects [JX22]. Wholesale CBDC targets communication of financial institutions and inter-bank settlements. Retail CBDC includes accessibility to the general public or their customers.

Chapter 3

Standardization in Threat Modeling

In this chapter, we present our contribution to standardization for threat modeling and it is based on the papers [HVHS19] [HVR+20] (see also Section 3.9). In particular, we introduce the security reference architecture (SRA) for blockchains, which adopts a stacked model (similar to the ISO/OSI) describing the nature and hierarchy of various security and privacy aspects. The SRA contains four layers: (1) the network layer, (2) the consensus layer, (3) the replicated state machine layer, and (4) the application layer. At each of these layers, we identify known security threats, their origin, and countermeasures, while we also analyze several cross-layer dependencies. Next, to enable better reasoning about the security aspects of blockchains by the practitioners, we propose a blockchain-specific version of the threat-risk assessment standard ISO/IEC 15408 by embedding the stacked model into this standard. Finally, we provide designers of blockchain platforms and applications with a design methodology following the model of SRA and its hierarchy.

3.1 Methodology and Scope

We aim to consolidate the literature, categorize found vulnerabilities and threats according to their origin, and as a result, we create four main categories (also referred to as layers). At the level of particular main categories, we apply sub-categorization that is based on the existing knowledge and operation principles specific to such subcategories, especially concerning the security implications. If some subcategories impose equivalent security implications, we merge them into a single subcategory. See the road-map of all the categories in Figure 3.1. Our next aim is to indicate and explain the co-occurrences or relations of multiple threats, either at the same main category or across more categories.

3.2 Security Reference Architecture

We present two models of the security reference architecture, which facilitate systematic studying of vulnerabilities and threats related to the blockchains and applications running on top of them. First, we introduce the stacked model, which we then project into the threat-risk assessment model.

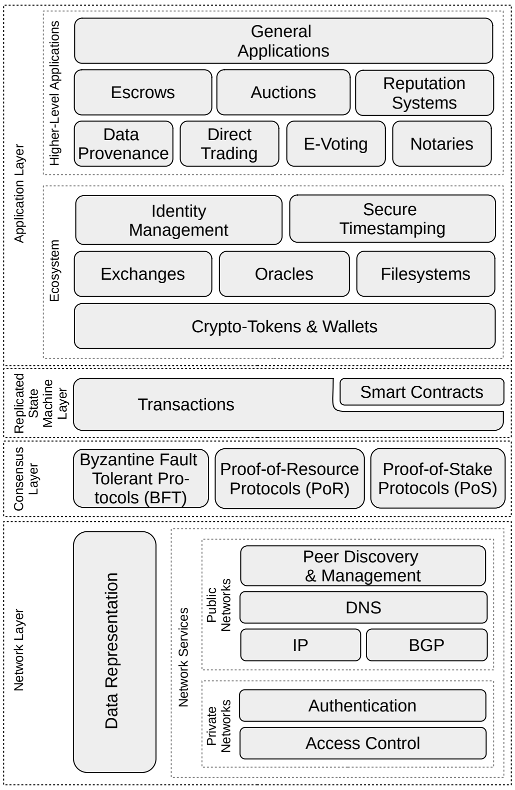


Figure 3.1: Stacked model of the security reference architecture.

3.2.1 Stacked Model

To classify the security aspects of blockchains, we utilize a stacked model consisting of four layers (see Figure 3.1). A similar stacked model was already proposed in the literature [WHH$^+$19], but in contrast to it, we preserve only such a granularity level that enables us to isolate security threats and their nature, which is the key focus of our work. In the following, we briefly describe each layer.

(1) The network layer consists of the data representation and network services planes. The data representation plane deals with the storage, encoding, and protection of data, while the network service plane contains the discovery and communication with protocol peers, addressing, routing, and naming services.

(2) The consensus layer deals with the ordering of transactions, and we divide it into three main categories according to the protocol type: Byzantine Fault Tolerant, Proof-of-Resource, and Proof-of-Stake protocols.

(3) The replicated state machine (RSM) layer deals with the interpretation of transactions, according to which the state of the blockchain is updated. In this layer, transactions are categorized into two parts, where the first part deals with the privacy of data in transactions as well as the privacy of the users who created them, and the second part – smart contracts – deals with the security and safety aspects of decentralized code execution in this environment.

(4) The application layer contains the most common end-user functionalities and services. We divide this layer into two groups. The first group represents the applications that provide common functionalities for most of the higher-level blockchain applications, and it contains the following categories: wallets, exchanges, oracles, filesystems, identity management, and secure timestamping. We refer to this group as applications of the blockchain ecosystem. The next group of application types resides at a higher level and focuses on providing certain end-user functionality. This group contains categories such as e-voting, notaries, identity management, auctions, escrows, etc.

3.2.2 Threat-Risk Assessment Model

To better capture the security-related aspects of blockchain systems, we introduce a threat-risk model (see Figure 3.2) that is based on the template of ISO/IEC 15408 [Com17] and projection of our stacked model (see Figure 3.1). This model includes the following components and actors:

Owners are blockchain users who run any type of node and they exist at the application layer and the consensus layer. Owners possess crypto-tokens, and they might use or provide blockchain-based applications and services. Additionally, owners involve consensus nodes that earn crypto-tokens from running the consensus protocol.

Assets are present at the application layer, and they consist of monetary value (i.e., crypto-tokens or other tokens) as well as the availability of application-layer services and functionalities built on top of blockchains (e.g., notaries, escrows, data provenance, auctions). The authenticity of users, the privacy of users, and the privacy of data might also be considered as application-specific assets. Furthermore, we include here the reputation of service providers using the blockchain services.

Threat agents are spread across all the layers of the stacked model, and they mostly involve malicious users whose intention is to steal assets, break functionalities, or disrupt services. However, threat agents might also be inadvertent entities, such as developers of smart contracts who unintentionally create bugs and designers of blockchain applications who make mistakes in the design or ignore some issues.

Threats facilitate various attacks on assets, and they exist at all layers of the stacked model. Threats arise from vulnerabilities in the network, smart contracts, applications, from consensus protocol deviations, violations of consensus protocol assumptions.

Countermeasures protect owners from threats by minimizing the risk of compromising/losing the assets. Alike the threats and threat agents, countermeasures can be applied at each of the layers of our stacked model, and they involve various security/privacy/safety solutions, incentive schemes, reputation techniques, best practices, etc. Nevertheless, we emphasize that their utilization usually imposes some limitations such as higher complexity and additional performance overheads (e.g., resulting in decreased throughput).

Risks are related to the application layer, and they are caused by threats and their agents. Risks may lead to a loss of monetary assets, a loss of privacy, a loss of reputation, service malfunctions, and disruptions of services and applications (i.e., availability issues).

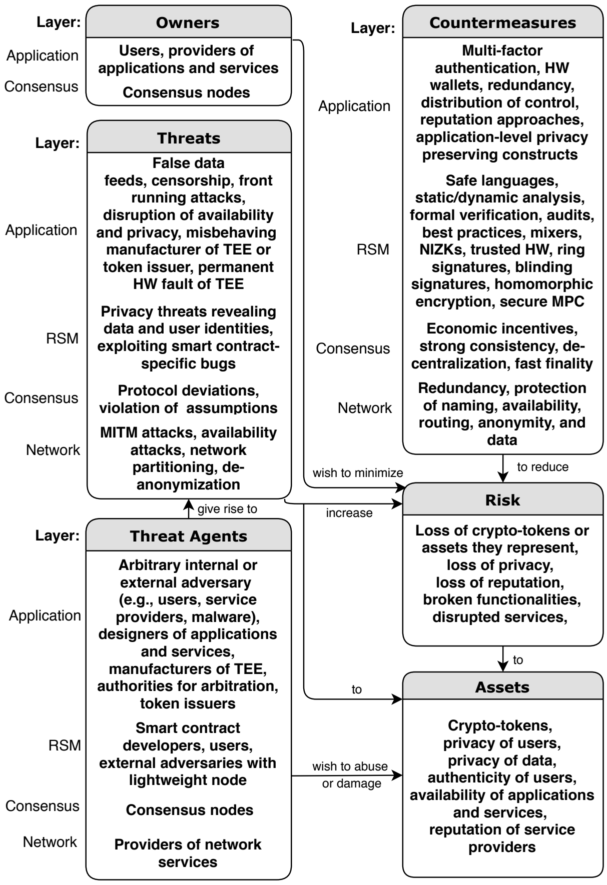

Layer:
Owners
Application
Users, providers of applications and services
Consensus
Consensus nodes
Layer:
Threats
Application
False data feeds, censorship, front running attacks, disruption of availability and privacy, misbehaving manufacturer of TEE or token issuer, permanent HW fault of TEE
RSM
Privacy threats revealing data and user identities, exploiting smart contract-specific bugs
Consensus
Protocol deviations, violation of assumptions
Network
MITM attacks, availability attacks, network partitioning, de-anonymization
give rise to
Layer:
Threat Agents
Application
Arbitrary internal or external adversary (e.g., users, service providers, malware), designers of applications and services, manufacturers of TEE, authorities for arbitration, token issuers
RSM
Smart contract developers, users, external adversaries with lightweight node
Consensus
Consensus nodes
Network
Providers of network services
Layer:
Countermeasures
Application
Multi-factor authentication, HW wallets, redundancy, distribution of control, reputation approaches, application-level privacy preserving constructs
RSM
Safe languages, static/dynamic analysis, formal verification, audits, best practices, mixers, NIZKs, trusted HW, ring signatures, blinding signatures, homomorphic encryption, secure MPC
Consensus
Economic incentives, strong consistency, de-centralization, fast finality
Network
Redundancy, protection of naming, availability, routing, anonymity, and data
to reduce
wish to minimize
increase
Risk
Loss of crypto-tokens or assets they represent, loss of privacy, loss of reputation, broken functionalities, disrupted services,
to
to
Assets
Crypto-tokens, privacy of users, privacy of data, authenticity of users, availability of applications and services, reputation of service providers
wish to abuse or damage

Figure 3.2: Threat-risk assessment model of the security reference architecture.

The owners wish to minimize the risk caused by threats that arise from threat agents. Within our stacked model, different threat agents appear at each layer. At the **network layer**, there are service providers including parties managing IP addresses and DNS names. The threats at this layer arise from man-in-the-middle (MITM) attacks, network partitioning, de-anonymization, and availability attacks. Countermeasures contain protection of availability, naming, routing, anonymity, and data. At the **consensus layer**, consensus nodes may be malicious and wish to alter the outcome of the consensus protocol by deviating from it. Moreover, if they are powerful enough, malicious nodes might violate assumptions of consensus protocols to take over the execution of the protocol or cause its disruption. The countermeasures include well-designed economic incentives, strong consistency, decentralization, and fast finality solutions. At the **RSM layer**, the threat agents may stand for developers who (un)intentionally introduce semantic bugs in smart contracts (intentional bugs represent backdoors) as well as users and external adversaries running lightweight nodes who pose threats due to the exploitation of such bugs. Countermeasures include safe languages, static/dynamic analysis, formal verification, audits, best practices, and design patterns. Other threats of the RSM layer are related to compromising the privacy of data and user identities with mitigation techniques involving mixers, privacy-preserving cryptography constructs (e.g., non-interactive zero-knowledge proofs (NIZKs), ring signatures, blinding signatures, homomorphic encryption) as well as usage of trusted hardware (respecting its assumptions and attacker models declared). At the **application layer**, threat agents are broad and involve arbitrary internal or external adversaries such as users, service providers, malware, designers of applications and services, manufactures of trusted execution environments (TEE) for concerned applications (e.g., oracles, auctions), authorities in the case of applications that require them for arbitration (e.g., escrows, auctions) or filtering of users (e.g., e-voting, auctions), token issuers. The threats on this layer might arise from false data feeds, censorship by application-specific authorities (e.g., auctions, e-voting), front running attacks, disruption of the availability of centralized components, compromising application-level privacy, misbehaving of the token issuer, misbehaving of manufacturer of TEE or permanent hardware (HW) faults in TEE. Examples of mitigation techniques are multi-factor authentication, HW wallets with displays for signing transactions, redundancy/distributions of some centralized components, reputation systems, and privacy preserving-constructs as part of the applications themselves. We elaborate closer on vulnerabilities, threats, and countermeasures (or mitigation techniques) related to each layer of the stacked model in the following sections.

Involved Parties & Blockchain's Life-Cycle. In Chapter 2, we presented several types of involved parties in the blockchain infrastructure (see Figure 2.1). We emphasize that these parties are involved in the operational stage of the blockchain's life-cycle. However, in the design and development stages of the blockchain's life-cycle, programmers and designers should also be considered as potential threat agents who influence the security aspects of the whole blockchain infrastructure (regardless of whether their intention is malicious or not). This is of great concern especially for applications built on top of blockchains (i.e., at the application layer) since these applications are usually not thoroughly reviewed by the community or public, as it is typical for other (lower) layers.

3.3 Network Layer

Blockchains usually introduce peer-to-peer overlay networks built on top of other networks. Hence, blockchains inherit security and privacy issues from their underlying networks. In our model (see Figure 3.1), we divide the network layer into *data representation* and *network services* sub-planes. The data representation plane is protected

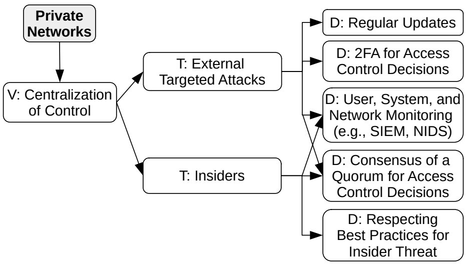

Private Networks
V: Centralization of Control
T: External Targeted Attacks
T: Insiders
D: Regular Updates
D: 2FA for Access Control Decisions
D: User, System, and Network Monitoring (e.g., SIEM, NIDS)
D: Consensus of a Quorum for Access Control Decisions
D: Respecting Best Practices for Insider Threat

Figure 3.3: Vulnerabilities, threats, and defenses in private networks (network layer).

by cryptographic primitives that ensure data integrity, user authentication, and optionally confidentiality, privacy, anonymity, non-repudiation, and accountability. The main services provided by the network layer are peer management and discovery, which rely on the internals of the underlying network, such as domain name resolution (i.e., DNS) or network routing protocols. Based on permission to join the blockchain system, the networks are either private or public. We model security threats and mitigation techniques for both private and public networks as vulnerability/threat/defense (VTD) graphs in Figure 3.3 and Figure 3.4, and we refer the interested reader to our paper [HVR+20] for more details.

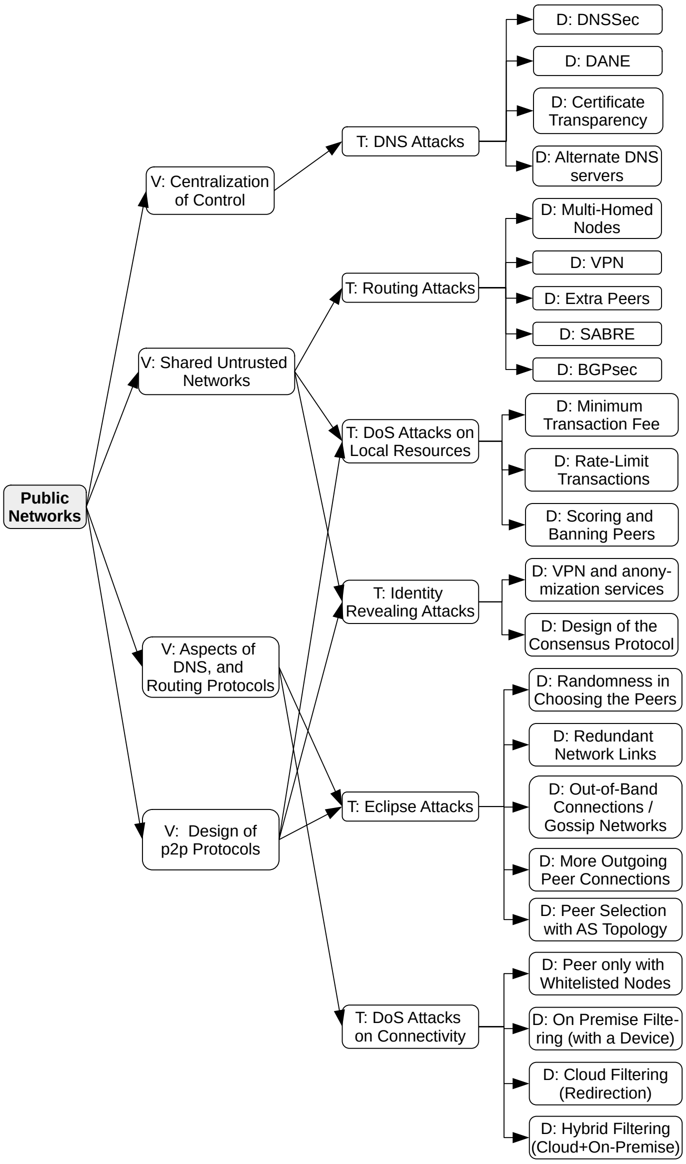

Public Networks
V: Centralization of Control
V: Shared Untrusted Networks
V: Aspects of DNS, and Routing Protocols
V: Design of p2p Protocols
T: DNS Attacks
T: Routing Attacks
T: DoS Attacks on Local Resources
T: Identity Revealing Attacks
T: Eclipse Attacks
T: DoS Attacks on Connectivity
D: DNSSec
D: DANE
D: Certificate Transparency
D: Alternate DNS servers
D: Multi-Homed Nodes
D: VPN
D: Extra Peers
D: SABRE
D: BGPsec
D: Minimum Transaction Fee
D: Rate-Limit Transactions
D: Scoring and Banning Peers
D: VPN and anonymization services
D: Design of the Consensus Protocol
D: Randomness in Choosing the Peers
D: Redundant Network Links
D: Out-of-Band Connections / Gossip Networks
D: More Outgoing Peer Connections
D: Peer Selection with AS Topology
D: Peer only with Whitelisted Nodes
D: On Premise Filtering (with a Device)
D: Cloud Filtering (Redirection)
D: Hybrid Filtering (Cloud+On-Premise)

Figure 3.4: Vulnerabilities, threats, and defenses in public networks (network layer).

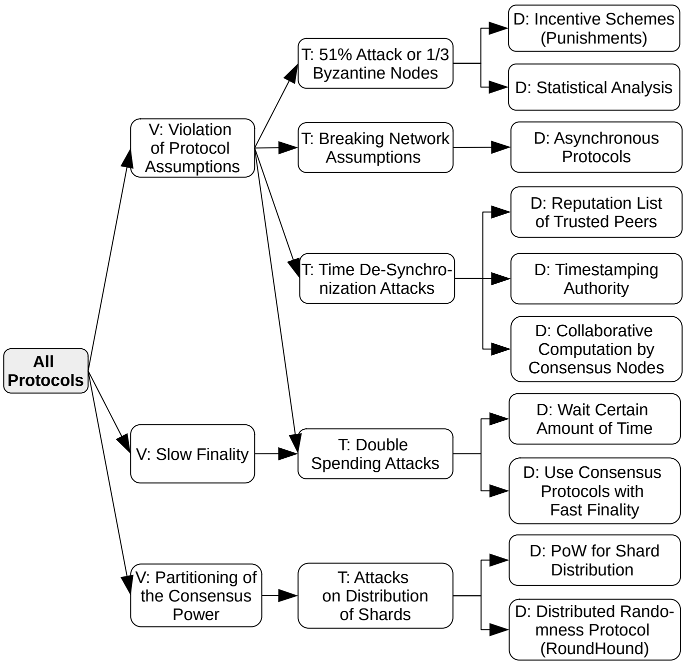

All Protocols
V: Violation of Protocol Assumptions
V: Slow Finality
V: Partitioning of the Consensus Power
T: 51% Attack or 1/3 Byzantine Nodes
T: Breaking Network Assumptions
T: Time De-Synchro-nization Attacks
T: Double Spending Attacks
T: Attacks on Distribution of Shards
D: Incentive Schemes (Punishments)
D: Statistical Analysis
D: Asynchronous Protocols
D: Reputation List of Trusted Peers
D: Timestamping Authority
D: Collaborative Computation by Consensus Nodes
D: Wait Certain Amount of Time
D: Use Consensus Protocols with Fast Finality
D: PoW for Shard Distribution
D: Distributed Rando-mness Protocol (RoundHound)

Figure 3.5: Generic threats and defenses of the consensus layer.

3.4 Consensus Layer

The consensus layer of the stacked model deals with the ordering of transactions, while the interpretation of them is left for the RSM layer (see Section 3.5). The consensus layer includes three main categories of consensus protocols concerning different principles of operation and thus their security aspects – Proof-of-Resource Protocols (PoR), Byzantine Fault Tolerant (BFT) Protocols, and Proof-of-Stake Protocols (PoS). Nevertheless, we can identify vulnerabilities and threats that are generic to all categories. Next, we outline modeling of security threats and mitigation techniques generic to all consensus protocols as VTD graphs in Figure 3.5, while particular categories of protocols are modeled in Figure 3.7, Figure 3.8, and Figure 3.6. For details about these categories and their threats, we refer the interested reader to our paper [HVR+20].

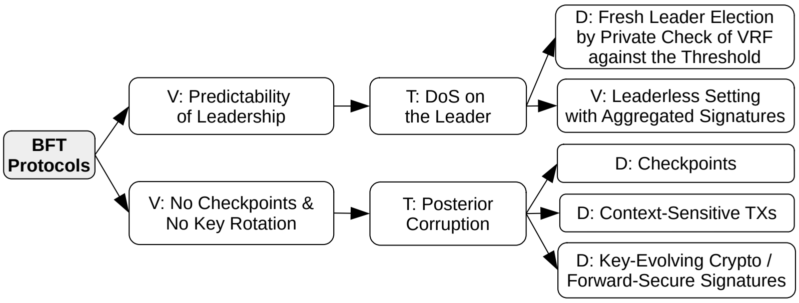

BFT Protocols
V: Predictability of Leadership
V: No Checkpoints & No Key Rotation
T: DoS on the Leader
T: Posterior Corruption
D: Fresh Leader Election by Private Check of VRF against the Threshold
V: Leaderless Setting with Aggregated Signatures
D: Checkpoints
D: Context-Sensitive TXs
D: Key-Evolving Crypto / Forward-Secure Signatures

Figure 3.6: Vulnerabilities, threats, and defenses of BFT protocols (consensus layer).

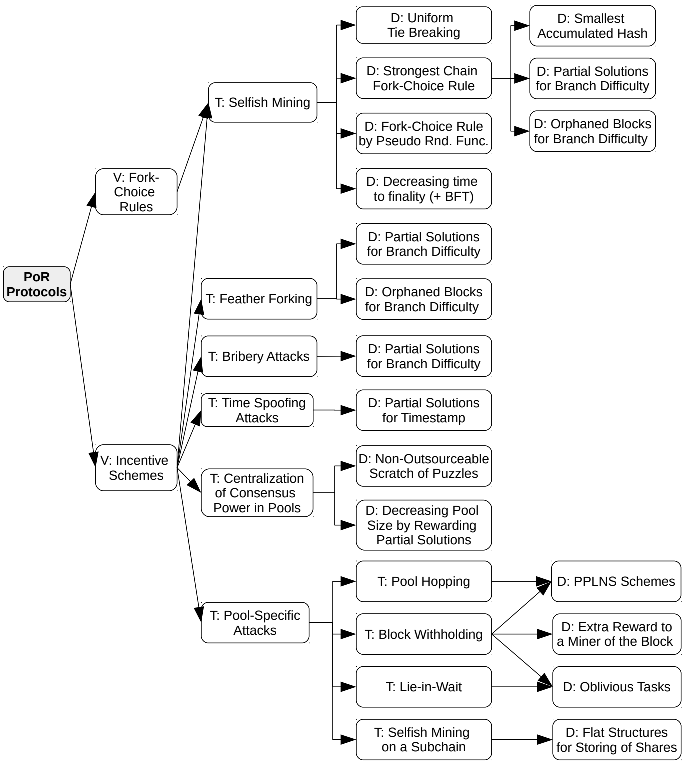

PoR Protocols
V: Fork-Choice Rules
V: Incentive Schemes
T: Selfish Mining
D: Uniform Tie Breaking
D: Strongest Chain Fork-Choice Rule
D: Fork-Choice Rule by Pseudo Rnd. Func.
D: Decreasing time to finality (+ BFT)
D: Smallest Accumulated Hash
D: Partial Solutions for Branch Difficulty
D: Orphaned Blocks for Branch Difficulty
T: Feather Forking
D: Partial Solutions for Branch Difficulty
D: Orphaned Blocks for Branch Difficulty
T: Bribery Attacks
D: Partial Solutions for Branch Difficulty
T: Time Spoofing Attacks
D: Partial Solutions for Timestamp
T: Centralization of Consensus Power in Pools
D: Non-Outsourceable Scratch of Puzzles
D: Decreasing Pool Size by Rewarding Partial Solutions
T: Pool-Specific Attacks
T: Pool Hopping
T: Block Withholding
T: Lie-in-Wait
T: Selfish Mining on a Subchain
D: PPLNS Schemes
D: Extra Reward to a Miner of the Block
D: Oblivious Tasks
D: Flat Structures for Storing of Shares

Figure 3.7: Vulnerabilities, threats, and defenses of PoR protocols (consensus layer).

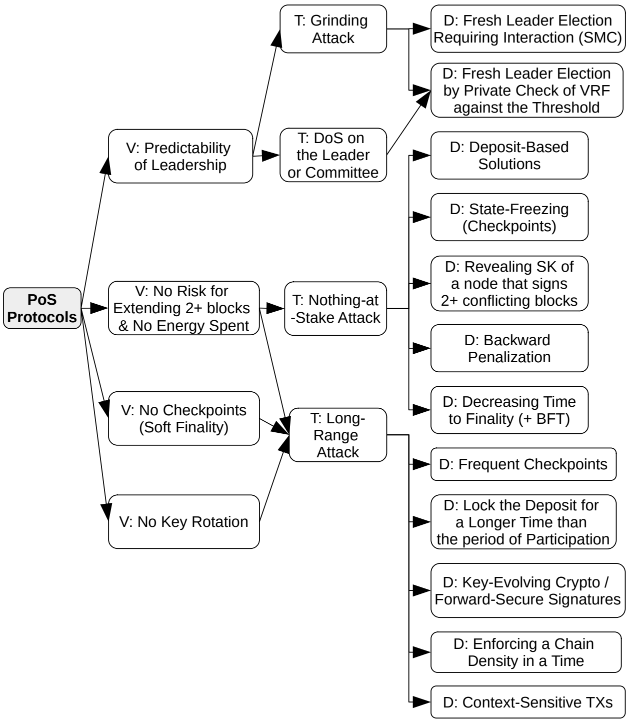

PoS Protocols
V: Predictability of Leadership
V: No Risk for Extending 2+ blocks & No Energy Spent
V: No Checkpoints (Soft Finality)
V: No Key Rotation
T: Grinding Attack
T: DoS on the Leader or Committee
T: Nothing-at -Stake Attack
T: Long-Range Attack
D: Fresh Leader Election Requiring Interaction (SMC)
D: Fresh Leader Election by Private Check of VRF against the Threshold
D: Deposit-Based Solutions
D: State-Freezing (Checkpoints)
D: Revealing SK of a node that signs 2+ conflicting blocks
D: Backward Penalization
D: Decreasing Time to Finality (+ BFT)
D: Frequent Checkpoints
D: Lock the Deposit for a Longer Time than the period of Participation
D: Key-Evolving Crypto / Forward-Secure Signatures
D: Enforcing a Chain Density in a Time
D: Context-Sensitive TXs

Figure 3.8: Vulnerabilities, threats, and defenses of PoS protocols (consensus layer).

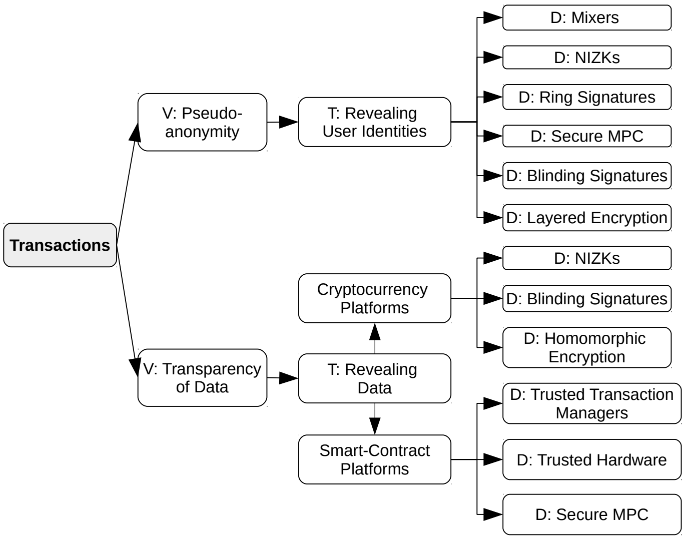


Figure 3.9: Vulnerabilities, threats, and defenses of privacy threats (RSM layer).

3.5 Replicated State Machine Layer

The Replicated State Machine (RSM) layer is responsible for the interpretation and execution of transactions that are already ordered by the consensus layer. Concerning security threats for this layer are related to the privacy of users, privacy and confidentiality of data, and smart contract-specific bugs. We split the security threats of the RSM layer into two parts: standard transactions and smart contracts.

3.5.1 Transaction Protection

Transactions containing plain-text data are digitally signed by private keys of users, enabling anybody to verify the validity of transactions with the corresponding public keys. However, such an approach provides only pseudonymous identities that can be traced to real IP addresses (and sometimes to identities) by a network-eavesdropping adversary, and moreover, it does not ensure the confidentiality of data [FHZ$^+$19]. Therefore, several blockchain-embedded mechanisms for the privacy of data and user identities were proposed in the literature, which we review in [HVR+20]. Note that some privacy-preserving techniques can be applied also on the application layer of our stacked model but imposing higher programming overheads and costs, which is common in the case of blockchain platforms that do not support them natively. We outline modeling of security threats and mitigation techniques related to transactions and their privacy as VTD graphs in Figure 3.9. For details of particular vulnerabilities and threats, we refer the interested reader to our paper [HVR+20].

3.5.2 Smart Contracts

Smart contracts introduced to automate legal contracts, now serve as a method for building decentralized applications on blockchains. They are usually written in a blockchain-

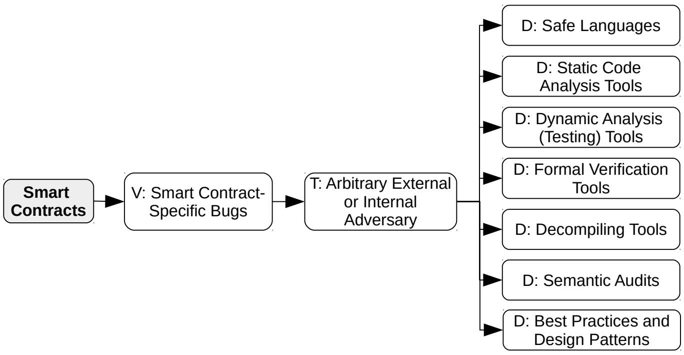

Smart Contracts
V: Smart Contract-Specific Bugs
T: Arbitrary External or Internal Adversary
D: Safe Languages
D: Static Code Analysis Tools
D: Dynamic Analysis (Testing) Tools
D: Formal Verification Tools
D: Decompiling Tools
D: Semantic Audits
D: Best Practices and Design Patterns

Figure 3.10: Vulnerabilities, threats, and defenses of smart contract platforms (RSM layer).

specific programming language that may be Turing-complete (i.e., contain arbitrary programming logic) or only serve for limited purposes. We outline modeling of security threats and mitigation techniques related to smart contracts as VTD graphs in Figure 3.9. For details of particular vulnerabilities and threats, we refer the interested reader to our paper [HVR+20].

3.6 Application Layer: Ecosystem Applications

We present a functionality-oriented categorization of the applications running on or utilizing the blockchain in Figure 3.11, where we depict hierarchy in the inheritance of security aspects among particular categories. In this categorization, we divide the applications into categories according to the main functionality/goal that is to be achieved by using the blockchain. Security threats of this layer are mostly specific to particular types of applications. Nevertheless, there are a few application-level categories that are often utilized by other higher-level applications. In the current section, we isolate such categories into a dedicated application-level group denoted as an *ecosystem*, while we cover the rest of the applications in Section 3.7. The group of ecosystem applications contains five categories, and we outline their security threats and mitigation techniques in VTD graphs as follows:

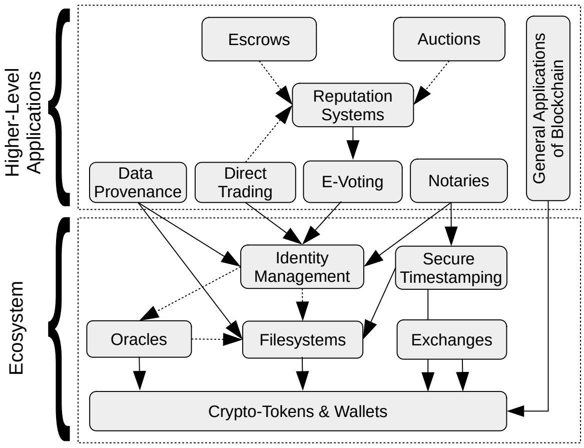


Figure 3.11: Hierarchy in inheritance of security aspects across categories of the application layer. Dotted arrows represent application-specific and optional dependencies.

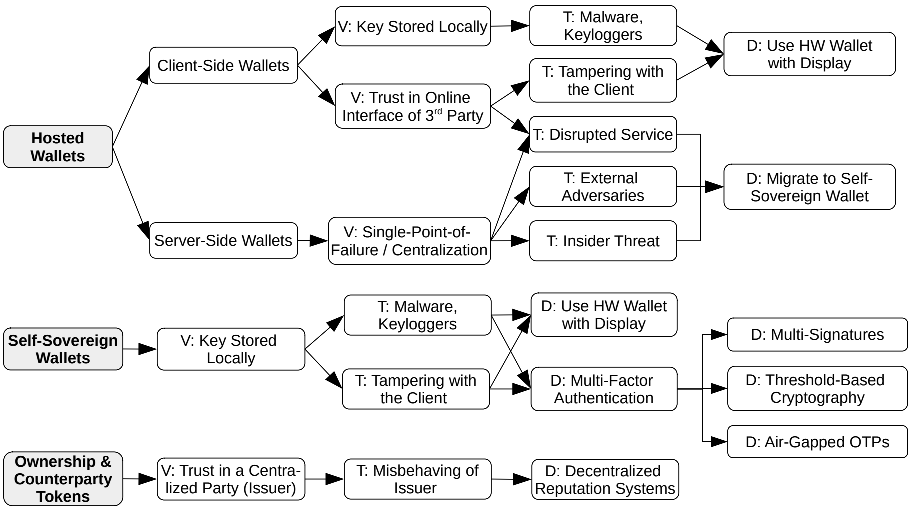

Hosted Wallets
Client-Side Wallets
Server-Side Wallets
V: Key Stored Locally
V: Trust in Online Interface of 3rd Party
V: Single-Point-of-Failure / Centralization
T: Malware, Keyloggers
T: Tampering with the Client
T: Disrupted Service
T: External Adversaries
T: Insider Threat
D: Use HW Wallet with Display
D: Migrate to Self-Sovereign Wallet
Self-Sovereign Wallets
V: Key Stored Locally
T: Malware, Keyloggers
T: Tampering with the Client
D: Use HW Wallet with Display
D: Multi-Factor Authentication
D: Multi-Signatures
D: Threshold-Based Cryptography
D: Air-Gapped OTPs
Ownership & Counterparty Tokens
V: Trust in a Centralized Party (Issuer)
T: Misbehaving of Issuer
D: Decentralized Reputation Systems

Figure 3.12: Vulnerabilities, threats, and defenses of the crypto-token & wallets category.

(1) **crypto-tokens and wallets** (see Figure 3.12), (2) **exchanges** (see Figure 3.13), (3) **oracles** (see Figure 3.14), (4) **filesystems** (see Figure 3.15), (5) **identity management** (see Figure 3.16), and (6) **secure-timestamping** (see Figure 3.17). For details of these categories of applications and their security threats and mitigation techniques, we refer the interested reader to our paper [HVR+20].

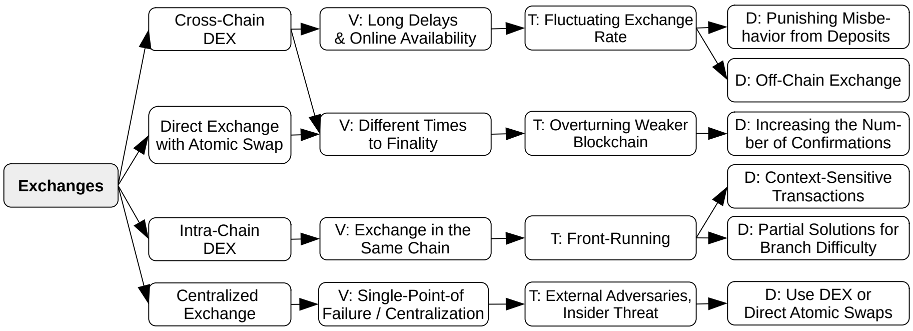

Exchanges
Cross-Chain DEX
Direct Exchange with Atomic Swap
Intra-Chain DEX
Centralized Exchange
V: Long Delays & Online Availability
V: Different Times to Finality
V: Exchange in the Same Chain
V: Single-Point-of Failure / Centralization
T: Fluctuating Exchange Rate
T: Overturning Weaker Blockchain
T: Front-Running
T: External Adversaries, Insider Threat
D: Punishing Misbehavior from Deposits
D: Off-Chain Exchange
D: Increasing the Number of Confirmations
D: Context-Sensitive Transactions
D: Partial Solutions for Branch Difficulty
D: Use DEX or Direct Atomic Swaps

Figure 3.13: Vulnerabilities, threats, and defenses of the exchanges category.

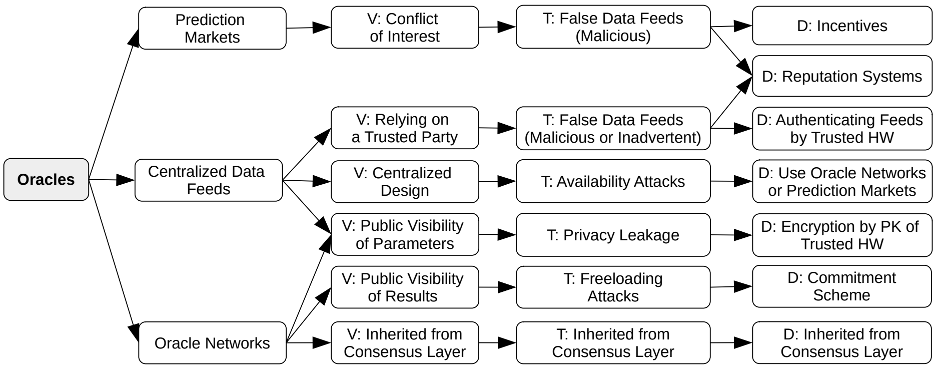

Oracles
Prediction Markets
Centralized Data Feeds
Oracle Networks
V: Conflict of Interest
V: Relying on a Trusted Party
V: Centralized Design
V: Public Visibility of Parameters
V: Public Visibility of Results
V: Inherited from Consensus Layer
T: False Data Feeds (Malicious)
T: False Data Feeds (Malicious or Inadvertent)
T: Availability Attacks
T: Privacy Leakage
T: Freeloading Attacks
T: Inherited from Consensus Layer
D: Incentives
D: Reputation Systems
D: Authenticating Feeds by Trusted HW
D: Use Oracle Networks or Prediction Markets
D: Encryption by PK of Trusted HW
D: Commitment Scheme
D: Inherited from Consensus Layer

Figure 3.14: Vulnerabilities, threats, and defenses of the oracles category.

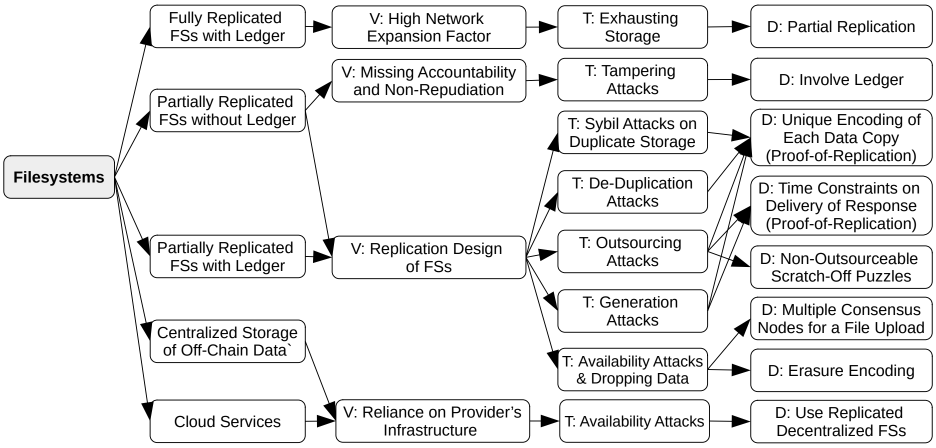

Filesystems
Fully Replicated FSs with Ledger
Partially Replicated FSs without Ledger
Partially Replicated FSs with Ledger
Centralized Storage of Off-Chain Data`
Cloud Services
V: High Network Expansion Factor
V: Missing Accountability and Non-Repudiation
V: Replication Design of FSs
V: Reliance on Provider's Infrastructure
T: Exhausting Storage
T: Tampering Attacks
T: Sybil Attacks on Duplicate Storage
T: De-Duplication Attacks
T: Outsourcing Attacks
T: Generation Attacks
T: Availability Attacks & Dropping Data
T: Availability Attacks
D: Partial Replication
D: Involve Ledger
D: Unique Encoding of Each Data Copy (Proof-of-Replication)
D: Time Constraints on Delivery of Response (Proof-of-Replication)
D: Non-Outsourceable Scratch-Off Puzzles
D: Multiple Consensus Nodes for a File Upload
D: Erasure Encoding
D: Use Replicated Decentralized FSs

Figure 3.15: Vulnerabilities, threats, and defenses of the filesystems category.

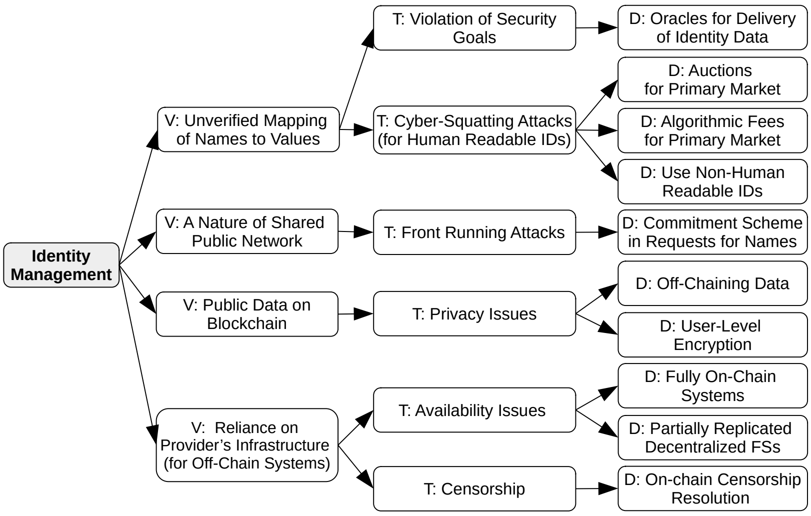

Identity Management
V: Unverified Mapping of Names to Values
V: A Nature of Shared Public Network
V: Public Data on Blockchain
V: Reliance on Provider's Infrastructure (for Off-Chain Systems)
T: Violation of Security Goals
T: Cyber-Squatting Attacks (for Human Readable IDs)
T: Front Running Attacks
T: Privacy Issues
T: Availability Issues
T: Censorship
D: Oracles for Delivery of Identity Data
D: Auctions for Primary Market
D: Algorithmic Fees for Primary Market
D: Use Non-Human Readable IDs
D: Commitment Scheme in Requests for Names
D: Off-Chaining Data
D: User-Level Encryption
D: Fully On-Chain Systems
D: Partially Replicated Decentralized FSs
D: On-chain Censorship Resolution

Figure 3.16: Vulnerabilities, threats, and defenses of the identity management category.

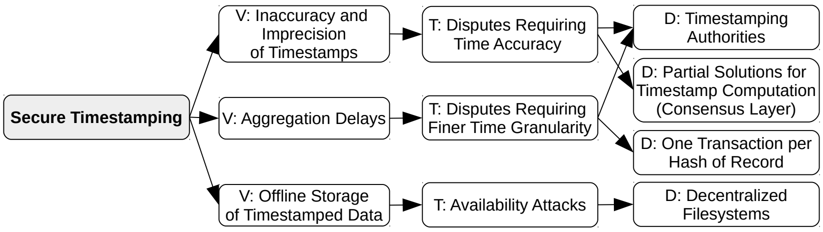

Secure Timestamping
V: Inaccuracy and Imprecision of Timestamps
V: Aggregation Delays
V: Offline Storage of Timestamped Data
T: Disputes Requiring Time Accuracy
T: Disputes Requiring Finer Time Granularity
T: Availability Attacks
D: Timestamping Authorities
D: Partial Solutions for Timestamp Computation (Consensus Layer)
D: One Transaction per Hash of Record
D: Decentralized Filesystems

Figure 3.17: Vulnerabilities, threats, and defenses of the secure timestamping category.

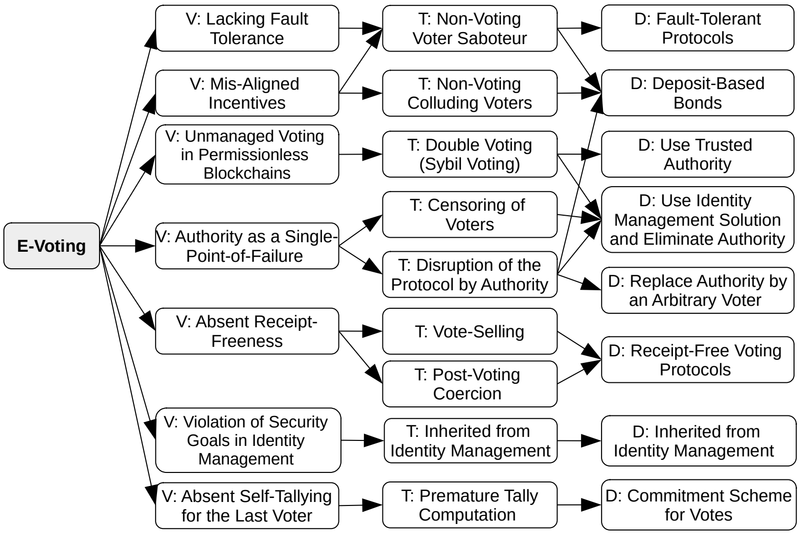

E-Voting
V: Lacking Fault Tolerance
V: Mis-Aligned Incentives
V: Unmanaged Voting in Permissionless Blockchains
V: Authority as a Single-Point-of-Failure
V: Absent Receipt-Freeness
V: Violation of Security Goals in Identity Management
V: Absent Self-Tallying for the Last Voter
T: Non-Voting Voter Saboteur
T: Non-Voting Colluding Voters
T: Double Voting (Sybil Voting)
T: Censoring of Voters
T: Disruption of the Protocol by Authority
T: Vote-Selling
T: Post-Voting Coercion
T: Inherited from Identity Management
T: Premature Tally Computation
D: Fault-Tolerant Protocols
D: Deposit-Based Bonds
D: Use Trusted Authority
D: Use Identity Management Solution and Eliminate Authority
D: Replace Authority by an Arbitrary Voter
D: Receipt-Free Voting Protocols
D: Inherited from Identity Management
D: Commitment Scheme for Votes

Figure 3.18: Vulnerabilities, threats, and defenses of the e-voting category.

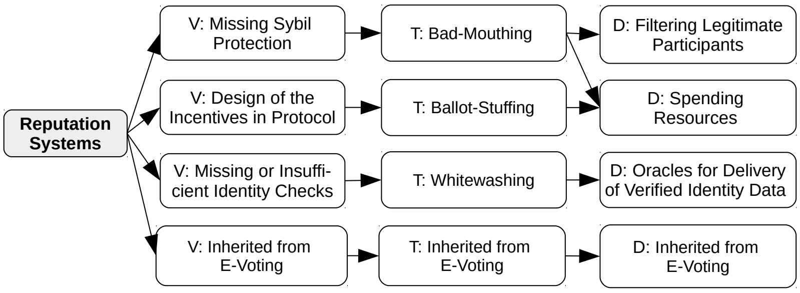

Reputation Systems
V: Missing Sybil Protection
V: Design of the Incentives in Protocol
V: Missing or Insufficient Identity Checks
V: Inherited from E-Voting
T: Bad-Mouthing
T: Ballot-Stuffing
T: Whitewashing
T: Inherited from E-Voting
D: Filtering Legitimate Participants
D: Spending Resources
D: Oracles for Delivery of Verified Identity Data
D: Inherited from E-Voting

Figure 3.19: Vulnerabilities, threats, and defenses of the reputation systems category.

3.7 Application Layer: Higher-Level Applications

In this section, we focus on more specific higher-level applications as opposed to ecosystem applications. In detail, we deal with eight categories of applications, and we outline their security threats and mitigation techniques in VTD graphs as follows: (1) **e-voting** (see Figure 3.18), (2) **reputation systems** (see Figure 3.19), (3) **data provenance** (see Figure 3.21), (4) **notaries** (see Figure 3.22), (5) **direct trading** (see Figure 3.20), (6) **escrows** (see Figure 3.23), (7) **auctions** (see Figure 3.24), and (8) **general application of blockchains**. For details of these categories of applications and their security threats and mitigation techniques, we refer the interested reader to our paper [HVR+20].

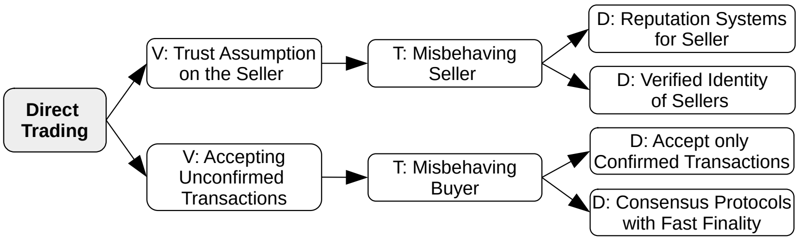

Direct Trading
V: Trust Assumption on the Seller
V: Accepting Unconfirmed Transactions
T: Misbehaving Seller
T: Misbehaving Buyer
D: Reputation Systems for Seller
D: Verified Identity of Sellers
D: Accept only Confirmed Transactions
D: Consensus Protocols with Fast Finality

Figure 3.20: Vulnerabilities, threats, and defenses of the direct trading category.

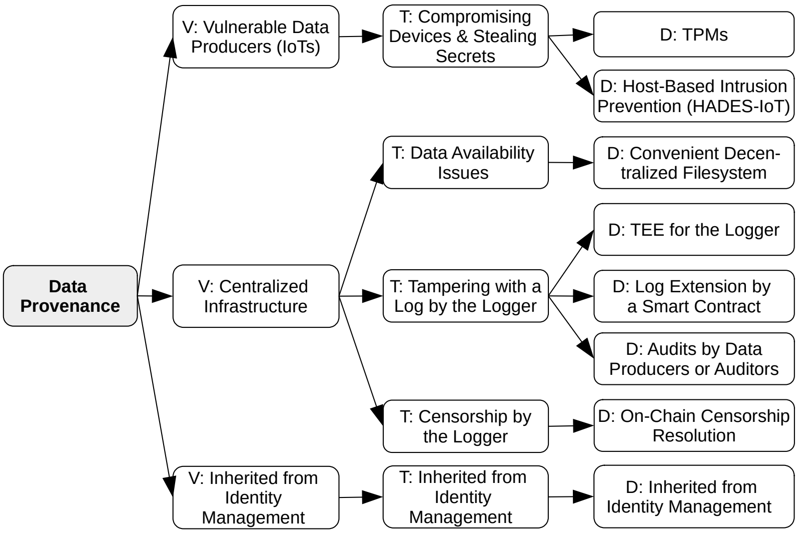

Data Provenance
V: Vulnerable Data Producers (IoTs)
T: Compromising Devices & Stealing Secrets
D: TPMs
D: Host-Based Intrusion Prevention (HADES-IoT)
T: Data Availability Issues
D: Convenient Decentralized Filesystem
D: TEE for the Logger
V: Centralized Infrastructure
T: Tampering with a Log by the Logger
D: Log Extension by a Smart Contract
D: Audits by Data Producers or Auditors
T: Censorship by the Logger
D: On-Chain Censorship Resolution
V: Inherited from Identity Management
T: Inherited from Identity Management
D: Inherited from Identity Management

Figure 3.21: Vulnerabilities, threats, and defenses of the data provenance category.

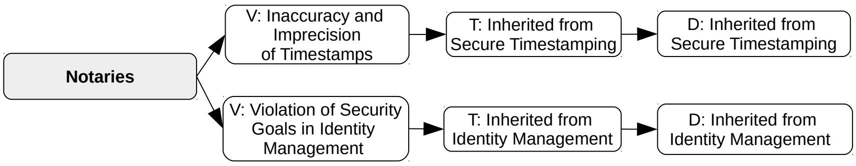

Notaries
V: Inaccuracy and Imprecision of Timestamps
T: Inherited from Secure Timestamping
D: Inherited from Secure Timestamping
V: Violation of Security Goals in Identity Management
T: Inherited from Identity Management
D: Inherited from Identity Management

Figure 3.22: Vulnerabilities, threats, and defenses of the notaries category.

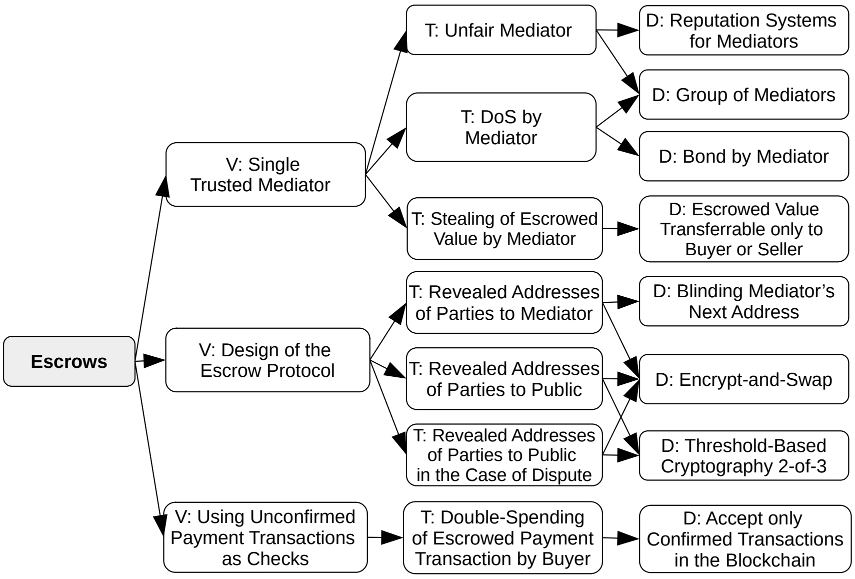

T: Unfair Mediator
D: Reputation Systems for Mediators
D: Group of Mediators
T: DoS by Mediator
D: Bond by Mediator
V: Single Trusted Mediator
T: Stealing of Escrowed Value by Mediator
D: Escrowed Value Transferrable only to Buyer or Seller
T: Revealed Addresses of Parties to Mediator
D: Blinding Mediator's Next Address
Escrows
V: Design of the Escrow Protocol
T: Revealed Addresses of Parties to Public
D: Encrypt-and-Swap
T: Revealed Addresses of Parties to Public in the Case of Dispute
D: Threshold-Based Cryptography 2-of-3
V: Using Unconfirmed Payment Transactions as Checks
T: Double-Spending of Escrowed Payment Transaction by Buyer
D: Accept only Confirmed Transactions in the Blockchain

Figure 3.23: Vulnerabilities, threats, and defenses of the escrows category.

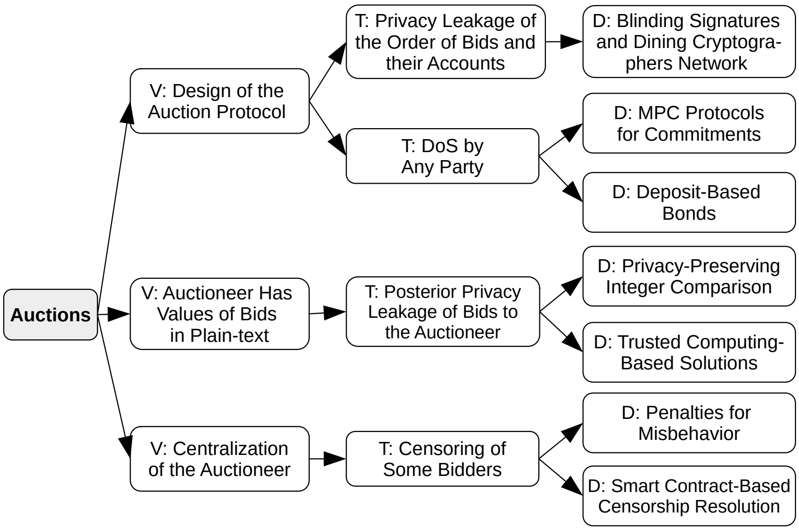

Auctions
V: Design of the Auction Protocol
T: Privacy Leakage of the Order of Bids and their Accounts
D: Blinding Signatures and Dining Cryptographers Network
T: DoS by Any Party
D: MPC Protocols for Commitments
D: Deposit-Based Bonds
V: Auctioneer Has Values of Bids in Plain-text
T: Posterior Privacy Leakage of Bids to the Auctioneer
D: Privacy-Preserving Integer Comparison
D: Trusted Computing-Based Solutions
V: Centralization of the Auctioneer
T: Censoring of Some Bidders
D: Penalties for Misbehavior
D: Smart Contract-Based Censorship Resolution

Figure 3.24: Vulnerabilities, threats, and defenses of the auctions category.

3.8 Lessons Learned

In this section, we summarize lessons learned concerning the security reference architecture (SRA) and its practical utilization. First, we describe the hierarchy of security dependencies among particular layers of the SRA. Second, assuming such a hierarchy, we describe a security-oriented methodology for designers of blockchain platforms and applications. Finally, we summarize the design goals of particular blockchain types and discuss the security-specific features of the blockchains.

3.8.1 Hierarchy of Dependencies in the SRA

In the proposed model of the SRA, we observe that consequences of vulnerabilities presented at lower layers of the SRA are manifested in the same layers and/or at higher layers, especially at the application layer. Therefore, we refer to *security dependencies* of these layers on lower layers or the same layers, i.e., *reflexive* and *bottom-up* dependencies. We describe these two types of dependencies in the following.

Reflexive Dependencies. If a layer of the SRA contains some assets, it also contains a reflexive security dependency on the countermeasures presented in the same layer. It means that a countermeasure at a particular layer protects the assets presented in the same layer. For example, in the case of the consensus layer whose protocols reward consensus nodes for participation, the countermeasures against selfish mining attacks protect rewards (i.e., crypto-tokens) of consensus nodes. In the case of the RSM layer, the privacy of user identities and data is protected by various countermeasures of this layer (e.g., blinding signatures, secure multiparty computations). Another group of reflexive security dependencies is presented at the application layer. Although the application layer contains some bottom-up security dependencies (see Figure 3.11), we argue that with regard to the overall stacked model of the SRA they can be viewed as reflexive security dependencies of the application layer.

Bottom-Up Dependencies. If a layer of the SRA contains some assets, besides reflexive security dependencies, it also contains bottom-up security dependencies on the countermeasures of all lower layers. Hence, the consequences of vulnerabilities presented at lower layers of SRA might be manifested at the same layers (i.e., reflexive dependencies) but more importantly, they are manifested at higher layers, especially at the application layer. For example, context-sensitive transactions and partial solutions as countermeasures of the consensus layer can protect against front-running attacks of intra-chain DEXes, which occur at the application layer. Another example represents programming bugs in the RSM layer, which influence the correct functionality at the application layer. The eclipse attack is an example that impacts the consensus layer from the network layer – a victim consensus node operates over the attacker-controlled chain, and thus causes a loss of crypto-tokens by a consensus node and at the same time it decreases honest consensus power of the network. In turn, this might simplify selfish mining attacks at the consensus layer, which in turn might impact the correct functionality of a blockchain-based application at the application layer. Bottom-up security dependencies are also presented in the context of the application layer, as we have already mentioned in Section 3.6.

3.8.2 Methodology for Designers

A hierarchy of security dependencies in the SRA can be utilized during the design of new blockchain-based solutions. When designing a new **blockchain platform** or a new **blockchain application**, we recommend designers to specify requirements on the blockchain features (see Section 2.1) and afterward analyze design options and their attack surfaces at the first three layers of the stacked model of SRA. We briefly summarize the pros and cons of particular categories within the first three layers of SRA in Table 3.1, while security threats and mitigations are covered in Section 3.3, Section 3.4, and Section 3.5.

On top of that, we recommend the designers of a new **blockchain application** to analyze particular options and their security implications at the application layer of SRA. We list the pros and cons of a few categories from the application layer in Table 3.2,[1] while security threats and mitigation techniques of this layer are elaborated in Section 3.6 and Section 3.7. During this process, we recommend the designers to follow security dependencies of the target category on other underlying categories (see Figure 3.11) if their decentralized variants are used (which is a preferable option from the security point-of-view). For example, if one intends to design a decentralized reputation system, she is advised to study the security threats from the reputation system category and its recursive dependencies on e-voting, identity management, crypto-tokens & wallets, and (optionally) filesystems.

[1] Note that the table contains only categories with specified sub-categorizations that represent the subject to a comparison.

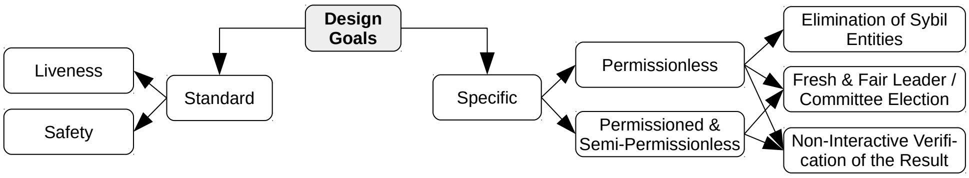

Design Goals
Standard
Specific
Liveness
Safety
Permissionless
Permissioned & Semi-Permissionless
Elimination of Sybil Entities
Fresh & Fair Leader / Committee Election
Non-Interactive Verification of the Result

Figure 3.25: Standard and specific design goals of consensus protocols.

Divide and Conquer. If a designer of the blockchain application is also designing a blockchain platform, we recommend her to split the functionality of the solution with the divide-and-conquer approach respecting particular layers of our stacked model. In detail, if some functionality is specific to the application layer, then it should be implemented at that layer. Such an approach minimizes the attack surface of a solution and enables isolating the threats to specific layers, where they are easier to protect from and reviewed by the community. A contra-example is to incorporate a part of application layer functionality/validation into the consensus layer. The consensus layer should deal only with the ordering of transactions, and it should be agnostic to the application.

Nevertheless, it is worth noting that the divide-and-conquer approach might not be suitable for some very specific cases. For example, some decentralized filesystems might combine data storage as an application-layer service with the proof-of-storage consensus algorithm, presented at the consensus layer. Therefore, the consensus layer also embeds a part of functionality from the application layer. However, when filesystems are in security dependencies of the target application other than filesystems, one should realize that they are usually running on a different blockchain or infrastructure than the target application, and this exception is not a concern.

3.8.3 Blockchain Types & Design Goals

We learned that the type of a blockchain (see Section 2.3) implies the specific design goals of its consensus protocol (see Figure 3.25), which must be considered on top of the standard design goals (i.e., liveness and safety) and the inherent features (see Section 2.1.1) during the design of a particular blockchain platform and its consensus protocol. In the following, we elaborate on such specific design goals.

Permissionless Type. The first design goal is to *eliminate Sybil entities* – such elimination can be done by requiring that some amount of scarce resources is spent for extension of the blockchain, and hence no Sybil entity can participate. This implies that no pure PoS protocol can be permissionless since it never spends resources on running a consensus protocol. The next design goal is *a fresh and fair leader/committee election*, which ensures that each consensus node influences the result of a consensus commensurately to the number of scarce resources spent. Moreover, freshness avoids the prediction of the selected nodes, and therefore elected nodes cannot become the subject of targeted DoS attacks. The last design goal is the *non-interactive verification* of the consensus result by any node – i.e., any node can verify the result of the consensus based on the data presented in the blockchain.

Permissioned and Semi-Permissionless Types. These types of blockchains require *fresh and fair leader/committee election* as well as *non-interactive verification* of the result of the consensus. However, in contrast to the permissionless blockchains, they do not require a means for the elimination of Sybil entities, as permission to enter the system is given by a centralized entity (i.e., permissioned type) or any existing consensus node (i.e., semi-permissionless type).

Blockchain Types and Incentives. We observed that no application running on a *public* (permissioned) blockchain has been able to work without introducing crypto-tokens (i.e., an incentive scheme), even if the use case is not financial in nature, e.g., e-voting, notaries, secure timestamping, or reputation systems. In these blockchains, incentive schemes serve as a means for the elimination of Sybil entities, besides other purposes. The situation is different in the context of *private* (permissioned) blockchains, which are usually provisioned by a single organization or a consortium and do not necessarily need crypto-tokens to operate. Misaligned incentives can cause consensus-level vulnerabilities, e.g., when it becomes profitable to drop blocks of other nodes to earn higher mining rewards [ES18] or transaction fees [CKWN16b]. The design of incentive mechanisms is a research field by itself and we refer the reader to the work of Leonardos et al. [LRP20].

3.8.4 Security-Specific Features of Blockchains

We realized that consensus protocols are the target of most financially-oriented attacks on the decentralized infrastructure of blockchains, even if such attacks might originate from the network layer (e.g., routing and eclipse attacks). The goal of these attacks is to overturn and re-order already ordered blocks while doing double-spending. Hence, the finality is the most security-critical feature of the consensus layer. The finality differs per various categories of the consensus layer. The best finality is achieved in the pure BFT protocols, and the worse finality is achieved in the single-leader-based PoR and PoS protocols. On the other hand, combinations of the BFT with PoS protocols (i.e., introducing committees) slightly deteriorate the finality of BFT in a probabilistic ratio that is commensurate to the committee size. In the case of PoR protocols with partial solutions, finality is improved as opposed to pure PoR protocols; however, it is also probabilistic, depending on the number of partial solutions.

3.8.5 Limitations in the Literature and Practice

Applications of Blockchains. Although the literature contains surveys and overviews [CDP18, ZXD$^+$18, WG18] of blockchain-based applications, these works introduce only domain-oriented categorizations (i.e., categories such financial, governance, security, education, supply chain, etc.) and they do not investigate the security aspects and functionalities that these applications leverage on and whether some of the applications do not belong to the same category from the security and functionality point-of-view. To address this limitation, we provide a security-driven functionality-oriented categorization of blockchain-based applications (see Section 3.6), which is agnostic to an application domain and thus can generalize different application scenarios. Furthermore, our proposed categorization enables us to model security and functionality-based dependencies among particular categories, which is not possible with state-of-the-art categorizations.

Centralization. Even though blockchains are meant to be fully decentralized, we have seen that this does not hold at some layers of the SRA – the network and application layers, in particular. In the network layer, some attacks are possible due to centralized DNS bootstrapping, while in the application layer a few categories utilize centralized components to ensure some functionality that cannot run on-chain or its provisioning would be too expensive and slow, which, however, forms the trade-off with the security. Some applications might depend on components from other application categories (e.g., identity management) but implementing these components in a centralized fashion, even though there exist some decentralized variants that are gaining popularity (e.g., DIDs [W3C19a] for identity management).

3.9 Contributing Papers

The papers that contributed to this research direction are enumerated in the following, while highlighted papers are attached to this thesis in their original form.

[HVHS19] Ivan Homoliak, Sarad Venugopalan, Qingze Hum, and Pawel Szalachowski. A security reference architecture for blockchains. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 390–397. IEEE, 2019.

[HVR+20] **Ivan Homoliak, Sarad Venugopalan, Daniël Reijsbergen, Qingze Hum, Richard Schumi, and Pawel Szalachowski. The security reference architecture for blockchains: Toward a standardized model for studying vulnerabilities, threats, and defenses. *IEEE Communications Surveys & Tutorials*, 23(1):341–390, 2020.**

Layer			Category in a Layer	Pros	Cons
Network Layer			Private Networks	● low latency, high throughput ● centralized administration, ● ease of access control ● the privacy of data and identities ● meeting regulatory obligations ● resilience to external attacks	● VPN is required for geographically spread participants ● suitable only for permissioned blockchains ● insider threat and external attacks at nodes with administrative privileges
			Public Networks	● high decentralization ● high availability ● openness & low entry barrier	● high and non-uniform latency ● single-point-of-failure (DNS, IP, and ASes are managed by centralized parties) ● external adversaries (botnets, compromised BGP/DNS servers) ● stolen identities
Consensus Layer			PoR	● high cost of overriding the history of blockchain ● high scalability	● high operational costs ● low throughput ● low finality
			BFT	● high throughput (with a small number of nodes) ● fast finality	● low scalability ● high communication complexity ● limited number of nodes (efficient use only in permissioned blockchains)
			PoS	● energy efficiency	● PoS specific attacks and issues ● supports only semi-permissionless setting ● slow finality
			PoS+BFT	● energy efficiency ● high scalability ● probabilistic security guarantees ● lower communication overheads than BFT	● some PoS specific attacks ● supports only semi-permissionless setting
			PoR+BFT	● high scalability ● fast finality	● spending some scarce resources
			PoR+PoS (i.e., PoA)	● high scalability	● spending some scarce resources ● some PoS specific attacks ● slow finality
Replicated State Machine Layer	Transactions	Protecting Identities	Standard Approach	● fast processing ● ease of verification	● identities are only pseoudonymous and can be traced to IPs ● all data of transactions are publicly visible
			Standard Approach + Mixers	● privacy identity protection of users in a group ● ease of verification	● additional complexity, in some cases unlinkability by the mixer or involved parties in a group ● all data of transactions are publicly visible
			NIZKs and Ring-Signatures	● identities are anonymized to the extend of the group	● additional computation overheads for running the schemes
			MPC Blinding Signatures, Layered-Encryption	● unlinkability for all involved parties	● additional computation overheads for running the schemes
		Protecting Data	NIZKs, Blinding Signatures, Homomorphic Encryption	● privacy of data in cryptocurrency platforms	● additional computation overheads for running the schemes
			Trusted Transaction Managers, Trusted Hardware, MPC	● privacy of data in transactions of smart contract platforms	● additional computation overheads for running the schemes
	Smart Contracts		Turing-Complete Languages	● smart contracts may contain an arbitrary programming logic	● wide surface for making the programming bugs that often results in vulnerabilities
			Turing-Incomplete Languages	● small attack surface and emphasis on safety	● the programming logic serves only for limited purposes

Table 3.1: Pros and cons of various categories within the first three layers of the stacked model.

Application Category	Subcategory	Pros	Cons
Wallets	**Server-Side Hosted Wallets**	● simplicity of control for end-users ● no storage requirements for end-users	● keys stored at the server, susceptibility to the theft of keys by external or internal attacks ● single-point-of-failure, availability attacks
	Client-Side Hosted Wallets	● simplicity of control for end-users ● no storage requirements for end-users ● keys stored locally	● single-point-of-failure, availability attacks ● possibility of key theft by malware ● possibility of tampering attacks
	Self-Sovereign Wallets	● keys stored locally or in a dedicated hardware device	● moderate storage requirements for end-users ● more difficult control for end-users ● extra device to carry in the case of hardware wallet
Exchanges	**Centralized Exchange**	● a high throughput and speed of operations ● the simplicity of control for end-users ● low costs for exchange transactions ● trading of obscure crypto-tokens	● risk of insider threat due to centralization ● external threats to exchange infrastructure ● overheads for secure storage of secrets ● a fee specified by the operator
	Direct Cross-Chain Exchange	● fairness of the exchange ● no fee to any operator	● costs for 4 transactions of the atomic swap ● user has to find the counter-order on her own ● counter-orders might not exist ● a lower throughput than in a centralized exchange ● a higher complexity for end-users
	Cross-Chain DEX	● fairness of the exchange ● order matching made by DEX ● trading of obscure crypto-tokens	● costs for 4 or 6 transactions of the atomic swap ● a lower throughput than in a centralized exchange ● a fee specified by the operator
	Intra-Chain DEX	● fairness of the exchange ● uniform finality for every pair ● a high speed of operations	● a limited number of pairs specific to the target platform ● a fee specified by the operator ● costs for smart contract execution
Oracles	**Prediction Markets**	● early (close to accurate) estimation of the future event's result ● decentralization	● possible conflict of interest ● a limited set of data specific to a few events ● a long time to obtain a final result, especially in the case of disputes
	Centralized Data Feeds	● wide range of data ● fast provisioning time ● handling of private parameters of requests ● censorship evidence	● centralization (accidentally or intentionally wrong data) ● availability issues
	Oracle Networks	● decentralization ● wide range of data ● fast provisioning time	● unsupported private parameters of requests ● publicly visible data and requests
Filesystems	**Fully Replicated FSs with Ledger**	● a high availability ● accountability and auditability	● a high storage overheads and operational costs ● a high price
	Partially Replicated FSs with Ledger	● reasonably high availability ● accountability and auditability ● a lower price than in a full replication	● attack vectors specific to partial replication
	Partially Replicated FSs without Ledger	● reasonably high availability ● a lower price than in a full replication	● a lack of native accountability and auditability ● low durability due to a lack of incentives for storage
	Centralized Storage of Off-Chain Data	● a low price ● accountability and auditability	● a low availability

Table 3.2: Pros and cons of some categories from the application layer.

Chapter 4

Consensus Protocols

In this chapter, we present our contributions to the area of consensus protocols in blockchains and their security, which belong to the consensus layer of our security reference architecture (see Chapter 3). In particular, this chapter is focused on Proof-of-Work (PoW) consensus protocols and is based on the papers [SRHS19, PBH+, PHB$^+$23, PHMH24, BHS23] (see also Section 4.4).

First, we address the selfish mining attack by revising the Nakamoto Consensus protocol used in Bitcoin into a new design called StrongChain [SRHS19], and we show that StrongChain mitigates this attack in our simulation experiments. Next, we demonstrate the existence of incentive attacks [PBH+, PHMH24, PBMH21] on several DAG-based PoW consensus protocols with random transaction selection by game theoretical analysis and simulation, while we also elaborate on a few mitigation techniques.[1] Finally, we address the problems of undercutting attacks and the mining gap in a sole transaction-fee-based regime of PoW blockchains by proposing fee-redistribution smart contracts [BHS23] as a modification of the Nakamoto Consensus. In the following, we briefly introduce these research directions and summarize our contributions, while in later sections we go into more detail.

Selfish Mining Attack & StrongChain

Selfish mining [ES14] is a strategy where the attacker holding a fraction α of total mining power can earn more than α of total rewards by occupying more than α of the total mined blocks. Selfish mining became a profitable strategy after reaching a certain mining power threshold by the attacker – e.g., 33% in Bitcoin.[2] The key principle of selfish mining is that the attacker keeps building her secret chain that poses a fork w.r.t., the honest chain and releases it when the honest chain starts to "catch up" with the attacker's chain, overriding the honest chain since miners accept the longer (attacker's) chain as the canonical one and continue to mine on it. This causes honest miners to waste their work/resources on the chain that is abandoned.

[1] Note that we first analyzed this attack on PHANTOM and its optimization GHOSTDAG [PBMH21], and later we generalized this attack and applied it to more concerned protocols [PBH+].

[2] Nevertheless, it can be arbitrarily low in the case sole transaction-fee-based regime.

Nakamoto consensus [Nak08] is PoW consensus protocol that stands behind the Bitcoin – the most successful cryptocurrency so far. However, despite its unprecedented success, Bitcoin suffers from many inefficiencies. In particular, Bitcoin's consensus mechanism has been proven to be incentive-incompatible (e.g., mostly by selfish mining), its high reward variance causes centralization (by creating mining pools), and its hardcoded deflation raises questions about its long-term sustainability with regard to the mining gap and sole transaction-fee-based regime.

Therefore, we revise the Bitcoin consensus mechanism by proposing StrongChain, a scheme that introduces transparency and incentivizes participants to collaborate rather than to compete. The core design of StrongChain is to utilize the computing power aggregated on the blockchain which is invisible and "wasted" in Bitcoin by default. Introducing relatively easy, although important changes to Bitcoin's design enable us to improve many crucial aspects of Bitcoin-like cryptocurrencies making them more secure, efficient, and profitable for participants. Most importantly, StrongChain improves the threshold where the selfish mining strategy starts to be profitable by 10% in contrast to Bitcoin. See further details on StrongChain in Section 4.1.

Incentive Attacks on DAG-based Blockchains

Blockchains inherently suffer from the processing throughput bottleneck, as consensus must be reached for each block within the chain. One approach to solve this problem is to increase the block creation rate. However, such an approach has drawbacks. If blocks are not propagated through the network before a new block is created, a *soft fork* might occur, in which two concurrent blocks reference the same parent block. A soft fork is resolved in a short time by a fork-choice rule, and thus only one block is eventually accepted. All transactions in an *orphaned* (a.k.a., stale) block are discarded. As a result, consensus nodes that created orphaned blocks wasted their resources and did not get rewarded.

As a response to the above issue, several proposals (e.g., Inclusive [LSZ15], PHANTOM [SWZ21], GHOSTDAG [SWZ21], SPECTRE [SLZ16]) have substituted a single chaining data structure for unstructured direct acyclic graph, while another proposal in this direction employed structured DAG (i.e., Prism [BKT$^+$19]). Such a structure can maintain multiple interconnected chains and thus theoretically increase processing throughput. The assumption of concerned dag-oriented solutions is to abandon transaction selection purely based on the highest fees since this approach intuitively increases the probability that the same transaction is included in more than one block (hereafter *transaction collision*). Instead, these approaches use the random transaction selection (i.e., RTS) [3] strategy to avoid transaction collisions. Although the consequences of deviating from such a strategy might seem intuitive, no one has yet thoroughly analyzed the performance and robustness of concerned dag-based approaches under incentive attacks aimed at transaction selection.

[3] Note that RTS involves certain randomness in transaction selection but does not necessarily equals to uniform random transaction selection (to be in line with the works utilizing Inclusive [LSZ15], such as PHANTOM, GHOSTDAG [SWZ21], SPECTRE [SLZ16], as well as the implementation of GHOSTDAG called Kaspa [Som22]).

Therefore, we focus on the impact of incentive attacks caused by **greedy**[4] actors in above-mentioned dag-oriented designs of consensus protocols. In particular, we study the situation where a greedy attacker deviates from the protocol by not following the RTS strategy that is assumed by the mentioned DAG-based approaches. We make a hypothesis stating that the attacker deviating from RTS strategy might earn greater rewards as compared to honest participants, and such an attacker harms transaction throughput since transaction collision is increased. We verify and prove our hypothesis in a game theoretical analysis and show that RTS does not constitute Nash equilibrium. Next, we substantiate conclusions from game theoretical analysis by a few simulation experiments on the abstracted DAG-PROTOCOL, which confirm that a greedy actor who selects transactions based on the highest fee profits significantly more than honest miners following the RTS. In another experiment, we demonstrate that multiple greedy actors can significantly reduce the effective transaction throughput by increasing the transaction collision rate across parallel chains of DAGs. Finally, we show that greedy actors have an incentive to form a mining pool to increase their relative profits, which degrades the decentralization of the concerned DAG-oriented designs. See further details on these incentive attacks together with potential mitigation techniques in Section 4.2.

Undercutting Attacks & Transaction-Fee-based Regime

In Bitcoin and its numerous clones, the block reward is divided by two approx. every four years (i.e., after every 210k blocks), which will eventually result in a zero block reward around the year 2140 and thus *a pure transaction-fee-based regime*. There was only very little research made to investigate the properties of transaction-fee-based regimes, which motivated our work.

Before 2016, there was a belief that the dominant source of the miners' income does not impact the security of the blockchain. However, Carlsten et al. [CKWN16a] pointed out the effects of the high variance of the miners' revenue per block caused by exponentially distributed block arrival time in the transaction-fee-based protocols. The authors showed that *undercutting* (i.e., forking) a wealthy block is a profitable strategy for a malicious miner. Nevertheless, literature [DGK$^+$20, MKV23] showed that this attack is viable even in blockchains containing traditional block rewards due to front-running competition of arbitrage bots who are willing to extremely increase transaction fees to earn Maximum Extractable Value profits.

Therefore, we focus on mitigation of the undercutting attack in the transaction-fee-based regime of the single chain PoW blockchains – i.e., blockchains that prefer availability over consistency within the CAP theorem and thus are designed to resolve forks often. We also discuss related problems present (not only) in a transaction-fee-based regime. In particular, we focus on minimizing the mining gap [CKWN16a, TE18a], (i.e., the situation, where the immediate reward from transaction fees does not cover miners' expenditures) as well as balancing significant fluctuations in miners' revenue. To mitigate these issues, we propose a solution that splits transaction fees from a mined block into two parts – (1) an instant reward for the miner and (2) a deposit sent into one or more

[4] Greedy actors deviate from the protocol to increase their profits.

fee-redistribution smart contracts ($\mathcal{FRSC}$s). At the same time, these $\mathcal{FRSC}$s reward the miner of a block with a certain fraction of the accumulated funds over a fixed number of blocks, thereby emulating the moving average on a portion of the transaction fees. We evaluate our approach using various fractions of the transaction fees (split across the miner and $\mathcal{FRSC}$) and experiment with the various numbers and lengths of $\mathcal{FRSC}$s – we demonstrate that usage of multiple $\mathcal{FRSC}$s of various lengths has the best advantages mitigating the problems we are addressing. Finally, we perform a simulation demonstrating that the threshold of DEFAULT-COMPLIANT miners who strictly do not execute undercutting attack is lowered from 66% (as reported in [CKWN16a]) to 30% with our approach. See further details on undercutting attacks and our solution in Section 4.3.

In the following section, we detail the individual directions of our research.

4.1 StrongChain

4.1.1 Bitcoin's Drawbacks

There are multiple drawbacks of Bitcoin that undermine its security promises and raise questions about its future. Bitcoin has been proved to be incentive-incompatible [Eya15, SSZ16, ES14, TE18b]. Namely, in some circumstances, the miners' best strategy is to not announce their found solutions immediately, but instead withhold them for some time period (e.g., selfish mining [ES18]). Another issue is that the increasing popularity of the system tends towards its centralization. Strong competition between miners resulted in a high reward variance, thus to stabilize their revenue miners started grouping their computing power by forming *mining pools*. Over time, mining pools have come to dominate the computing power of the system, and although they are beneficial for miners, large mining pools are risky for the system as they have multiple ways of abusing the protocol [KAC12, Eya15, ES14, SSZ16]. Also, a few researchers rigorously analyzed one of the impacts of Bitcoin's deflation [MB15, CKWN16a, TE18b]. Their results indicate that Bitcoin may be unsustainable in the long term, mainly due to decreasing miners' rewards that will eventually stop at all. Besides that, unusually for a transaction system, Bitcoin is designed to favor availability over consistency. This choice was motivated by its open and permissionless spirit, but in the case of inconsistencies (i.e., *forks* in the blockchain) the system can be slow to converge.

4.1.2 Overview of Proposed Approach

Motivated by these drawbacks, we propose StrongChain, a simple yet powerful revision of the Bitcoin consensus mechanism. Our main intuition is to design a system such that the mining process is more transparent and collaborative, i.e., miners get better knowledge about the mining power of the system and they are incentivized to solve puzzles together rather than compete. In order to achieve it, in the heart of the StrongChain's design we employ *weak solutions*, i.e., puzzle solutions with a PoW that is significant yet insufficient for a standard solution. We design our system, such that a) weak solutions are part of the consensus protocol, b) their finders are rewarded independently, and c) miners have incentives to announce own solutions and append solutions of others immediately. We show that with these changes, the mining process is becoming more transparent, collaborative, secure, efficient, and decentralized. Surprisingly, we also show how our approach can improve the freshness properties offered by Bitcoin.

4.1.3 Details

In our scheme, miners solve a puzzle as today but in addition to publishing solutions, they exchange weak solutions too (i.e., almost-solved puzzles). The lucky miner publishes her solution that embeds gathered weak solutions (pointing to the same previous block) of other miners. Such a published block better reflects the aggregated PoW of a block, which in the case of a fork can indicate that more mining power is focused on a given branch (i.e., actually it proves that more computing power "believes" that the given branch is correct). Another crucial change is to redesign the Bitcoin reward system, such that the finders of weak solutions are also rewarded. Following lessons learned from mining pool attacks, instead of sharing rewards among miners, our scheme rewards weak solutions proportionally to their PoW contributed to a given block and all rewards are independent of other solutions of the block.[5]

There are a few intuitions behind these design choices. First, a selfish miner finding a new block takes a high risk by keeping this block secret. This is because blocks have a better granularity due to honest miners exchanging partial solutions and strengthening their prospective block, which in the case of a fork would be stronger than the older block kept secret (i.e., the block of the selfish miner). Secondly, miners are actually incentivized to collaborate by a) exchanging their weak solutions, and b) by appending weak solutions submitted by other miners. For the former case, miners are rewarded whenever their solutions are appended, hence keeping them secret can be unprofitable for them. For the latter case, a miner appending weak solutions of others only increases the strength of her potential block, and moreover, appending these solutions does not negatively influence the miner's potential reward. Finally, our approach comes with another benefit. Proportional rewarding of weak solutions decreases the reward variance, thus miners do not have to join large mining pools in order to stabilize their revenue. This could lead to a higher decentralization of mining power on the network.

Mining

As in Bitcoin, in StrongChain miners authenticate transactions by collecting them into blocks whose headers are protected by a certain amount of PoW. A simplified description of a block mining procedure in StrongChain is presented as the *mineBlock()* function in Algorithm 1. Namely, every miner tries to solve a PoW puzzle by computing the hash function over a newly created header. The header is constantly being changed by modifying its nonce field,[6] until a valid hash value is found. Whenever a miner finds a

[5] Note, that this change requires a Bitcoin *hard fork*.

[6] In fact, other fields can be modified too if needed.

Algorithm 1: Pseudocode of StrongChain.

```
function mineBlock()
    $weakHdrsTmp \leftarrow \emptyset$;
    for $nonce \in \{0, 1, 2, ...\}$ do
        $hdr \leftarrow createHeader(nonce)$;
        /* check if the header meets the strong target */
        $h_{tmp} \leftarrow H(hdr)$;
        if $h_{tmp} < T_s$ then
            $B \leftarrow createBlock(hdr, weakHdrsTmp, Txs)$;
            $broadcast(B)$;
            return; /* signal to mine with the new block */
        /* check if the header meets the weak target */
        if $h_{tmp} < T_w$ then
            $weakHdrsTmp.add(hdr)$;
            $broadcast(hdr)$;
function onRecvWeakHdr(hdr)
    $h_w \leftarrow H(hdr)$;
    assert($T_s \leq h_w < T_w$ and validHeader(hdr));
    assert($hdr.PrevHash == H(lastBlock.hdr)$) ;
    $weakHdrsTmp.add(hdr)$;
function rewardBlock(B)
    /* reward block finder with $R$ */
    reward($B.hdr.Coinbase, R + B.TxFees$);
    $w \leftarrow \gamma * T_s/T_w$; /* reward weak headers proportionally */
    for $hdr \in B.weakHdrSet$ do
        reward($hdr.Coinbase, w * c * R$);
function validateBlock(B)
    assert($H(B.hdr) < T_s$ and validHeader(B.hdr));
    assert($B.hdr.PrevHash == H(lastBlock.hdr)$) ;
    assert($validTransactions(B)$);
    for $hdr \in B.weakHdrSet$ do
        assert($T_s \leq H(hdr) < T_w$ and validHeader(hdr));
        assert($hdr.PrevHash == H(lastBlock.hdr)$);
function chainPoW(chain)
    $sum \leftarrow 0$;
    for $B \in chain$ do
        /* for each block compute its aggregated PoW */
        $T_s \leftarrow B.hdr.Target$;
        $sum \leftarrow sum + T_{max}/T_s$;
        for $hdr \in B.weakHdrSet$ do
            $sum \leftarrow sum + T_{max}/T_w$;
    return $sum$;
function getTimestamp(B)
    $sumT \leftarrow B.hdr.Timestamp$;
    $sumW \leftarrow 1.0$;
    /* average timestamp by the aggregated PoW */
    $w \leftarrow T_s/T_w$;
    for $hdr \in B.weakHdrSet$ do
        $sumT \leftarrow sumT + w * hdr.Timestamp$;
        $sumW \leftarrow sumW + w$;
    return $sumT/sumW$;
```

header hdr whose hash value $h = H(hdr)$ is smaller than the *strong target* T_s, i.e., a h that satisfies the following:

$$h < T_s,$$

then the corresponding block is announced to the network and becomes, with all its transactions and metadata, part of the blockchain. We refer to headers of included blocks as *strong headers*.

One of the main differences with Bitcoin is that our mining protocol handles also headers whose hash values do not meet the strong target T_s, but still are low enough to prove a significant PoW. We call such a header a *weak header* and its hash value h has to satisfy the following:

$$T_s \leq h < T_w, \tag{4.1}$$

where $T_w > T_s$ and T_w is called the *weak target*.

Whenever a miner finds such a block header, she adds it to her local list of weak headers (i.e., *weakHdrsTmp*) and she propagates the header among all miners. Then every miner that receives this information first validates it (see *onRecvWeakHdr()*) by checking whether

- the header points to the last strong header,
- its other fields are correct (see Section 50),
- and Equation 4.1 is satisfied.

Afterward, miners append the header to their lists of weak headers. We do not limit the number of weak headers appended, although this number is correlated with the T_w/T_s ratio. Finally, miners continue the mining process in order to find a strong header. In this process, a miner keeps creating candidate headers by computing hash values and checking whether the strong target is met. Every candidate header "protects" all collected weak headers (note that all of these weak headers point to the same previous strong header).

In order to keep the number of found weak headers close to a constant value, StrongChain adjusts the difficulty T_w of weak headers every 2016 blocks immediately following the adjustment of the difficulty T_s of the strong headers according to Equation 2.1, such that the ratio T_w/T_s is kept at a constant.

Block Layout and Validation

A block in our scheme consists of transactions, a list of weak headers, and a strong header that authenticates these transactions and weak headers. Strong and weak headers in our system inherit the fields from Bitcoin headers and additionally enrich it by a new field. A block header consists of the following fields:

- $PrevHash$: is a hash of the previous block header,
- $Target$: is the value encoding the current target defining the difficulty of finding new blocks,
- $Nonce$: is a nonce, used to generate PoW,
- $Timestamp$: is a Unix timestamp,
- $TxRoot$: is the root of the Merkle tree [Mer88] aggregating all transactions of the block, and
- $Coinbase$: represents an address of the miner that will receive a reward.

As our protocol rewards finders of weak headers (see details in Section 50), every weak header has to be accompanied with the information necessary to identify its finder. Otherwise, a finder of a strong block could maliciously claim that some (or all) weak headers

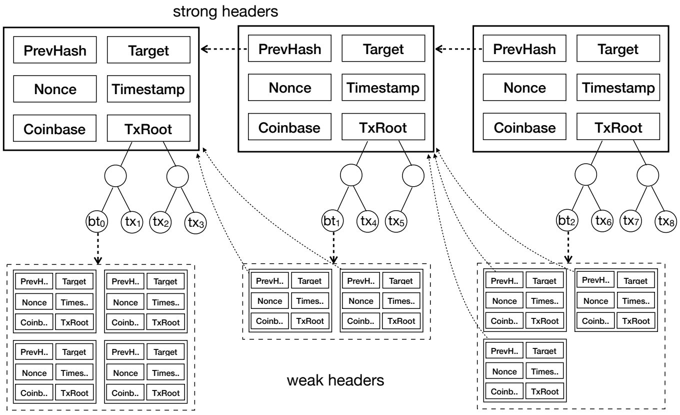


Figure 4.1: An example of a blockchain fragment with strong headers, weak headers, and binding and regular transactions.

were found by her and get rewards for them. For this purpose and for efficiency, we introduced a new 20B-long header field named *Coinbase*. With the introduction of this field, StrongChain headers are 100B long. But on the other hand, there is no longer any need for Bitcoin coinbase transactions.

Weak headers are exchanged among nodes as part of a block, hence it is necessary to protect the integrity of all weak headers associated with the block. To realize it, we introduce a special transaction, called a *binding transaction*, which contains a hash value computed over the weak headers. This transaction is the first transaction of each block and it protects the collected weak headers. Whenever a strong header is found, it is announced together with all its transactions and collected weak headers, therefore, this field protects all associated weak headers. To encode this field we utilize the `OP_RETURN` operation as follows:

$$\texttt{OP\_RETURN} \quad H(hdr_0 \| hdr_1 \| ... \| hdr_n), \tag{4.2}$$

where hdr_i is a weak header pointing to the previous strong header. Since weak headers have redundant fields (the *PrevHash*, *Target*, and *Version* fields have the same values as the strong header), we propose to save bandwidth and storage by not including these fields into the data of a block. This modification reduces the size of a weak header from 100B to 60B only, which is especially important for SPV clients who keep downloading new block headers.

With our approach, a newly mined and announced block can encompass multiple weak headers. Weak headers, in contrast to strong headers, are not used to authenticate transactions, and they are even stored and exchanged *without* their corresponding transactions. Instead, the main purpose of including weak headers is to contribute and reflect the aggregated mining power concentrated on a given branch of the blockchain. We present a fragment of a blockchain of StrongChain in Figure 4.1. As depicted in the figure, each block contains a single strong header, transactions, and a set of weak headers aggregated via a binding transaction.

On receiving a new block, miners validate the block by checking the following (see *validateBlock()* in Algorithm 1):

1. The strong header is protected by the PoW and points to the previous strong header.

2. Header fields have correct values (i.e., the version, target, and timestamp are set correctly).
3. All included transactions are correct and protected by the strong header. This check also includes checking that all weak headers collected are protected by a binding transaction included in the block.
4. All included weak headers are correct: a) they meet the targets as specified in Equation 4.1, b) their *PrevHash* fields point to the previous strong header, and c) their version, targets, and timestamps have correct values.

If the validation is successful, the block is accepted as part of the blockchain.

Forks

One of the main advantages of our approach is that blocks reflect their aggregated mining power more precisely. Each block beside its strong header contains multiple weak headers that contribute to the block's PoW. In the case of a fork, our scheme relies on the strongest chain rule, however, the PoW is computed differently than in Bitcoin. For every chain its PoW is calculated as presented by the *chainPoW()* procedure in Algorithm 1. Every chain is parsed and for each of its blocks the PoW is calculated by adding:

1. the PoW of the strong header, computed as T_{max}/T_s, where T_{max} is the maximum target value, and
2. the accumulated PoW of all associated weak headers, counting each weak header equally as T_{max}/T_w.

Then the chain's PoW is expressed as just the sum of all its blocks' PoW. Such an aggregated chain's PoW is compared with the competing chain(s). The chain with the largest aggregated PoW is determined as the current one. As difficulty in our protocol changes over time, the strong target T_s and PoW of weak headers are relative to the maximum target value T_{max}. We assume that nodes of the network check whether every difficulty window is computed correctly (we skipped this check in our algorithms for easy description).

Including and empowering weak headers in our protocol moves away from Bitcoin's "binary" granularity and gives blocks better expression of the PoW they convey. An example is presented in Figure 4.2. For instance, nodes having the blocks B_i and B_i' can immediately decide to follow the block B_i as it has more weak headers associated, thus it has accumulated more PoW than the block B_i'.

An exception to this rule is when miners solve conflicts. Namely, on receiving a new block, miners run the algorithm as presented, however, they also take into consideration PoW contributions of known weak headers that point to the last blocks. For instance, for a one-block-long fork within the same difficulty window, if a block B includes l weak headers and a miner knows of k weak headers pointing to B, then that miner will select B over any competing block B' that includes l' weak and has k' known weak headers pointing to it if $l + k > l' + k'$. Note that this rule incentivizes miners to propagate their solutions as quickly as possible as competing blocks become "stronger" over time.

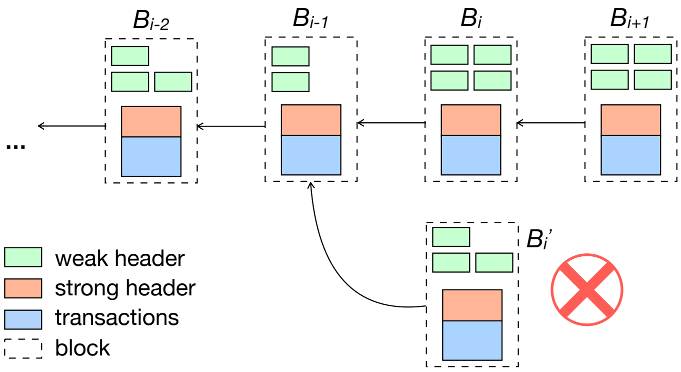


Figure 4.2: An example of a forked blockchain in StrongChain.

Rewarding Scheme

The rewards distribution is another crucial aspect of StrongChain and it is presented by the *rewardBlock()* procedure from Algorithm 1. The miner that found the strong header receives the full reward R. Moreover, in contrast to Bitcoin, where only the "lucky" miner is paid the full reward, in our scheme all miners that have contributed to the block's PoW (i.e., whose weak headers are included) are paid by commensurate rewards to the provided PoW. A weak header finder receive a fraction of R, i.e., $\gamma * c * R * T_s/T_w$, as a reward for its corresponding solution contributing to the total PoW of a particular branch, where the γ parameter influences the relative impact of weak header rewards and c is just a scaling constant. Moreover, we do not limit weak header rewards and miners can get multiple rewards for their weak headers within a single block. Similar reward mechanisms are present in today's mining pools, but unlike them, weak header rewards in StrongChain are independent of each other. Therefore, the reward scheme is not a zero-sum game and miners cannot increase their own rewards by dropping weak headers of others – they can only lose since their potential strong blocks would have less aggregated PoW without others' weak headers. Furthermore, weak header rewards decrease significantly the mining variance as miners can get steady revenue, making the system more decentralized and collaborative. In Table 4.1, we estimate a size reduction of the largest Bitcoin mining pools in 2019 with StrongChain while maintaining the same reward variance. In sum, StrongChain can offer 75x-105x size reduction.

As mentioned before, the number of weak headers of a block is unlimited, they are rewarded independently (i.e., do not share any reward), and all block rewards in our system are proportional to the PoW contributed. In such a setting, a mechanism incentivizing miners to terminate a block creation is needed (without such a mechanism, miners could keep creating huge blocks with weak headers only). In order to achieve this, StrongChain always attributes block transaction fees ($B.TxFees$) to the finder of the strong header (who also receives the full reward R).

Note that in our rewarding scheme, the amount of newly minted coins is always at least R, and consequently, unlike Bitcoin or Ethereum [Woo14a], the total supply of the currency in our protocol is not upper-bounded. This design decision is made in accordance with recent results on the long-term instability of deflationary cryptocurrencies [MB15, CKWN16a, TE18b].

Mining Pool	Pool Size		Size
	Bitcoin	StrongChain	Reduction
BTC.com	18.1%	0.245%	74×
F2Pool	14.1%	0.172%	82×
AntPool	11.7%	0.135%	87×
SlushPool	9.1%	0.099%	92×
ViaBTC	7.5%	0.079%	95×
BTC.TOP	7.1%	0.074%	96×
BitClub	3.1%	0.030%	103×
DPOOL	2.6%	0.025%	104×
Bitcoin.com	1.9%	0.018%	106×
BitFury	1.7%	0.016%	106×

Table 4.1: Largest Bitcoin mining pools and the corresponding pool sizes in StrongChain offering the same relative reward variance ($T_w/T_s = 1024$ and $\gamma = 10$).

Timestamps

In StrongChain, we follow the Bitcoin rules on constraining timestamps (see Section 2.7), however, we redefine how block timestamps are interpreted. Instead of solely relying on a timestamp put by the miner who mined the block, block timestamps in our system are derived from the strong header and all weak headers included in the corresponding block. The algorithm to derive a block's timestamp is presented as *getTimestamp()* in Algorithm 1. A block's timestamp is determined as a weighted average timestamp over the strong header's timestamp and all timestamps of the weak headers included in the block. The strong header's timestamp has a weight of 1, while weights of weak header timestamps are determined as their PoW contributed (namely, a weak header's timestamp has a weight of the ratio between the strong target and the weak target). Therefore, the timestamp value is adjusted proportionally to the mining power associated with a given block. That change reflects an average time of the block creation and mitigates miners that intentionally or misconfigured put incorrect timestamps into the blockchain.

SPV Clients

Our protocol supports light SPV clients. With every new block, an SPV client is updated with the following information:

$$hdr, hdr_0, hdr_1, ..., hdr_n, BTproof, \tag{4.3}$$

where *hdr* is a strong header, hdr_i are associated weak headers, and *BTproof* is an inclusion proof of a binding transaction that contains a hash over the weak headers (see Equation 4.2). Note that headers contain redundant fields, thus as described in Section 50, they can be provided to SPV clients efficiently.

With this data, the client verifies fields of all headers, computes the PoW of the block (analogous, as in *chainPoW()* from Algorithm 1), and validates the *BTproof* proof to check whether all weak headers are correct, and whether the transaction is part of the blockchain (the proof is validated against *TxRoot* of *hdr*). Afterward, the client saves the strong header *hdr* and its computed PoW, while other messages (the weak headers and the proof) can be dropped.

4.1.4 Experiments

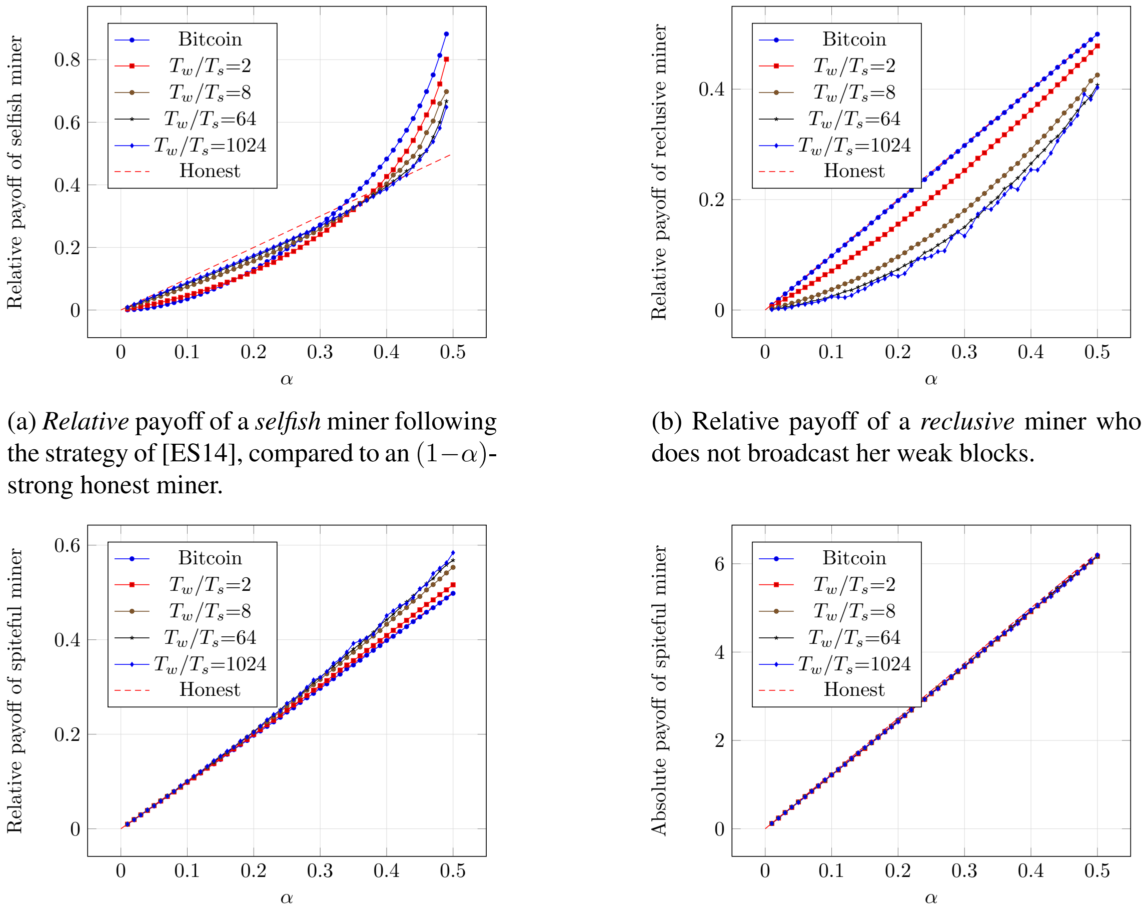
We executed a few experiments in the original paper. Nevertheless, for the purpose of this thesis we show only the results of experiments that demonstrate improved threshold of selfish mining profitability. We will consider the selfish mining strategy of [ES14], described as follows:

- The attacker does not propagate a newly found block until she finds at least a second block on top of it, and then only if the difference in difficulty between her chain and the strongest known alternative chain is between zero and R.
- The attacker adopts the strongest known alternative chain if its difficulty is at least greater than her own by R.

In Figure 4.3a, we have depicted the profitability of this selfish mining strategy for different choices of T_w/T_s. As we can see, for $T_w/T_s = 1024$ the probability of being 'ahead' after two strong blocks is so low that the strategy only begins to pay off when the attackers' mining power share is close to 43% — this is an improvement over Bitcoin, where the threshold is closer to 33%.

StrongChain does introduce new adversarial strategies based on the mining of new weak headers. Some examples include not broadcasting any newly found weak blocks ("reclusive" mining), refusing to include the weak headers of other miners ("spiteful" mining), and postponing the publication of a new strong block and wasting the weak headers found by other miners in the meantime. In the former case, the attacker risks losing their weak blocks, whereas in both of the latter two cases, the attacker risks their strong block going stale as other blocks and weak headers are found. Hence, these are not cost-free strategies. Furthermore, because the number of weak headers does not affect the difficulty rescale, the attacker's motive for increasing the stale rate of other miners' weak headers is less obvious (although in the long run, an adversarial miner could push other miners out of the market entirely, thus affecting the difficulty rescale).

In Figure 4.3b, we have displayed the relative payout (with respect to the total rewards) of a reclusive α-strong miner — this strategy does not pay for any $\alpha < 0.5$. In Figure 4.3c, we have depicted the relative payoff of a spiteful mine who does not include other miners' weak blocks unless necessary (i.e., unless others' weak blocks together contribute more than R to the difficulty, which would mean that any single block found by the spiteful miner would always go stale). For low latencies (the graphs were generated with an average latency of 0.53 seconds), the strategy is almost risk-free, and the attacker does manage to hurt other miners more than herself, leading to an increased relative payout. However, as displayed in Figure 4.3d, there are no absolute gains, even mild losses. As mentioned earlier, the weak headers do not affect the difficulty rescale so there is no short-term incentive to engage in this behavior — additionally there is little gain in computational overhead as the attacker still needs to process her own weak headers.



(a) *Relative* payoff of a *selfish* miner following the strategy of [ES14], compared to an $(1-\alpha)$-strong honest miner.

(b) Relative payoff of a *reclusive* miner who does not broadcast her weak blocks.

(c) *Relative* payoff (with respect to the rewards of all miners combined) of a *spiteful* miner, who does not include other miners' weak blocks unless necessary.

(d) *Absolute* payoff of a *spiteful* miner, with 12.5 BTC on average awarded per block.

Figure 4.3: Payoffs of an α-strong adversarial miner for different strategies. We consider Bitcoin and StrongChain with different choices of T_w/T_s, with $\gamma = \log_2(T_w/T_s)$.

4.2 Incentive Attacks on DAG-Based Blockchains

4.2.1 Problem Definition & Scope

Let there be a PoW blockchain network that uses the Nakamoto consensus (NC) and consists of honest and greedy miners, with the greedy miners holding a fraction adversarial-mining-power of the total mining power (i.e., adversarial mining power). Then, we denote the network propagation delay in seconds as network-propagation-delay and the block creation time in seconds as block-creation-rate. We assume that the minimum value of block-creation-rate is constrained by network-propagation-delay of the blockchain network. It is well-known that Nakamoto-style blockchains generate stale blocks (a.k.a., orphan blocks). As a result, a fraction of the mining power is wasted. The rate at which stale blocks are generated increases when block-creation-rate is decreased, which is one of the reasons why Bitcoin maintains a high block-creation-rate of 600s.

DAG-Oriented Designs

Many DAG-oriented designs were proposed to allow a decrease of block-creation-rate while utilizing stale blocks in parallel chains, which should increase the transaction throughput. Although there are some dag-oriented designs that do not address the problem of increasing transaction throughput (e.g., IoTA [SM20], Nano [LeM18], Byteball [Ant16]), we focus on the specific group of solutions addressing this problem, such as Inclusive [LSZ15], GHOSTDAG, PHANTOM [SWZ21], SPECTRE [SLZ16], and Prism [BKT$^+$19]. We are targeting the RTS strategy, which is a common property of this group of protocols. In the RTS, the miners do not take into account transaction fees of all included transactions; instead, they select transactions to blocks randomly – although not necessarily uniformly at random (e.g., [Som22]). In this way, these designs aim to eliminate transaction collision within parallel blocks of the dag structure. Nevertheless, the interpretation of randomness in RTS is not enforced/verified by these designs, and miners are trusted to ignore fees of all (or the majority (e.g., [Som22]) of) transactions for the common "well-being" of the protocol. Contrary, miners of blockchains such as Bitcoin use a well-known transaction selection mechanism that maximizes profit by selecting transactions of the block based on the highest fees – we refer to this strategy as the *greedy strategy* in this work.

Assumptions

We assume a generic DAG-oriented consensus protocol using the RTS strategy (denoted as DAG-PROTOCOL). Then, we assume that the incentive scheme of DAG-PROTOCOL relies on transaction fees (but additionally might also rely on block rewards),[7] and transactions are of the same size.[8] Let us assume that the greedy miners may only choose a different transaction selection strategy to make more profit than honest miners. Then, we assume that DAG-PROTOCOL uses rewarding where the miner of the block phantom-block gets rewarded for all unique not-yet-mined transactions in phantom-block (while she is not rewarded for transaction duplicates mined before).

Identified Problems – Incentive Attacks

Although the assumptions stated above might seem intuitive, there is no related work studying the impact of greedy miners deviating from the RTS strategy on any of the considered DAG-PROTOCOLs ([SWZ21],[SLZ16],[LSZ15],[BKT$^+$19]) and the effect it might have on the throughput of these protocols as well as a fair distribution of earned rewards. Note that we assume GHOSTDAG, PHANTOM, and SPECTRE are utilizing the RTS strategy that was proposed in the Inclusive protocol [SLZ16], as recommended by the (partially overlapping) authors of these works – this is further substantiated by the

[7] Note that block rewards would not change the applicability of our incentive attacks, and the constraints defined in the game theoretic model (see Section 4.2.2) would remain met even with them.

[8] Note that this assumption serves only for simplification of the follow-up sections. Transactions of different sizes would require normalizing fees by the sizes of transactions to obtain an equivalent setup (i.e., a fee per Byte).

practical implementation of GHOSTDAG/PHANTOM called Kaspa [Som22], which utilizes a variant of RTS strategy that selects a majority portion of transactions in a block uniformly random, while a small portion of the block capacity is seized by the transaction selected based on the highest fees. Nevertheless, besides potentially increased transaction collision rate, even such an approach enables more greedy behavior. We make a hypothesis for our incentive attacks:

Hypothesis 1. *A greedy transaction selection strategy will decrease the relative profit of honest miners as well as transaction throughput in the* DAG-PROTOCOL.[9]

4.2.2 Game Theoretical Analysis

In this section, we model a DAG-PROTOCOL[10] as a two-player game, in which the honest player/phenotype (P_{hon}) uses the RTS strategy and the greedy player/phenotype (P_{grd}) uses the greedy transaction selection strategy. We assume that the fees of transactions vary – the particular variance of fees is agnostic to this analysis. We present the game theoretical approach widely used to analyze interactions of players (i.e., consensus nodes) in the blockchain. Several works attempted to study the outcomes of different scenarios in blockchain networks (e.g., [LLW$^+$19, WBW$^+$21, SDS$^+$20]) but none of them addressed the case of DAG-PROTOCOLS and their transaction selection. In game theoretic terms, we examine the following hypothesis:

Hypothesis 2. *So-called (honest) honest-behavior with RTS is a Subgame Perfect Nash Equilibrium (SPNE) in an infinitely repeated* DAG-PROTOCOL *game. This was presented in Inclusive [LSZ15] and we will contradict it.*

Model of the DAG-PROTOCOL

Players in DAG-PROTOCOL receive transaction fees after a delay. To simplify analysis, we can divide the flow of transactions into rounds of the game. This allows us to study player behavior within defined time. In each round, players make decisions and receive payoffs. Since no round is explicitly marked as the last one, this game is repeated infinitely.

We model DAG-PROTOCOL in the form of *an infinitely repeated two players game with a base game*

$$\Gamma = (\{P_{hon}, P_{grd}\}; \{honest, greedy\}; U_{hon}, U_{mal}), \tag{4.4}$$

where P_{hon} is the player's determination to play honest strategy and P_{grd} the player's determination to the greedy-behavior. Pure strategy honest is interpreted as the RTS, while greedy strategy represents picking the transactions with the highest fees. Payoff functions

[9] Note that the greedy transaction selection strategy deviates from the DAG-PROTOCOL and thus is considered adversarial.

[10] Note that we consider DAG-based designs under this generic term of DAG-PROTOCOLS to simplify the description but not to claim that all DAG-PROTOCOLS (with RTS) can be modeled as we do.

are depicted in Table 4.2, where the profits in the strategic profiles $(honest, honest)$ and $(greedy, greedy)$ are uniformly distributed between players. In the following, we analyze the model in five possible scenarios with generic levels a, b, c, d of the payoffs.

Analysis of the Model

For purposes of our analysis, lets start with the assumption that greedy-behavior is more attractive and profitable than honest-behavior. Otherwise, there would be no reason to investigate Hypothesis 2. Thus, let us consider $c > a$ as the basic constraint. We also assume $c > b$, meaning that honest-behavior loses against greedy-behavior in the cases of $(honest, greedy)$ and $(greedy, honest)$ profiles. These basic constraints yield the following scenarios:

- **Scenario 1**: $d > c > a > b$,
- **Scenario 2**: $c > d > a > b$,
- **Scenario 3**: $c > a > d > b$,
- **Scenario 4**: $c > a > b > d$,
- **Scenario 5**: where $a = d$ and $c > a$, $c > b$.

Note that we do not assume the case $a = b$ since the presence of P_{grd} will drain all high-fee transactions that P_{hon} would originally obtain.
The following provides a high-level summary of the scenarios. For a more comprehensive analysis, we refer the refer to the full version of our paper [PHB$^+$23].

- Scenarios 1 and 2 are covered just for a sake of completeness. If the transaction fees were to cause such game outcomes, there would be no need to trust in honest-behavior, and the system would settle in the unique $(greedy, greedy)$ Pure Nash Equilibrium (PNE).

- Scenario 3A **Purely Non-Cooperative Interpretation**. In Scenario 3, both players (P_{hon} and P_{grd}) are incentivized to choose the greedy greedy strategy, even though this leads to a worse overall outcome for both of them. This is because each player can do better by betraying the other player than by cooperating. This situation is known as a *Prisoner's dilemma* [OR94].

  Proof. (Informal) Strategy $greedy$ strictly dominates $honest$ and thus ($greedy$, $greedy$) is the unique PNE. □

  Corollary 1. *If P_{hon} is willing to follow the social norm of using the DAG protocol, then P_{grd}'s best response is also to use the $greedy$ strategy. This is because P_{hon}'s cooperation is not credible, and P_{grd} can always benefit from betraying P_{hon}.*

- Scenario 3B **When Some Coordination is Allowed**. It is possible for players to coordinate their behavior and achieve a better outcome for both of them, both playing $honest$ strategy. It must be common knowledge to the players that P_{hon} uses *grim trigger strategy* [OR94, MS06]. This means that P_{hon} will cooperate as long as P_{grd} cooperates. However, if P_{grd} defects even once (playing $greedy$), then P_{hon} will

P_{hon}/P_{grd}	honest	greedy
honest	(a,a)	(b,c)
greedy	(c,b)	(d,d)

Table 4.2: The utility functions U_{hon}, U_{mal} in the *base game.*

switch to the $greedy$ strategy forever. P_{grd} must also have a high discount factor. This means that she must value future payoffs more than immediate payoffs. If P_{grd}'s discount factor is too low, then she will be tempted to defect even if she knows it will lead to punishment in the long run.

- Scenario 4A **Purely Non-Cooperative Interpretation**. We choose utility functions: $a = 2$, $b = 1$, $c = 3$ and $d = 0$. This scenario is an anti-coordination game [OR94] instance, so the game has two PNEs $(honest, greedy)$ & $(greedy, honest)$, and one Mixed Nash Equilibrium (MNE) in mixed strategic profile $\left((\frac{1}{2}, \frac{1}{2}), (\frac{1}{2}, \frac{1}{2})\right)$.

  Claim 1. *The most reasonable behavior in Scenario 4 is to play* $(\frac{1}{2}, \frac{1}{2})$ *for both players.*

  Proof. (Informal) Both players have two equally good choices: either be honest or be greedy. From P_{hon}'s perspective, mixed behavior $(\frac{1}{2}, \frac{1}{2})$ guarantees the best stable outcome. If P_{grd} expects $(\frac{1}{2}, \frac{1}{2})$ behavior from P_{hon}, then P_{grd}'s best response is to play the same mixed behavior that establishes MNE. The players gain $(\frac{3}{2}, \frac{3}{2})$ in that MNE, which is the highest expectation they can obtain. □

  Therefore, the most reasonable behavior for both players is to play a mixed strategy where they are half-honest and half-greedy.

- Scenario 4B **When Some Coordination is Allowed**. Similarly to Scenario 3, it is possible for players to coordinate their behavior and agree to always be $honest$. This would be a good outcome for both players, as they would both get a payoff of 2. The same principle and consequences apply as in scenario 3(B) (Grimm trigger strategy). This will make the P_{grd} player regret defecting, and it will make her more likely to cooperate in the future. Therefore, the conclusion from Scenario 3 applies here as well.

- Scenario 5A **Purely Non-Cooperative Interpretation**. In this scenario, the game is a zero-sum game, which means that no player can gain more than 100% profit, regardless of their chosen strategy. This is because the sum of all incoming transaction fees is fixed in any set of rounds. As a result, the total profit for all players is always constant if they all play the honest or greedy strategy. Therefore, the only rational outcome of this scenario is for both players to play the $greedy$ strategy.

  If we consider a social norm, it may be tempting to appeal to players' sense of responsibility and ask them to refrain from playing the $greedy$ strategy. However, this is unlikely to be effective, as the $honest$ strategy does not benefit either player. Scenario 5 is highly similar to the classic game-theoretical model called *The Tragedy* *of Commons* [Mil03]. In this model, individuals are incentivized to use a shared resource to the maximum extent possible, even if this depletes the resource and harms the group as a whole. In anonymous environments, where individuals cannot be held accountable for their actions, it is even more likely that they will prioritize their own interests over the interests of the group. This is because they know that they will not be punished for acting in their self-interest, meaning there is no harm to play $greedy$ strategy.

Summary

We conclude that Hypothesis 2 is not valid. The $(honest, honest)$ profile is not a PNE in any of our scenarios. Incentives enforcing honest-behavior are hardly feasible in the anonymous (permissionless) environment of blockchains. A community of honest miners can follow the DAG-PROTOCOL until the attacker appears. The attacker playing the greedy strategy can parasite on the system and there is no defense against such a behavior (since greedy miners can leave the system anytime and mine elsewhere, which is not assumed in [LSZ15]). Therefore, honest is not an *evolutionary stable strategy* [Smi82], and thus honest does not constitute a stable equilibrium. For more details about game theoretical analysis, we refer the reader to the full extended version of our paper [PBH+], which is not yet published.

4.2.3 Simulation Model

We created a simulation model to conduct various experiments investigating the behavior of DAG-PROTOCOL under incentive attacks related to the problems identified in Section 4.2.1 and thus Hypothesis 1. Some experiments were designed to provide empirical evidence for the conclusions from Section 4.2.2.

Abstraction of DAG-PROTOCOL

For evaluation purposes, we simulated the DAG-PROTOCOL (with RTS) by modeling the following aspects:

- All blocks in DAG are deterministically ordered.
- The mining rewards consist of transaction fees only.
- A fee of a particular transaction is awarded only to a miner of the block that includes the transaction as the first one in the sequence of totally ordered blocks.

Also, in terms of PHANTOM/GHOSTDAG terminology, we generalize and do not reduce transaction fees concerning the delay from “appearing” of the block until it is strongly connected to the DAG. Hence, we utilize discount-function = 1. In other words, for each block phantom-block, the discount function does not penalize a block according to its gap parameter $gap - parameter(phantom - block)$, i.e. $discount - function(gap - parameter(phantom - block)) = 1$. Such a setting is optimistic for honest miners and maximizes their profits from transaction fees when following the RTS strategy. This abstraction enables us to model the concerned problems of considered DAG-PROTOCOLS.

(Simple) Network Topology

We created a simple network topology that is convenient for proof-of-concept simulations and encompasses some important aspects of the real-world blockchain network. In particular, we were interested in emulating the network propagation delay network-propagation-delay to be similar to Bitcoin (i.e., $\sim 5s$ at most of the time in 2022), but using a small ring topology. To create such a topology, we assumed that the Bitcoin network contains 7592 nodes, according to the snapshot of reachable Bitcoin nodes found on May 24, 2022.[11] In Bitcoin core, the default value of the consensus node's peers is set to 8 (i.e., the node degree).[12] Therefore, the maximum number of hops that a gossiped message requires to reach all consensus nodes in the network is ~ 4.29 (i.e., $log_8(7592)$). Moreover, if we were to assume $2 - 3x$ more independent blockchain clients (that are not consensus nodes), then this number would be increased to 4.83–4.96. To model this environment, we used the ring network topology with 10 consensus nodes, which sets the maximum value of hops required to propagate a message to 5. Next, we set the inter-node propagation delay $\partial\tau$ to $1s$, which fits assumed network-propagation-delay (i.e., 5s / 5 hops = 1s).

Simulator

There are simulators [PGF21] that model blockchain protocols, mainly focusing on network delays, different consensus protocols, and behaviors of specific attacks (e.g., SimBlock [AOK$^+$19], Blocksim [AvM19], Bitcoin-Simulator [GG16]). However, none of these simulators was sufficient for our purposes due to missing support for multiple chains and incentive schemes assumed in DAG-PROTOCOLS. To verify Hypothesis 1, we built a simulator that focuses on the mentioned problems of DAG-PROTOCOLS. In detail, we started with the Bitcoin mining simulator [Gav15], which is a discrete event simulator for the PoW mining on a single chain, enabling a simulation of network propagation delay within a specified network topology. We extended this simulator to support DAG-PROTOCOLS, enabling us to monitor transaction duplicity, throughput, and relative profits of miners with regard to their mining power. The simulator is written in `C++` (see details and its evaluation in [PHMH23]. In addition, we added more simulation complexity to simulate each block, including the particular transactions (as opposed to simulating only the number of transactions in a block [Gav15]). Most importantly, we implemented two different transaction selection strategies – greedy and random. For demonstration purposes, we implemented the exponential distribution of transaction fees in mempool, based on several graph cuts of fee distributions in mempool of Bitcoin from [Hoe22].[13] Our simulator is available at `https://github.com/Tem12/DAG-simulator`.

[11] `https://bitnodes.io/nodes/`

[12] Nevertheless, the node degree is often higher than 8 in reality [MMC$^+$19].

[13] Distribution of transaction fees in mempool might change over time; however, it mostly preserves the low number of high-fee transactions.

4.2.4 Evaluation

We designed a few experiments with our simulator, which were aimed at investigating the relative profit of greedy miners and transaction collision rate (thus throughput) to investigate Hypothesis 1. In all experiments, honest miners followed the RTS, while greedy miners followed the greedy strategy. Unless stated otherwise, the block creation time was set to $block-creation-rate = 20s$. However, we abstracted from network-propagation-delay of transactions and ensured that the mempools of nodes were regularly filled (i.e., every 60s) by the same set of new transactions, while the number of transactions in the mempool was always sufficient to fully satisfy the block capacity that was set to 100 transactions. We set the size of mempool equal to 10000 transactions, and thus the ratio between these two values is similar to Bitcoin [Hoe22] in common situations. In all experiments, we executed multiple runs and consolidated their results; however, in all experiments with the simple topology, the spread was negligible, and therefore we do not depict it in graphs.

Experiment I

Goal. The goal of this experiment was to compare the relative profits earned by two miners/phenotypes in a network, corresponding to our game theoretical settings (see Section 4.2.2). Thus, one miner was greedy and followed the greedy strategy, while the other one was honest and followed the RTS.

Methodology and Results. The ratio of total mining power between the two miners was varied with a granularity of 10%, and the network consisted of 10 miners, where only the two miners had assigned the mining power. Other miners acted as relays, emulating the maximal network delay of 5 hops between the two miners in a duel. The relative profits of the miners were monitored as their profit factor $\mathbb{P}$ w.r.t. their mining power. We conducted 10 simulation runs and averaged their results (see Figure 4.4). Results show that the greedy miner earned a profit disproportionately higher than her mining power, while the honest miner's relative profit was negatively affected by the presence of the greedy miner. We can observe that $\mathbb{P}$ of greedy miner was indirectly proportional to her adversarial-mining-power, which was caused by the exponential distribution of transaction fees that contributed more significantly to the higher $\mathbb{P}$ of a smaller miner. In sum, the profit advantage of the greedy miner aligns with the conclusions from the game theoretical model (Scenario 5, see Section 4.2.2) in particular, which represents the case of adversarial-mining-power=50%. Nevertheless, our results indicate that the greedy strategy is more profitable than the RTS for any non-zero adversarial-mining-power.

Experiment II

Goal. The goal of this experiment was investigation of the relative profits of a few greedy miners following the greedy strategy in contrast to honest miners following the RTS.

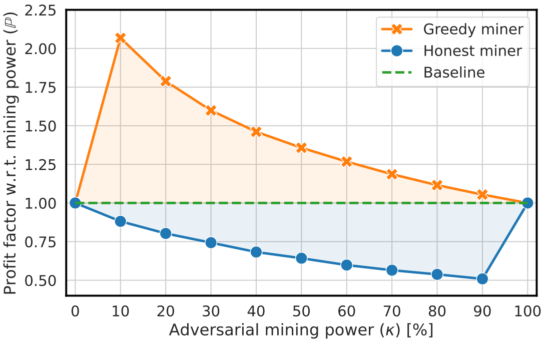


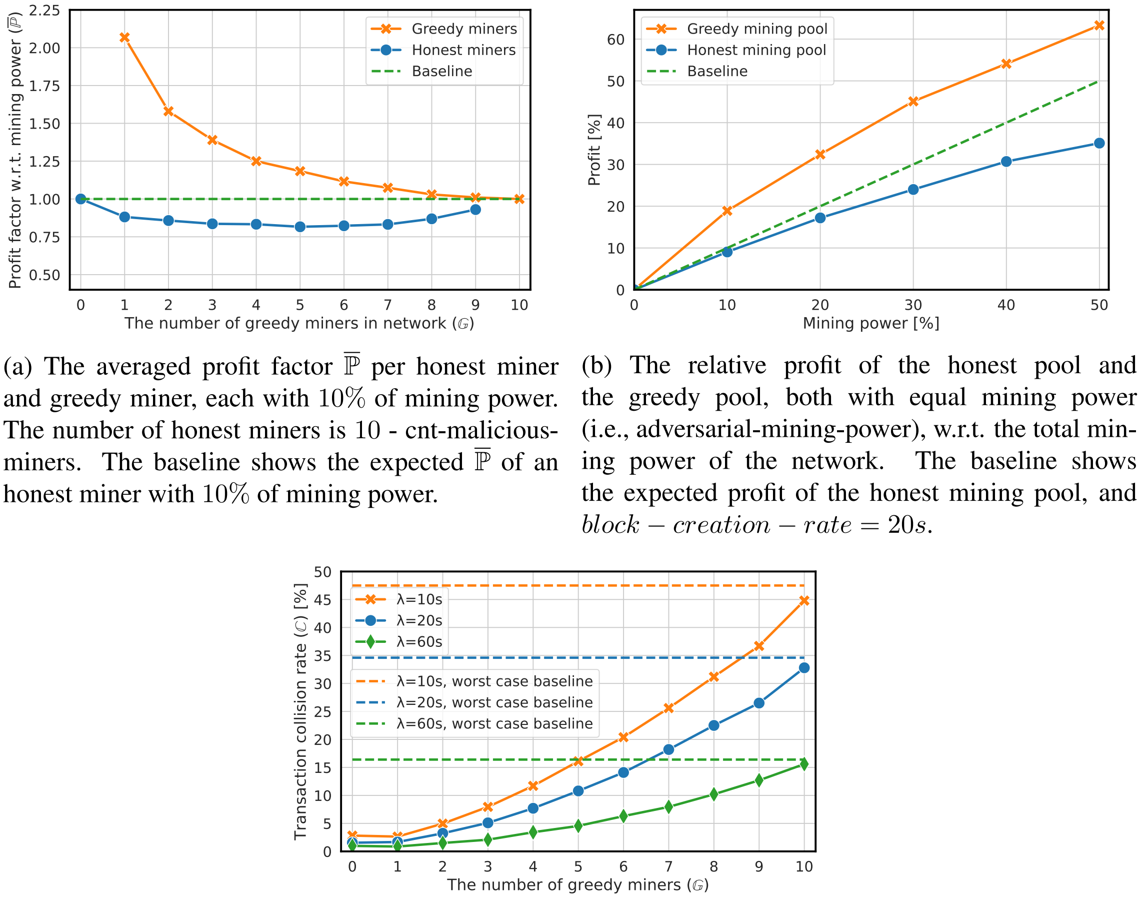
Figure 4.4: The profit factor $\mathbb{P}$ of an honest vs. a greedy miner with their mining powers of 100% - adversarial-mining-power and adversarial-mining-power, respectively. The baseline shows the expected $\mathbb{P}$ of the honest miner; $block - creation - rate = 20s$.

Methodology and Results. We experimented with 10 miners, where the number of greedy miners $cnt - malicious - miners$ vs. the number of honest miners (i.e., 10 - cnt-malicious-miners) was varied, and each held 10% of the total mining power. We monitored their profit factor $\overline{\mathbb{P}}$ averaged per miner. We conducted 10 simulation runs and averaged their results (see Figure 4.5a). Alike in Section 4.2.4, we can see that greedy miners earned profit disproportionately higher than their mining power. Similarly, this experiment showed that the profit advantage of greedy miners decreases as their number increases. This is similar to increasing adversarial-mining-power in a duel of two miners from Section 4.2.4; however, in contrast to it, $\overline{\mathbb{P}}$ of greedy miners is slightly lower with the same total adversarial-mining-power of all greedy miners, while $\overline{\mathbb{P}}$ of honest miners had not suffered with such a decrease. Intuitively, this happened because multiple greedy miners increase transaction collision. In detail, since miners are only rewarded for transactions that were first to be included in a new block, the profit for the second and later miners is lost if a duplicate transaction is included. This observation might be seen as beneficial for the protocol as it disincentivizes multiple miners to use the greedy transaction selection strategy, which would support the sequential equilibrium from [LSZ15]. However, the authors of [LSZ15] do not assume cooperating players, which is unrealistic since miners can cooperate and create the pool to avoid collisions and thus maximize their profits (resulting in a similar outcome, as in Section 4.2.4).

Experiment III

Goal. The goal of this experiment was to investigate the relative profit of the greedy mining pool depending on its adversarial-mining-power versus the honest mining pool with the same mining power. It is equivalent to Scenario 5 of game theoretical analysis (see Section 4.2.2) although there is the honest rest of the network.

Methodology and Results. We experimented with 10 miners, and out of them, we choose one greedy miner and one honest miner, both having equal mining power, while the remaining miners in the network were honest and possessed the rest of the network's mining power. Thus, we emulated a duel of the greedy pool versus the honest pool.



(a) The averaged profit factor $\overline{\mathbb{P}}$ per honest miner and greedy miner, each with 10% of mining power. The number of honest miners is 10 - cnt-malicious-miners. The baseline shows the expected $\overline{\mathbb{P}}$ of an honest miner with 10% of mining power.

(b) The relative profit of the honest pool and the greedy pool, both with equal mining power (i.e., adversarial-mining-power), w.r.t. the total mining power of the network. The baseline shows the expected profit of the honest mining pool, and $block-creation-rate = 20s$.

(c) The transaction collision rate $\mathbb{C}$ w.r.t. # of greedy miners cnt-malicious-miners (each with adversarial-mining-power = 10%), where # of honest miners was $10-cnt-malicious-miners$ and block-creation-rate $\in \{10s, 20s, 60s\}$. The worst case baseline shows $\mathbb{C}$ when all transactions are duplicates.

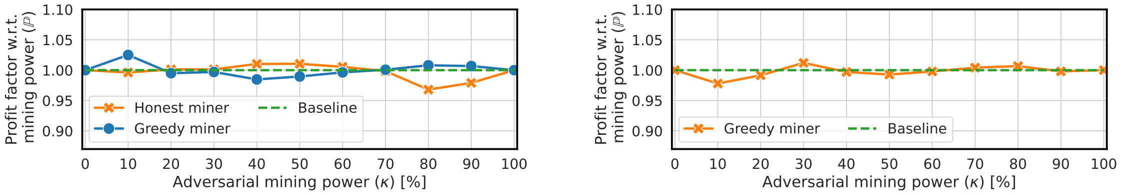
Figure 4.5: Experiment II, Experiment III (i.e., duel of mining pools) and Experiment IV (i.e., transaction collision rate & throughput).

We conducted 10 simulation runs and averaged their results (see Figure 4.5b). The results demonstrate that the greedy pool's relative earned profit grows proportionally to adversarial-mining-power as compared to the honest pool with equal mining power, supporting our conclusions from Section 4.2.2.

Experiment IV

Goal. The goal of this experiment was to investigate the transaction collision rate under the occurrence of greedy miners who selected transactions using the greedy strategy.

Methodology and Results. In contrast to the previous experiments, we considered three different values of block creation time (block-creation-rate $\in \{10s, 20s, 60s\}$). We exper-



(a) The profit factor $\mathbb{P}$ of a honest vs. a greedy miner with the mining power of 100% - adversarial-mining-power and adversarial-mining-power, respectively.

(b) The averaged profit factor $\overline{\mathbb{P}}$ of a greedy miner with adversarial-mining-power. The rest of the network consisted of 9 honest miners, each equipped with $\frac{100\%-adversarial-mining-power}{9}\%$ of mining power.

Figure 4.6: Profit factors of honest and greedy miners. The baseline shows the expected $\mathbb{P}$ of the honest miner; $block-creation-rate = 20s$.

imented with 10 miners, where the number of greedy miners $cnt-malicious-miners$ vs. the number of honest miners (i.e., 10 - cnt-malicious-miners) was varied, and each held 10% of the total mining power. For all configurations, we computed the transaction collision rate (see Figure 4.5c). We can see that the increase of $cnt-malicious-miners$ causes the increase in the transaction collision rate. Note that lower block-creation-rate has a higher impact on the collision rate, and DAG protocols are designed with the intention to have small block-creation-rate (i.e., even smaller than network-propagation-delay). Consequently, the increased collision rate affected the overall throughput of the network (which is complementary to Figure 4.5c).

4.2.5 Experiments with Complex Topology

Additionally, we conducted more than 500 experiments in complex topology with 7592 nodes in various configurations (such as different connectivity and positions of greedy miners in the topology). We emulated weakly and strongly connected miners by setting a different node degree – we utilized a node degree distribution from [MMC$^+$19] and projected it into our network by setting the weakly connected edge and a highly connected core. The results of these experiments confirm the conclusions from the game theoretic analysis (see Section 4.2.2) as well as they match the experiments with the simple topology (see Section 4.2.4). The details of these experiments are presented in the extended version of our paper [PBH+], which is not yet published.

4.2.6 Countermeasures

Experiments supported Hypothesis 1. The main problem is **not sufficiently enforcing the RTS**, i.e., verifying that transaction selection was indeed random at the protocol level. Therefore, using the RTS in the DAG-PROTOCOL that does not enforce the interpretation of randomness will never avoid the occurrence of attackers from greedy transaction selection that increases their individual (or pooled) profits.

Enforcing Interpretation of the Randomness. One countermeasure how to avoid arbitrary interpretation of the randomness in the RTS is to enforce it by the consensus protocol. An example of a DAG-based design using this approach is Sycomore [AGLS18], which utilizes the prefix of cryptographically-secure hashes of transactions as the criteria for extending a particular chain in dag. The PoW mining in Sycomore is further equipped with the unpredictability of a chain that the miner of a new block extends, avoiding the concentration of the mining power on "rich" chains. Note that transactions are evenly spread across all chains of the DAG, which happens because prefixes of transaction hashes respect the uniform distribution – transactions are created by clients (different from miners) who have no incentives for biasing their transactions.

Fixed Transaction Fees. Another option how to make the RTS viable is to employ fixed fees for all transactions as a blockchain network-adjusted parameter. In the case of the full block capacity utilization within some period, the fixed fee parameter would be increased and vice versa in the case of not sufficiently utilized block capacity. In contrast to the previous countermeasure, this mechanism does not enforce the interpretation of randomness while at the same time does not make incentives for greedy miners to follow other than the RTS strategy. Therefore, miners using other than the RTS would not earn extra profits – we demonstrate it in Figure 4.6a and Figure 4.6b, considering one honest vs. one greedy miner and one greedy vs. 9 honest miners, respectively. Note that small deviations from the baseline are caused by the inherent simulation error that is present in the original simulator that we extended. On the other hand, greedy miners may still cause increased transaction collision rate, and thus decreased throughput. Therefore, we consider the fixed transaction fee option weaker than the previous one.

4.3 Undercutting Attacks

In transaction fee-based regime schemes, a few problems have emerged, which we can observe even nowadays in Bitcoin protocol [CKWN16a]. We have selected three main problems and aim to lower their impact for protocols relying on transaction fees only. In detail, we focus on the following problems:

1. **Undercutting attack.** In this attack (see Figure 4.7), a malicious miner attempts to obtain transaction fees by re-mining a top block of the longest chain, and thus motivates other miners to mine on top of her block [CKWN16a]. In detail, consider a situation, where an honest miner mines a block containing transactions with substantially higher transaction fees than is usual. The malicious miner can fork this block while he leaves some portion of the "generous" transactions un-mined. These transactions motivate other miners to mine on top of the attacker's chain, and thus undercut the original block. Such a malicious behavior might result in higher orphan rate, unreliability of the system, and even double spending.

2. **The mining gap.** As discussed in [CKWN16a], the problem of mining gap arises once the mempool does not contain enough transaction fees to motivate miners in

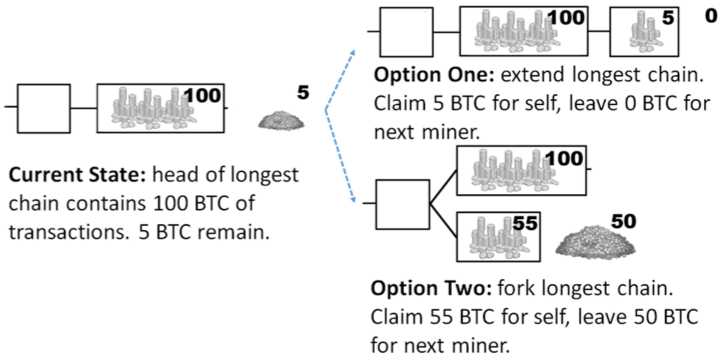

100
5
Current State: head of longest chain contains 100 BTC of transactions. 5 BTC remain.
100
5
0
Option One: extend longest chain. Claim 5 BTC for self, leave 0 BTC for next miner.
100
55
50
Option Two: fork longest chain. Claim 55 BTC for self, leave 50 BTC for next miner.

Figure 4.7: The undercutting attack, according to Carlsten et al. [CKWN16a].

mining. Suppose a miner succeeds at mining a new block shortly after the previous block was created, which can happen due to well known exponential distribution of block creation time in PoW blockchains. Therefore, the miner might not receive enough rewards to cover his expenses because most of the transactions from the mempool were included in the previous block, while new transactions might not have yet arrived or have small fees. Consequently, the miners are motivated to postpone mining until the mempool is reasonably filled with enough transactions (and their fees). The mining gap was also analyzed by the simulation in the work of Tsabary and Eyal [TE18a], who further demonstrated that mining gap incentivizes larger mining coalitions (i.e., mining pools), negatively impacting decentralization.

3. **Varying transaction fees over time.** In the transaction-fee-based regime, any fluctuation in transaction fees directly affects the miners' revenue. High fluctuation of transaction fees during certain time frames, e.g., in a span of a day or a week [Wik22], can lead to an undesirable lack of predictability in miners' rewards and indirectly affect the security of the underlying protocol.

4.3.1 Overview of Proposed Approach

We propose a solution that collects a percentage of transaction fees in a native cryptocurrency from the mined blocks into one or multiple fee-redistribution smart contracts (i.e., $\mathcal{FRSC}$s). Miners of the blocks who must contribute to these contracts are at the same time rewarded from them, while the received reward approximates a moving average of the incoming transaction fees across the fixed sliding window of the blocks. The fraction of transaction fees (i.e., $\mathbb{C}$) from the mined block is sent to the $\mathcal{FRSC}$ and the remaining fraction of transaction fees (i.e., $\mathbb{M}$) is directly assigned to the miner, such that $\mathbb{C}+\mathbb{M} = 1$. The role of $\mathbb{M}$ is to incentivize the miners in prioritization of the transactions with the higher fees while the role of $\mathbb{C}$ is to mitigate the problems of undercutting attacks and the mining gap. Our solution can be deployed with hard-fork and imposes only negligible performance overhead.

We depict the overview of our approach in Figure 3.1, and it consists of the following steps:

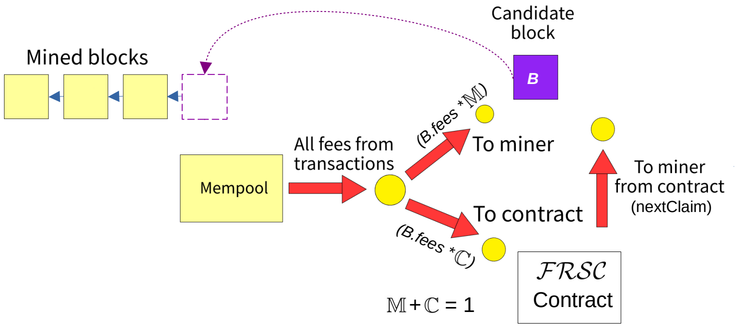


Figure 4.8: Overview of our solution.

1. Using $\mathcal{FRSC}$, the miner calculates the reward for the next block B (i.e., $nextClaim$-($\mathcal{FRSC}$) – see Equation 4.8) that will be payed by $\mathcal{FRSC}$ to the miner of that block.

2. The miner mines the block B using the selected set of the highest fee transactions from her mempool.

3. The mined block B directly awards a certain fraction of the transaction fees (i.e., $B.fees\ *\ \mathbb{M}$) to the miner and the remaining part (i.e., $B.fees\ *\ \mathbb{C}$) to $\mathcal{FRSC}$.

4. The miner obtains $nextClaim$ from $\mathcal{FRSC}$.

Our approach is embedded into the consensus protocol, and therefore consensus nodes are obliged to respect it in order to ensure that their blocks are valid. It can be implemented with standard smart contracts of the blockchain platform or within the native code of the consensus protocol.

4.3.2 Details of Fee-Redistribution Smart Contracts

We define the fee-redistribution smart contract as a tuple

$$\mathcal{FRSC} = (\nu, \lambda, \rho), \tag{4.5}$$

where ν is the accumulated amount of tokens in the contract, λ denotes the size of $\mathcal{FRSC}$'s sliding window in terms of the number of preceding blocks that contributed to ν, and ρ is the parameter defining the ratio for redistribution of incoming transaction fees among multiple contracts (if there are multiple $\mathcal{FRSC}$s), while the sum of ρ across all $\mathcal{FRSC}$s must be equal to 1:

$$\sum_{x\ \in\ \mathcal{FRSC}s} x.\rho \quad = \quad 1. \tag{4.6}$$

In contrast to a single $\mathcal{FRSC}$, we envision multiple $\mathcal{FRSC}$s to enable better adjustment of compensation to miners during periods of higher transaction fee fluctuations or in an unpredictable environment (we show this in Section 4.3.3).

We denote the state of $\mathcal{FRSC}$s at the blockchain height H as $\mathcal{FRSC}_{[H]}$. Then, we determine the reward from $\mathcal{FRSC}_{[H]} \in \mathcal{FRSC}s_{[H]}$ for the miner of the next block with height $H+1$ as follows:

$$\partial Claim_{[H+1]}^{\mathcal{FRSC}_{[H]}} = \frac{\mathcal{FRSC}_{[H]}.\nu}{\mathcal{FRSC}_{[H]}.\lambda}, \tag{4.7}$$

while the reward obtained from all $\mathcal{FRSC}$s is

$$nextClaim_{[H+1]} = \sum_{\mathcal{X}_{[H]} \,\in\, \mathcal{FRSC}s_{[H]}} \partial Claim_{[H+1]}^{\mathcal{X}_{[H]}}. \tag{4.8}$$

Then, the total reward of the miner who mined the block $B_{[H+1]}$ with all transaction fees $B_{[H+1]}.fees$ is

$$rewardT_{[H+1]} = nextClaim_{[H+1]} + \mathbb{M} * B_{[H+1]}.fees. \tag{4.9}$$

The new state of contracts at the height $H+1$ is

$$\mathcal{FRSC}s_{[H+1]} = \{\mathcal{X}_{[H+1]}(\nu, \lambda, \rho) \;| \tag{4.10}$$

$$\begin{aligned}
\lambda &= \mathcal{X}_{[H]}.\lambda, && (4.11)\\
\rho &= \mathcal{X}_{[H]}.\rho, && (4.12)\\
\nu &= \mathcal{X}_{[H]}.\nu - \partial Claim_{[H+1]} + deposit * \rho, && (4.13)\\
deposit &= B_{[H+1]}.fees * \mathbb{C}\}, && (4.14)
\end{aligned}$$

where $deposit$ represents the fraction $\mathbb{C}$ of all transaction fees from the block $B_{[H+1]}$ that are deposited across all $\mathcal{FRSC}$s in ratios respecting Equation 4.6.

Example

We consider Bitcoin [Nak08] with the current height of the blockchain H. We utilize only a single $\mathcal{FRSC}$:

$$\mathcal{FRSC}_{[H]} = (2016, 2016, 1).$$

We set $\mathbb{M} = 0.4$ and $\mathbb{C} = 0.6$, which means a miner directly obtains 40% of the $B_{[H+1]}.fees$ and $\mathcal{FRSC}$ obtains 60%. Next, we compute the reward from $\mathcal{FRSC}$ obtained by the miner of the block with height $H+1$ as

$$\partial Claim_{[H+1]} = \frac{\mathcal{FRSC}_{[H]}.\nu}{\mathcal{FRSC}_{[H]}.\lambda} = \frac{2016}{2016} = 1 \text{ BTC},$$

resulting into

$$nextClaim_{[H+1]} = \partial Claim_{[H+1]} \;=\; 1 \text{ BTC}.$$

Further, we assume that the total reward collected from transactions in the block with height $H + 1$ is $B_{[H+1]}.fees = 2$ BTC. Hence, the total reward obtained by the miner of the block $B_{[H+1]}$ is

$$\begin{aligned} rewardT_{[H+1]} &= nextClaim_{[H+1]} + \mathbb{M} * B_{[H+1]}.fees \\ &= 1 + 0.4 * 2 \ = \ 1.8 \text{ BTC}, \end{aligned}$$

and the contribution of transaction fees from $B_{[H+1]}$ to the $\mathcal{FRSC}$ is

$$deposit = B_{[H+1]}.fees * \mathbb{C} = 1.2 \text{ BTC}.$$

Therefore, the value of ν in $\mathcal{FRSC}$ is updated at height H + 1 as follows:

$$\begin{aligned} v_{[H+1]} &= \mathcal{FRSC}_{[H]}.\nu - nextClaim_{[H+1]} + deposit \\ &= 2016 - 1 + 1.2 \text{ BTC} \ = \ 2016.2 \text{ BTC}. \end{aligned}$$

Traditional Way in Tx-Fee Regime

In traditional blockchains, $rewardT_{[H+1]}$ would be equal to the sum of all transaction fees $B_{[H+1]}.fees$ (i.e., 2 BTC); hence, using $\mathbb{M} = 1$. In our approach, $rewardT_{[H+1]}$ is equal to the sum of all transaction fees in the block $B_{[H+1]}$, if:

$$B_{[H+1]}.fees = \frac{nextClaim_{[H+1]}}{\mathbb{C}}. \tag{4.15}$$

In our example, a miner can mine the block $B_{[H+1]}$ while obtaining the same total reward as the sum of all transaction fees in the block if the transactions carry 1.66 BTC in fees:

$$B_{[H+1]}.fees = \frac{1}{0.6} = 1.66 \text{ BTC}.$$

Initial Setup of $\mathcal{FRSC}$s Contracts

To enable an even start, we propose to initiate $\mathcal{FRSC}$s of our approach by a genesis value. The following formula calculates the genesis values per $\mathcal{FRSC}$ and initializes starting state of $\mathcal{FRSC}s_{[0]}$:

$$\{\mathcal{FRSC}^{x}_{[0]}(\nu, \lambda, \rho) \mid \nu = \overline{fees} * \mathbb{C} * \rho * \lambda\}, \tag{4.16}$$

where $\overline{fees}$ is the expected average of incoming fees.

4.3.3 Evaluation

We base on Bitcoin Mining Simulator [Kal15], introduced in [CKWN16a], which we modified for our purposes. We have created a configuration file to simulate custom scenarios of incoming transactions instead of the accumulated fees in the original design [CKWN16a]. We added an option to switch simulation into a mode with a full mempool, and thus bound the total fees (and consequently the total number of transactions) that can be earned within a block – this mostly relates to blocks whose mining takes longer time than the average time to mine a block.[14] Next, we moved several parameters to arguments of the simulator to eliminate the need for frequent recompilation of the program, and therefore simplified the process of running various experiments with the simulator. Finally, we integrated our $\mathcal{FRSC}$-based solution into the simulator. $\mathcal{FRSC}$s are initiated from a corresponding configuration file. The source code of our modified simulator is available at `https://github.com/The-Huginn/mining_simulator`.

Experiments. We evaluated our proof-of-concept implementation of $\mathcal{FRSC}$s on a custom long-term scenario designed to demonstrate significant changes in the total transaction fees in the mempool evolving across the time. This scenario is depicted in the resulting graphs of most of our experiments, represented by the "*Fees in mempool*" series – see Section 4.3.3 and Section 4.3.3.

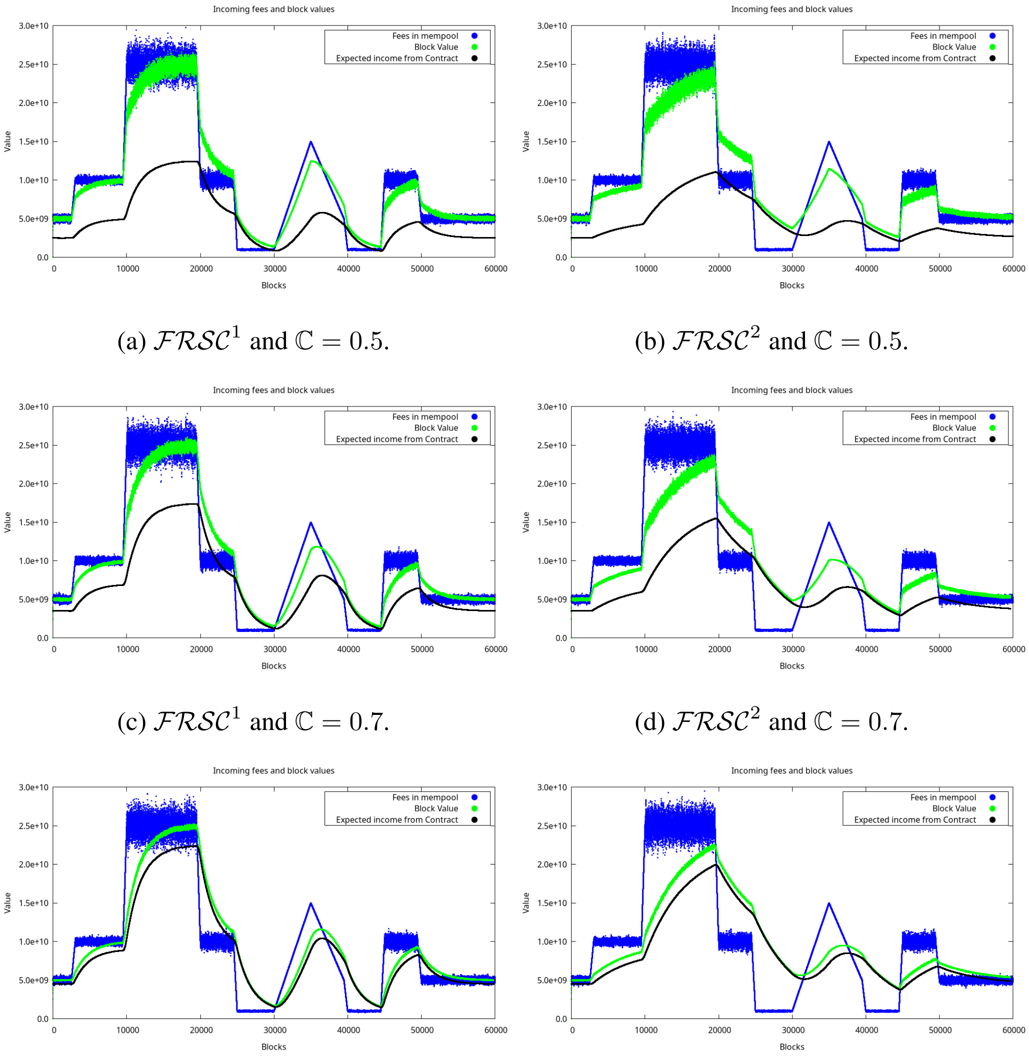
We experimented with different parameters and investigated how they influenced the total rewards of miners coming from $\mathcal{FRSC}$s versus the baseline without our solution. Mainly, these included a setting of $\mathbb{C}$ as well as different lengths λ of $\mathcal{FRSC}$s. For demonstration purposes, we used the value of transaction fees per block equal to 50 BTC, the same as Carlsten et al. [CKWN16a] used. Across all our experiments but the last one (i.e., Section 4.3.3), we enabled the full mempool option to ensure more realistic conditions.

Experiment I

Methodology. The purpose of this experiment was to investigate the amount of the reward a miner earns with our approach versus the baseline (i.e., the full reward is based on all transaction fees). We investigated how $\mathbb{C}$ influences the total reward of the miner and how λ of the sliding window averaged the rewards. In detail, we created two independent $\mathcal{FRSC}$s with different λ – one was set to 2016 (i.e., $\mathcal{FRSC}^1$), and the second one was set to 5600 (i.e., $\mathcal{FRSC}^2$). We simulated these $\mathcal{FRSC}$s with three values of $\mathbb{C} \in \{0.5,\ 0.7,\ 0.9\}$.

Results. The results of this experiment are depicted in Figure 4.9. Across all runs of our experiment, we can observe that $\mathcal{FRSC}^2$ adapts slower as compared to $\mathcal{FRSC}^1$, which leads to a more significant averaging of the total reward paid to the miner.

[14] Note that the original simulator [CKWN16a] assumes that the number of transactions (and thus the total fees) in the block is constrained only by the duration of a time required to mine the block, which was also criticized in [GMSK22].



(a) $\mathcal{FRSC}^1$ and $\mathbb{C} = 0.5$.

(b) $\mathcal{FRSC}^2$ and $\mathbb{C} = 0.5$.

(c) $\mathcal{FRSC}^1$ and $\mathbb{C} = 0.7$.

(d) $\mathcal{FRSC}^2$ and $\mathbb{C} = 0.7$.

(e) $\mathcal{FRSC}^1$ and $\mathbb{C} = 0.9$.

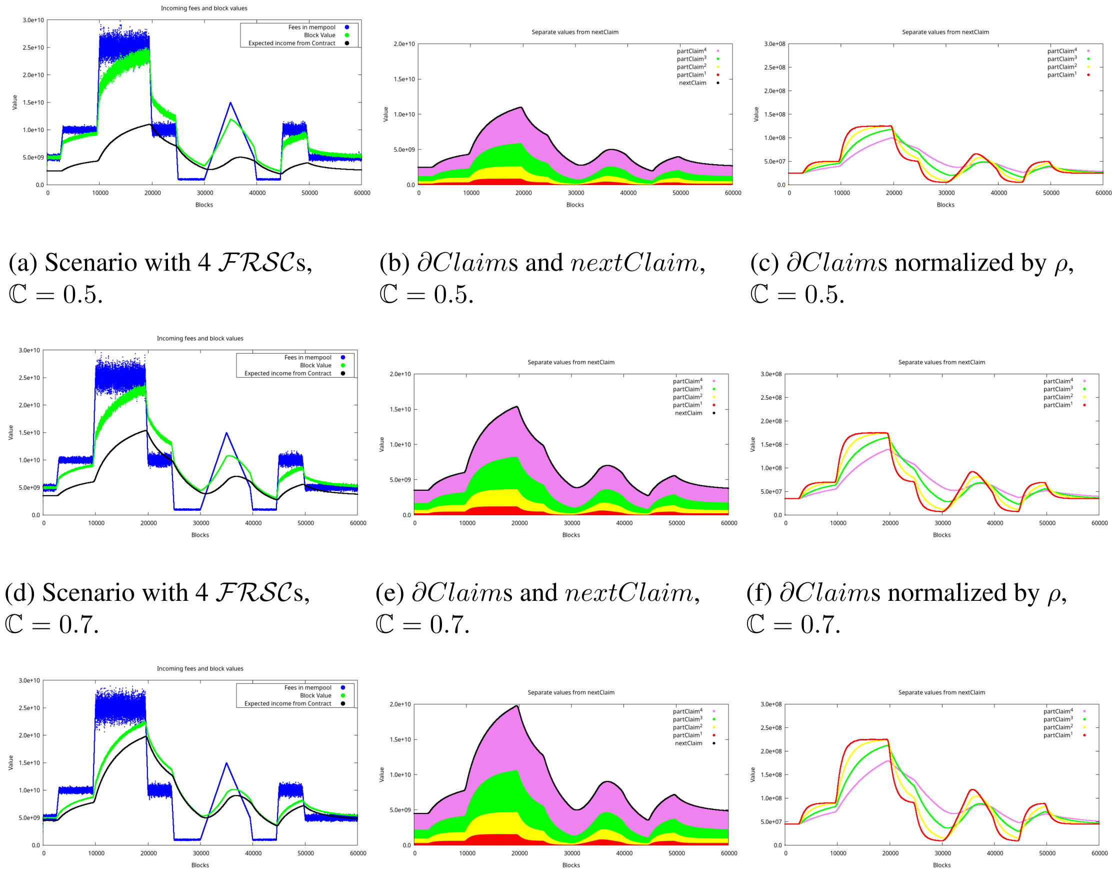
(f) $\mathcal{FRSC}^2$ and $\mathbb{C} = 0.9$.

Figure 4.9: Experiment I investigating various $\mathbb{C}s$ and λs of a single $\mathcal{FRSC}$, where $\mathcal{FRSC}^1.\lambda = 2016$ and $\mathcal{FRSC}^2.\lambda = 5600$. *Fees in mempool* show the total value of fees in the mined block (i.e., representing the baseline). *Block Value* is the reward a miner received in block B as a sum of the fees he obtained directly (i.e. $\mathbb{M} * B.fees$) and the reward he got from $\mathcal{FRSC}$ (i.e., $nextClaim_{[H]}$). *Expected income from Contract* represents the reward of a miner obtained from $\mathcal{FRSC}$ (i.e., $nextClaim_{[H]}$).

Experiment II

Methodology. In this experiment, we investigated how multiple $\mathcal{FRSC}$s dealt with the same scenario as before – i.e., varying $\mathbb{C}$. In detail, we investigated how individual $\mathcal{FRSC}$s contributed to the $nextClaim_{[H+1]}$ by their individual $\partial Claim_{[H+1]}^{\mathcal{FRSC}_{[H]}}$. This time, we varied only the parameter $\mathbb{C} \in \{0.5,\ 0.7,\ 0.9\}$, and we considered four $\mathcal{FRSC}$s:

$$\mathcal{FRSC}s = \{$$
$$\mathcal{FRSC}^1(\_, 1008, 0.07), \mathcal{FRSC}^2(\_, 2016, 0.14),$$



(a) Scenario with 4 $\mathcal{FRSC}$s, $\mathbb{C} = 0.5$.

(b) $\partial Claims$ and $nextClaim$, $\mathbb{C} = 0.5$.

(c) $\partial Claims$ normalized by ρ, $\mathbb{C} = 0.5$.

(d) Scenario with 4 $\mathcal{FRSC}$s, $\mathbb{C} = 0.7$.

(e) $\partial Claims$ and $nextClaim$, $\mathbb{C} = 0.7$.

(f) $\partial Claims$ normalized by ρ, $\mathbb{C} = 0.7$.

(g) Scenario with 4 $\mathcal{FRSC}$s, $\mathbb{C} = 0.9$.

(h) $\partial Claims$ and $nextClaim$, $\mathbb{C} = 0.9$.

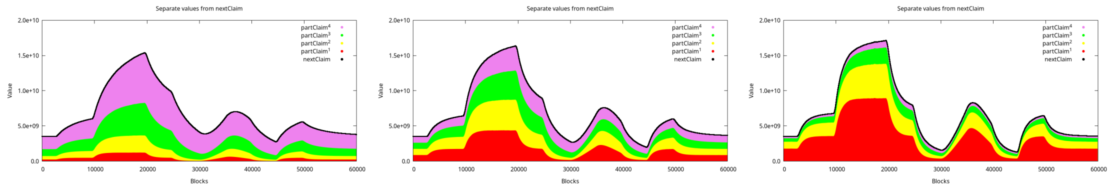
(i) $\partial Claims$ normalized by ρ, $\mathbb{C} = 0.9$.

Figure 4.10: Experiment II investigating various $\mathbb{C}$s in the setting with multiple $\mathcal{FRSC}$s with their corresponding $\lambda = \{1008, 2016, 4032, 8064\}$ and $\rho = \{0.07, 0.14, 0.28, 0.51\}$. $\partial Claims$ represents contributions of individual $\mathcal{FRSC}$s to the total reward of the miner (i.e., its $nextClaim$ component).

$$\mathcal{FRSC}^3(\_, 4032, 0.28), \mathcal{FRSC}^4(\_, 8064, 0.51)\},$$

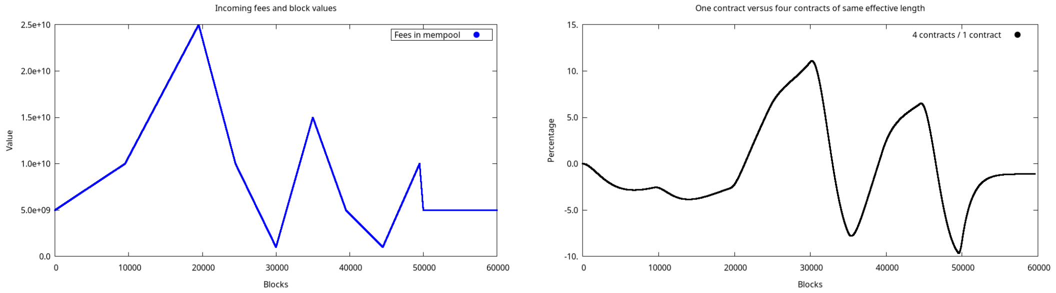
where their lengths λ were set to consecutive multiples of 2 (to see differences in more intensive averaging across longer intervals), and their redistribution ratios ρ were set to maximize the potential of averaging by longer $\mathcal{FRSC}$s.

Results. The results of this experiment are depicted in Figure 4.10. We can observe that the shorter $\mathcal{FRSC}$s quickly adapted to new changes and the longer $\mathcal{FRSC}$s kept more steady income for the miner. In this sense, we can see that $\partial Claim^4$ held steadily over the scenario while for example $\partial Claim^1$ fluctuated more significantly. Since the scenarios of fees evolution in the mempool was the same across all our experiments (but Section 4.3.3), we can compare the $\mathcal{FRSC}$ with λ = 5600 from Section 4.3.3 and the current setup involving four $\mathcal{FRSC}$s – both had some similarities. This gave us intuition for replacing multiple $\mathcal{FRSC}$s with a single one (see Section 4.3.3).



(a) ρ correlates with λ. (b) ρ equal for every $\mathcal{FRSC}$. (c) ρ negatively correlates with λ.

Figure 4.11: Experiment II – multiple $\mathcal{FRSC}$s using various distributions of ρ and their impact on $\partial Claim$, where $\mathbb{C}$ = 0.7.



(a) A custom fee scenario for Experiment III. (b) A relative difference in $nextClaim$ between 4 $\mathcal{FRSC}$s and a single $\mathcal{FRSC}$.

Figure 4.12: Experiment III comparing 4 $\mathcal{FRSC}$s and 1 $\mathcal{FRSC}$, both configurations having the same effective_λ.

Different Fee Redistribution Ratios Across $\mathcal{FRSC}$s

In Figure 4.11 we investigated different values of ρ in the same set of four contracts and their impact on $\partial Claims$. The results show that the values of ρ should correlate with λ of multiple $\mathcal{FRSC}$s to maximize the potential of averaging by longer $\mathcal{FRSC}$s.

Experiment III

Methodology. In this experiment, we investigated whether it is possible to use a single $\mathcal{FRSC}$ setup to replace a multiple $\mathcal{FRSC}$s while preserving the same effect on the $nextClaim$. To quantify a difference between such cases, we introduced a new metric of $\mathcal{FRSC}s$, called effective_λ, which can be calculated as follows:

$$\text{effective\_}\lambda(\mathcal{FRSC}s) = \sum_{x \,\in\, \mathcal{FRSC}s} x.\rho * x.\lambda. \tag{4.17}$$

We were interested in comparing a single $\mathcal{FRSC}$ with 4 $\mathcal{FRSC}$s, both configurations having the equal effective_λ. The configurations of these two cases are as follows:

(1) $\mathcal{FRSC}(\_, 5292, 1)$ and
(2) $\mathcal{FRSC}s = \{$

$$\mathcal{FRSC}^1(\_, 1008, 0.07), \mathcal{FRSC}^2(\_, 2016, 0.19),$$
$$\mathcal{FRSC}^3(\_, 4032, 0.28), \mathcal{FRSC}^4(\_, 8064, 0.46)\}.$$

We can easily verify that the effective_λ of 4 $\mathcal{FRSC}$s is the same as in a single $\mathcal{FRSC}$ using Equation 4.17: $0.07 * 1008 + 0.19 * 2016 + 0.28 * 4032 + 0.46 * 8064 = 5292$.

We conducted this experiment using a custom fee evolution scenario involving mainly linearly increasing/decreasing fees in the mempool (see Figure 4.12a), and we set $\mathbb{C}$ to 0.7 for both configurations. The custom scenario of the fee evolution in mempool in this experiment was chosen to contain extreme changes in fees, emphasizing possible differences in two investigated setups.

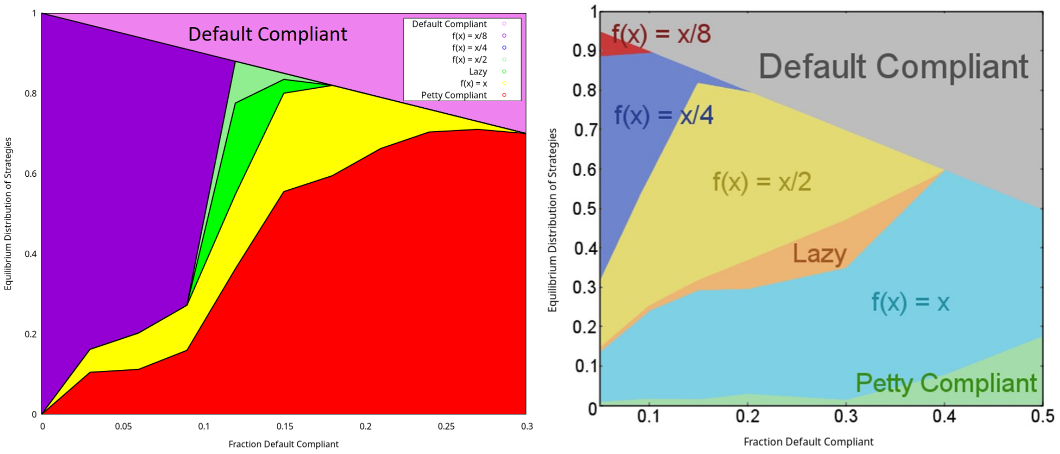
Results. In Figure 4.12b, we show the relative difference in percentages of $nextClaim$ rewards between the settings of 4 $\mathcal{FRSC}$s versus 1 $\mathcal{FRSC}$. It is clear that the setting of 4 $\mathcal{FRSC}$s in contrast to a single $\mathcal{FRSC}$ provided better reward compensation in times of very low fees value in the mempool, while it provided smaller reward in the times of higher values of fees in the mempool. Therefore, we concluded that it is not possible to replace a setup of multiple $\mathcal{FRSC}$s with a single one while retaining the same fee redistribution behavior.

Experiment IV

We focused on reproducing the experiment from Section 5.5 of [CKWN16a], while utilizing our approach. The experiment is aimed on searching for the minimal ratio of DEFAULT-COMPLIANT miners, at which the undercutting attack is no longer profitable strategy. DEFAULT-COMPLIANT miners are honest miners who follow the rules of the consensus protocol such as building on top of the longest chain. We executed several simulations, each consisting of multiple games (i.e., 300k as in [CKWN16a]) with various fractions of DEFAULT-COMPLIANT miners. From the remaining miners we evenly created *learning miners*, who learn on the previous runs of games and switch with a certain probability the best strategy out of the following:

- PETTYCOMPLIANT: This miner behaves as DEFAULT-COMPLIANT except one difference. In the case of seeing two chains, he does not mine on the oldest block but rather the most profitable block. Thus, this miner is not the (directly) attacking miner.
- LAZYFORK: This miner checks which out of two options is more profitable: (1) mining on the longest-chain block or (2) undercutting that block. In either way, he leaves half of the mempool fees for the next miners, which prevents another LAZYFORK miner to undercut him.
- FUNCTION-FORK() The behavior of the miner can be parametrized with a function f(.) expressing the level of his undercutting. The higher the output number the less reward he receives and more he leaves to incentivize other miners to mine on top of his block. This miner undercuts every time he forks the chain.

Methodology. With the missing feature for difficulty re-adjustment (in the simulator from [CKWN16a] that we extended) the higher orphan rate occurs, which might directly



(a) Simulations of our approach.

(b) Simulations of the original work [CKWN16a].

Figure 4.13: Experiment IV – The ratio of DEFAULT-COMPLIANT miners in our approach is $\sim$30% (in contrast to $\sim$ 66% of [CKWN16a]).

impact our $\mathcal{FRSC}$-based approach. If the orphan rate is around 40%, roughly corresponding to [CKWN16a], our blocks would take on average 40% longer time to be created, increasing the block creation time (i.e., time to mine a block). This does not affect the original simulator, as there are no $\mathcal{FRSC}s$ that would change the total reward for the miner who found the block.

Nevertheless, this is not true for $\mathcal{FRSC}$-based simulations as the initial setup of $\mathcal{FRSC}s$ is calculated with $\overline{fees} = 50$ BTC (as per the original simulations). However, with longer block creation time and transaction fees being calculated from it, the amount of $\overline{fees}$ also changes. With no adjustments, this results in $\mathcal{FRSC}s$ initially paying smaller reward back to the miner before $\mathcal{FRSC}s$ are saturated. To mitigate this problem, we increased the initial values of individual $\mathcal{FRSC}$s by the orphan rate from the previous game before each run. This results in very similar conditions, which can be verified by comparing the final value in the longest chain of our simulation versus the original simulations. We decided to use this approach to be as close as possible to the original experiment. This is particularly important when the full mempool parameter is equal to $false$, which means that the incoming transaction fees to mempool are calculated based on the block creation time. In our simulations, we used the following parameters: 100 miners, 10 000 blocks per game, 300 000 games (in each simulation run), exp3 learning model, and $\mathbb{C} = 0.7$. Modeling of fees utilized the same parameters as in the original paper [CKWN16a]: the full mempool parameter disabled, a constant inflow of 5 000 000 000 Satoshi (i.e., 50 BTC) every 600s. For more details about the learning strategies and other parameters, we refer the reader to [CKWN16a].

Setup of $\mathcal{FRSC}s$. Since we have a steady inflow of fees to the mempool, we do not need to average the income for the miner. Therefore, we used only a single $\mathcal{FRSC}$ defined as $\mathcal{FRSC}$(7 056 000 000 000, 2016, 1), where the initial value of $\mathcal{FRSC}.\nu$ was adjusted according to Equation 4.16, assuming $\overline{fees} = 50$ BTC. In the subsequent runs of each game, $\mathcal{FRSC}.\nu$ was increased by the orphan rate from the previous runs.

Results. The results of this experiment, depicted in Figure 4.13, demonstrate, that with our approach using $\mathcal{FRSC}$s, we decreased the number of Default-Compliant miners from the original 66% to 30%. This means that the profitability of undercutting miners is avoided with at least 30% of Default-Compliant miners, indicating more robust results.

For other details, we refer the reader to our paper [BHS23].

4.4 Contributing Papers

The papers that contributed to this research direction are enumerated in the following, while highlighted papers are attached to this thesis in their original form.

[SRHS19] **Pawel Szalachowski, Daniël Reijsbergen, Ivan Homoliak, and Siwei Sun. Strongchain: Transparent and collaborative proof-of-work consensus. In *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019.*, pages 819–836, 2019.**

[PBH+23] **Martin Perešíni, Federico Matteo Benčić, Martin Hrubý, Kamil Malinka, and Ivan Homoliak. Incentive attacks on DAG-based blockchains with random transaction selection. In *IEEE International Conference on Blockchain, Blockchain 2023, Hainan, China, December 17-21, 2023*. IEEE.**

[BHS23] **Rastislav Budinský, and Ivana Stančíková and Ivan Homoliak. Fee-redistribution smart contracts for transaction-fee-based regime of blockchains with the longest chain rule. In *IEEE International Conference on Blockchain, Blockchain 2023, Hainan, China, December 17-21, 2023*. IEEE, 2023.**

[PHMH24] Martin Perešíni, Tomáš Hladký, Kamil Malinka, and Ivan Homoliak. Dag-Sword: A simulator of large-scale network topologies for dag-oriented proof-of-work blockchains. In *Hawaii International Conference on System Sciences (HICSS), Hawaii, USA, January 3-6, 2024.* IEEE, 2024.

Chapter 5

Cryptocurrency Wallets

In this chapter, we present our contributions to the area of authentication for blockchain and decentralized applications, which belong to the application layer of our security reference architecture (see Chapter 3). In particular, this chapter is focused on cryptocurrency wallets and their subcategory of smart contract wallets, and it is based on the papers [HBH+20a, HBH$^+$20b] (see also Section 5.4).

First, we review existing cryptocurrency wallet solutions (with their security issues) and propose a classification scheme based on authentication factors validated against the blockchain or a centralized party [HBH$^+$20b]. We apply the proposed classification to the existing wallet solutions and also cross-compare other security features of them. Next, we propose SmartOTPs [HBH+20a], a 2FA authentication scheme against the blockchain, which, on top of using a hardware wallet, introduces the authenticator App (or a device) generating OTPs that are transferred in an air-gapped fashion to the client.

Notation

We denote the user by $\mathbb{U}$, the client (e.g., the user agent/browser) by $\mathbb{C}$, a wallet holding a private key by $\mathbb{W}$, the authenticator device or App as $\mathbb{A}$, and an adversary by $\mathcal{A}$.

5.1 Security Issues in Authentication Schemes of Wallets

According to works [ECBS18, BMC$^+$15], there are a few categories of key management approaches. In password-protected wallets, private keys are encrypted with selected passwords. Unfortunately, users often choose weak passwords that can be brute-forced if stolen by malware [Del15]; optionally, such malware may use a keylogger for capturing a passphrase [BMC$^+$15, Pey17]. Another similar option is to use password-derived wallets that generate keys based on the provided password. However, they also suffer from the possibility of weak passwords [CSC16]. Hardware wallets are a category that promises the provision of better security by introducing devices that enable only the signing of transactions, without revealing the private keys stored on the device. However, these wallets do not provide protection from an attacker with full access to the device [Kra20, Kra19, Don19], and more importantly, wallets that do not have a secure channel for informing the user about the details of a transaction being signed (e.g., [Led18a]) may be exploited by malware targeting IPC mechanisms [BRA$^+$18].

A popular option for storing private keys is to deposit them into server-side hosted (i.e., custodial) wallets and currency-exchange services [Coi18a, Bin20, Pol20, Pay20, Lun18, Pax20]. In contrast to the previous categories, server-side wallets imply trust in a provider, which is a potential risk of this category. Due to many cases of compromising server-side wallets [Wol18, Rac14, Reu16, MC13, Bin19] or fraudulent currency-exchange operators [VM15], client-side hosted wallets have started to proliferate. In such wallets, the main functionality, including the storage of private keys, has moved to the user side [Myc18, Car18, Cit18, Coi18b, In18]; hence, trust in the provider is reduced but the users still depend on the provider's infrastructure.

To increase security of former wallet categories, multi-factor authentication (MFA) is often used, which enables spending crypto-tokens only when a number of secrets are used together. Wallets from a split control category [ECBS18] provide MFA against the blockchain. This can be achieved by threshold cryptography wallets [GGK$^+$15, Myc19], multi-signature wallets [Arm16, Ele18, Tru19, Cop19], and state-aware smart-contract wallets [Unc18, Tec18, Con19a]. Nevertheless, these schemes might impose additional usability implications, performance overhead, or cost of wallet devices.

5.2 Classification of Authentication Schemes

We introduce the notion of k-factor authentication against the blockchain and k-factor authentication against the authentication factors. Using these notions, we propose a classification of authentication schemes, and we apply it to examples of existing key management solutions (see Section 5.2.1 and Section 5.2.2).

In the context of the blockchain, we distinguish between k-factor authentication *against the blockchain* and k-factor authentication *against the authentication factors* themselves. For example, an authentication method may require the user to perform 2-of-2 multi-signature in order to execute a transfer, while $\mathbb{U}$ may keep each private key stored in a dedicated device – each requiring a different password. In this case, 2FA is performed against the blockchain, since both signatures are verified by all miners of the blockchain. Additionally, a one-factor authentication is performed once in each device of $\mathbb{U}$ by entering a password in each of them. For clarity, we classify authentication schemes by the following notation:

$$\left(Z + X_1/\ldots/X_Z\right),$$

where $Z \in \{0, 1, \ldots\}$ represents the number of authentication factors against the blockchain and $X_i \in \{0, 1, \ldots\} \mid i \in [1, \ldots, Z]$ represents the number of authentication factors against the i-th factor of Z. With this in mind, we remark that the previous example provides $(2 + 1/1)$-factor authentication: twice against the blockchain (i.e., two signatures), once for accessing the first device (i.e., the first password), and once for accessing the second device (i.e., the second password).

Since the previous notation is insufficient for authentication schemes that use secret sharing [Sha79], we extend it as follows:

$$\left(Z^{(W_1,\ldots,W_Z)} + \left(X_1^1,\ldots,X_1^{W_1}\right) / \ldots / \left(X_Z^1,\ldots,X_Z^{W_Z}\right)\right),$$

where Z has the same meaning as in the previous case, $W_i \in \{0,1,\ldots\} \mid i \in [1,\ldots,Z]$ denotes the minimum number of secret shares required to use the complete i-th secret X_i. With this in mind, we remark that the aforementioned example provides $\left(2^{(1,1)} + (1)/(1)\right)$-factor authentication: twice against the blockchain (i.e., two signatures), once for accessing the first device (i.e., the first password), and once for accessing the second device (i.e., the second password). We consider an implicit value of $W_i = 1$; hence, the classification $(2+1/1)$ represents the same as the previous one (the first notation suffices). If one of the private keys were additionally split into two shares, each encrypted by a password, then such an approach would provide $\left(2^{(2,1)} + (1,1)/(1)\right)$-factor authentication.

5.2.1 Review of Wallet Types Using the Classification

We extend the previous work of Eskandari et al. [ECBS18] and Bonneau et al. [BMC$^+$15], by categorizing and reviewing a few examples of key management solutions, while assuming our classification.

Keys in Local Storage. In this category of wallets, the private keys are stored in plaintext form on the local storage of a machine, thus providing $(1 + 0)$-factor authentication. Examples that enable the use of unencrypted private key files are Bitcoin Core [Bit18a] or MyEtherWallet [MyE18] wallets.

Password-Protected Wallets. These wallets require the user-specified password to encrypt a private key stored on the local storage, thus providing $(1 + 1)$-factor authentication. Examples that support this functionality are Armory Secure Wallet [Arm16], Electrum Wallet [Ele18], MyEtherWallet [MyE18], Bitcoin Core [Bit18a], and Bitcoin Wallet [Bit18b]. This category addresses physical theft, yet enables the brute force of passwords and digital theft (e.g., keylogger).

Password-Derived Wallets. Password-derived wallets [Max11] (a.k.a., brain wallets or hierarchical deterministic wallets) can compute a sequence of private keys from only a single mnemonic string and/or password. This approach takes advantage of the key creation in the ECDSA signature scheme that is used by many blockchain platforms. Examples of password-derived wallets are Electrum [Ele18], Armory Secure Wallet [Arm16], Metamask [Met19], and Daedalus Wallet [Dae18]. The wallets in this category provide $(1 + X_1)$-factor authentication (usually $X_1 = 1$) and also suffer from weak passwords [CSC16].

Hardware Storage Wallets. In general, wallets of this category include devices that can only sign transactions by private keys stored inside sealed storage, while the keys never leave the device. To sign a transaction, $\mathbb{U}$ connects the device to a machine and enters his passphrase. When signing a transaction, the device displays the transaction's data to $\mathbb{U}$, who may verify the details. Thus, wallets of this category usually provide $(1 + 1)$-factor authentication. Popular USB (or Bluetooth) hardware wallets containing displays are offered by Trezor [Tre18], Ledger [Led18b], KeepKey [Kee18], and BitLox [Bit18d]. An example of a USB wallet that is not resistant against tampering with $\mathbb{C}$ (e.g., keyloggers) is Ledger Nano [Led18a] – it does not have a display, hence $\mathbb{U}$ cannot verify the details of transactions being signed. An air-gapped transfer of transactions using QR codes is provided by ELLIPAL wallet [ELL19]. In ELLIPAL, both $\mathbb{C}$ (e.g., smartphone App) and the hardware wallet must be equipped with cameras and display. $(1 + 0)$-factor authentication is provided by a credit-card-shaped hardware wallet from CoolBitX [Coo18]. A hybrid approach that relies on a server providing a relay for 2FA is offered by BitBox [SHI18]. Although a BitBox device does not have a display, after connecting to a machine, it communicates with $\mathbb{C}$ running on the machine and at the same time, it communicates with a smartphone App through BitBox's server; each requested transaction is displayed and confirmed by $\mathbb{U}$ on the smartphone. One limitation of this solution is the lack of self-sovereignty.

Split Control – Threshold Cryptography. In threshold cryptography [Sha79, MR01, GJKR07, B^{+}79], a key is split into several parties which enables the spending of crypto-tokens only when n-of-m parties collaborate. Threshold cryptography wallet provide $\big(1^{(W_1,\ldots,W_n)} + (X_1, \ldots, X_n)\big)$-factor authentication, as only a single signature verification is made on a blockchain, but n verifications are made by parties that compute a signature. Therefore, all the computations for co-signing a transaction are performed off-chain, which provides anonymity of access control policies (i.e., a transaction has a single signature) in contrast to the multi-signature scheme that is publicly visible on the blockchain. An example of this category is presented by Goldfeder et al. [GGK^{+}15]. One limitation of this solution is a computational overhead that is directly proportional to the number of involved parties m (e.g., for $m = 2$ it takes 13.26s). Another example of this category is a USB dongle called Mycelium Entropy [Myc19], which, when connected to a printer, generates triplets of paper wallets using 2-of-3 Shamir's secret sharing; providing $(1^{(2)} + (0,\ 0))$-factor authentication.

Split Control – Multi-Signature Wallets. In the case of multi-signature wallets, n-of-m owners of the wallet must co-sign the transaction made from the multi-owned address. Thus, the wallets of this category provide $(n + X_1/\ldots/X_n)$-factor authentication. One example of a multi-owned address approach is Bitcoin's Pay to Script Hash (P2SH).[1] Examples supporting multi-owned addresses are Lockboxes of Armory Secure Wallet [Arm16] and Electrum Wallet [Ele18]. A property of multi-owned address is that each

[1] We refer to the term *multi-owned address of P2SH* for clarity, although it can be viewed as Turing-incomplete smart contract.

transaction with such an address requires off-chain communication. A hybrid instance of this category and client-side hosted wallets category is Trusted Coin's cosigning service [Tru19], which provides a 2-of-3 multi-signature scheme – $\mathbb{U}$ owns a primary and a backup key, while TrustedCoin owns the third key. Each transaction is signed first by user's primary key and then, based on the correctness of the OTP from Google Authenticator, by TrustedCoin's key. Another hybrid instance of this category and client-side hosted wallets is Copay Wallet [Cop19]. With Copay, the user can create a multi-owned Copay wallet, where $\mathbb{U}$ has all keys in his machines and each transaction is co-signed by n-of-m keys. Transactions are resent across user's machines during multi-signing through Copay.

Split-Control – State-Aware Smart Contracts. State-aware smart contracts provide "rules" for how crypto-tokens of a contract can be spent by owners, while they keep the current setting of the rules on the blockchain. The most common example of state-aware smart contracts is the 2-of-3 multi-signature scheme that provides $(2 + X_1/X_2)$-factor authentication. An example of the 2-of-3 multi-signature approach that only supports Trezor hardware wallets is *TrezorMultisig2of3* from Unchained Capital [Unc18]. One disadvantage of this solution is that $\mathbb{U}$ has to own three Trezor devices, which may be an expensive solution that, moreover, relies only on a single vendor. Another example of this category, but using the n-of-m multi-signature scheme, is Parity Wallet [Tec18]. However, two critical bugs [Par17b, Par17a] have caused the multi-signature scheme to be currently disabled. The n-of-m multi-signature scheme is also used in *Gnosis Wallet* from ConsenSys [Con19a].

Hosted Wallets. Common features of hosted wallets are that they provide an online interface for interaction with the blockchain, managing crypto-tokens, and viewing transaction history, while they also store private keys at the server side. If a hosted wallet has full control over private keys, it is referred to as a *server-side wallet*. A server-side wallet acts like a bank – the trust is centralized. Due to several cases of compromising such server-side wallets [Wol18], [Rac14], [Reu16], [MC13], the hosted wallets that provide only an interface for interaction with the blockchain (or store only user-encrypted private keys) have started to proliferate. In such wallets, the functionality, including the storage of private keys, has moved to $\mathbb{U}$'s browser (i.e., client). We refer to these kinds of wallets as *client-side wallets* (a.k.a., hybrid wallets [ECBS18]).

Server-Side Wallets. Coinbase [Coi18a] is an example of a server-side hosted wallet, which also provides exchange services. Whenever the user logs in or performs an operation, he authenticates himself against Coinbase's server using a password and obtains a code from Google Authenticator/Authy app/SMS. Other examples of server-side wallets having equivalent security level to Coinbase are Circle Pay Wallet [Cir18] and Luno Wallet [Lun18]. The wallets in this category provide $(0 + 2)$-factor authentication when 2FA is enabled.

Client-Side Wallets. An example of a client-side hosted wallet is Blockchain Wallet [Blo18]. Blockchain Wallet is a password-derived wallet that provides 1-factor authentication against the server based on the knowledge of a password and additionally enables 2FA against the server through one of the options consisting of Google Authenticator, YubiKey, SMS, and email. When creating a transaction, $\mathbb{U}$ can be authenticated by entering his secondary password. Equivalent functionality and security level as in Blockchain Wallet are offered by BTC Wallet [BTC18]. In contrast to Blockchain Wallet, BTC wallet uses 2FA also during the confirmation of a transaction. Other examples of this category are password-derived wallets, like Mycelium Wallet [Myc18], CarbonWallet [Car18], Citowise Wallet [Cit18], Coinomi Wallet [Coi18b], and Infinito Wallet [In18], which, in contrast to the previous examples, do not store backups of encrypted keys at the server. A 2FA feature is provided additionally to password-based authentication, in the case of CarbonWallet. In detail, the 2-of-2 multi-signature scheme uses the machine's browser and the smartphone's browser (or the app) to co-sign transactions.

5.2.2 A Comparison of Security Features of Wallets

We present a comparison of wallets and approaches from Section 5.2.1 in Table 5.1. We apply our proposed classification on authentication schemes, while we also survey a few selected security and usability properties of the wallets from the work of Eskandri et al. [ECBS18]. In the following, we briefly describe each property and explain the criteria stating how we attributed the properties to particular wallets.

Air-Gapped Property. We attribute this property (Y) to approaches that involve at least one hardware device storing secret information, which do not need a connection to a machine in order to operate.

Resilience to Tampering with the Client. We attribute this property (Y) to all hardware wallets that sign transactions within a device, while they require $\mathbb{U}$ to confirm transaction's details at the device (based on displayed information). Then, we attribute this property to wallets containing multiple clients that collaborate in several steps to co-signs transactions (a chance that all of them are tampered with is low).

Post-Quantum Resilience. We attribute this property (Y) to approaches that utilize hash-based cryptography that is known to be resilient against quantum computing attacks [ADMG^{+}16].

No Need for Off-Chain Communication. We attribute this property (Y) to approaches that do not require an off-chain communication/transfer of transaction among parties/devices to build a final (co-)signed transaction, before submitting it to a blockchain (applicable only for $Z \geq 2$ or $W_i \geq 2$).

	Authentication Scheme		Air-Gapped Property	Resilience to Tampering with the Client	Post-Quantum Resilience	No Need for Off-Chain Communication	Malware Resistance	Secret(s) Kept Offline	Independence of Trusted Third Party	Resilience to Physical Theft	Resilience to Password Loss	Comments
	Classification	Details										
Keys in Local Storage	$1 + (0)$	Private key										
Bitcoin Core [Bit18a]	$1 + (0)$	For one of the options	N	N	N	Y	N	N	Y	N	N/A	
MyEtherWallet [MyE18]	$1 + (0)$	For one of the options	N	N	N	Y	N	N	Y	N	N/A	
Password-Protected Wallets	$1 + (1)$	Private key + encryption										
Armory Secure Wallet [Arm16]	$1 + (1)$		N	N	N	Y	N	N	Y	Y	N	
Electrum Wallet [Ele18]	$1 + (1)$		N	N	N	Y	N	N	Y	Y	N	
MyEtherWallet (Offline) [MyE18]	$1 + (1)$		N	N	N	Y	N	N	Y	Y	N	
Bitcoin Core [Bit18a]	$1 + (1)$		N	N	N	Y	N	N	Y	Y	N	
Bitcoin Wallet [Bit18b]	$1 + (1)$		N	N	N	Y	N	N	Y	Y	N	
Password-Derived Wallets	$1 + (X_1)$											
Armory Secure Wallet [Arm16]	$1 + (1)$		N	N	N	Y	N	N	Y	Y	Y	
Electrum Wallet [Ele18]	$1 + (1)$		N	N	N	Y	N	N	Y	Y	Y	
Metamask [Met19]	$1 + (1)$		N	N	N	Y	N	N	Y	Y	Y	
Daedalus Wallet [Dae18]	$1 + (2)$	2 passwords	N	N	N	Y	N	N	Y	Y	Y	
Hardware Storage Wallets	$1 + (X_1)$											
Trezor [Tre18]	$1 + (1)$		N	Y	N	Y	Y	Y	Y	Y	Y	
Ledger [Led18b]	$1 + (1)$		N	Y	N	Y	Y	Y	Y	Y	Y	
KeepKey [Kee18]	$1 + (1)$		N	Y	N	Y	Y	Y	Y	Y	Y	
BitLox [Bit18d]	$1 + (2)$	2 passwords*	N	Y	N	Y	Y	Y	Y	Y	Y	*Additionally, protection against the evil maid attack
CoolWallet S [Coo18]	$1 + (0)$		N	Y	N	Y	Y	Y	Y	P†	N/A	†Depending on the mode
Ledger Nano [Led18a]	$1 + (2)$	Password + GRID card	N	N	N	Y	N	Y	Y	Y	Y	
ELLIPAL wallet [ELL19]	$1 + (1)$		Y	Y	N	Y	Y	Y	Y	Y	Y	
BitBox USB Wallet [SHI18]	$1 + (2)$	1 password and App	N	Y	N	Y	Y	Y	P‡	Y	Y	‡Requires a relay server
Split Control – Threshold Cryptography	$1^{(W_1)} + (X_1^1, \ldots, X_1^{W_1})$											
Goldfeder et al. [GGK$^+$15]	$1^{(2)} + (1, 1)$	Assuming 2 devices, each protected by a password	N	Y	N	N	Y	N/A	N/A	N/A	N/A	
Mycelium Entropy [Myc19]	$1^{(2)} + (0, 0)$		N	Y	N	N	Y	Y	Y	Y	N/A	
Split Control – Multi-Signature Wallets	$Z + (X_1/\ldots/X_z)$											
Lockboxes of Armory Secure Wallet [Arm16]	$Z + (X_1/\ldots/X_z)$	Z up to 7, $X_i = 1$	N	Y	N	N	Y	N	Y	Y	N	
Electrum Wallet [Ele18]	$Z + (X_1/\ldots/X_z)$	Z up to 15, $X_i = 1$	N	Y	N	N	Y	N	Y	Y	Y	
Trusted Coin's cosigning service [Tru19]	$2 + (1/2)$	2 private keys + 2 passwords and Google Auth.	N	Y	N	N	Y	N	N	Y	Y	A hybrid client-side wallet
Copay Wallet [Cop19]	$2 + (1/1)$		N	Y	N	N	Y	N	P	Y	Y	A hybrid client-side wallet
Split-Control – State-Aware Smart Contracts	$Z + (X_1/\ldots/X_z)$											
TrezorMultisig2of3 [Unc18]	$2 + (1/1)$	Assuming that each device is protected by a password	N	Y	N	N	Y	Y	Y	Y	Y	
Parity Wallet [Tec18]	$Z + (X_1/\ldots/X_z)$	Z is unlimited, $X_i = 1$	N	Y	N	Y	Y	N	Y	Y	Y	
Gnosis Wallet [Con19a]	$Z + (X_1/\ldots/X_z)$	Z up to 50, $X_i = 1$	N	Y	N	Y	Y	N	Y	N/A	Y	
SmartOTPs [HBH+20a]	$2 + (1/1)$	Private key and OTPs + passwords	Y$^{\circ}$	Y$^{\$}$	Y	Y	Y	Y	Y	Y	Y$^{\#}$	$^{\circ}$Fully air-gapped, if combined with ELLIPAL wallet $^{\$}$Thanks to a hardware wallet $^{\#}$Also resilient to loss of all secrets
Server-Side Wallets	$0 + (X_1)$											
Coinbase [Coi18a]	$0 + (2)$	Password, Google Auth./SMS	N	N	N	Y	N	N	N	Y	Y	
Circle Pay [Cir18]	$0 + (2)$	—"—	N	N	N	Y	N	N	N	Y	Y	
Luno Wallet [Lun18]	$0 + (2)$	Password and Google Auth.	N	N	N	Y	N	N	N	Y	Y	
Client-Side Wallets	$Z + (X_1)$											
Blockchain Wallet [Blo18]	$1 + (2)$	Password and one of: Google Auth., YubiKey, SMS, or email	N	N	N	Y	N	N	N	Y	Y	
BTC Wallet [BTC18]	$1 + (2)$	—"—	N	N	N	Y	N	N	N	Y	Y	
Mycelium Wallet [Myc18]	$1 + (1)$		N	N	N	Y	N	N	N	Y	Y	
CarbonWallet [Car18]	$2 + (2)$	2 private keys stored in browser and smartphone	N	Y	N	N	N	N	N	Y	Y	
Citowise Wallet [Cit18]	$1 + (2)$		N	Y¶	N	Y	N	P¶	N	Y	Y	¶If combined with Trezor or Ledger
Coinomi Wallet [Coi18b]	$1 + (1)$		N	N	N	Y	N	N	N	Y	Y	
Infinito Wallet [In18]	$1 + (1)$		N	N	N	Y	N	N	N	Y	Y	

Table 5.1: Comparison of state-of-the-art cryptocurrency wallets using our classification (see Section 5.2) and other security features.

Malware Resistance (e.g., Key-Loggers). We attribute this property (Y) to approaches that either enable signing transactions inside of a sealed device or split signing control over secrets across multiple devices.

Secret(s) Kept Offline. We attribute this property (Y) to approaches that keep secrets inside their sealed storage, while they expose only signing functionality. Next, we attribute this property to paper wallets and fully air-gapped devices.

Independence of Trusted Third Party We attribute this property (Y) to approaches that do not require trusted party for operation, while we do not attribute this property to all client-side and server-side hosted wallets. We partially (P) attribute this property to approaches requiring an external relay server for their operation.

Resilience to Physical Theft. We attribute this property (Y) to approaches that are protected by an encryption password or PIN. We partially (P) attribute this property to approaches that do not provide password and PIN protection but have a specific feature to enforce uniqueness of an environment in which they are used (e.g., bluetooth pairing).

Resilience to Password Loss. We attribute this property (Y) to approaches that provide means for recovery of secrets (e.g., a seed of hierarchical deterministic wallets).

5.3 SmartOTPs

In this section, we propose SmartOTPs, a smart-contract wallet framework that gives a flexible, usable, and secure way of managing crypto-tokens in a self-sovereign fashion. The proposed framework consists of four components (i.e., an authenticator, a client, a hardware wallet, and a smart contract), and it provides 2-factor authentication (2FA) performed in two stages of interaction with the blockchain. To the best of our knowledge, our framework is the first one that utilizes one-time passwords (OTPs) in the setting of the public blockchain. In SmartOTPs, the OTPs are aggregated by a Merkle tree and hash chains whereby for each authentication only a short OTP (e.g., 16B-long) is transferred from the authenticator to the client. Such a novel setting enables us to make a fully air-gapped authenticator by utilizing small QR codes or a few mnemonic words, while additionally offering resilience against quantum cryptanalysis. We have made a proof-of-concept based on the Ethereum platform. Our cost analysis shows that the average cost of a transfer operation is comparable to existing 2FA solutions using smart contracts with multi-signatures.

Notation

By the term *operation* we refer to an action with a smart-contract wallet using SmartOTPs, which may involve, for instance, a transfer of crypto-tokens or a change of daily spending limits. Then, we use the term *transfer* for the indication of transferring crypto-tokens. By $\{msg\}_{\mathbb{U}}$ we denote the message msg digitally signed by $\mathbb{U}$, and by $msg.\sigma$ we refer to the signature; $\mathcal{RO}$ is the random oracle; $h(.)$: stands for a cryptographic hash function; $h^i(.)$ substitutes i-times chained function $h(.)$, e.g., $h^2(.) \equiv h(h(.))$; $||$ is the string concatenation; $h^i_{\mathcal{D}}(.)$ substitutes i-times chained function $h(.)$ with embedded domain separation, e.g., $h^2_{\mathcal{D}}(.) = h(2 \mid\mid h(1 \mid\mid .))$; $F_k(.) \equiv h(k \parallel .)$ denotes a pseudo-random function that is parametrized by a secret seed k; $\%$ represents modulo operation over integers; $\Sigma.\{KeyGen, Verify, Sign\}$ represents a signature scheme of the blockchain platform; $SK_{\mathbb{U}}$, $PK_{\mathbb{U}}$ is the private/public key-pair of $\mathbb{U}$, under Σ, and $a \mid b$ represents bitwise OR of arguments a and b.

5.3.1 Problem Definition

The main goal of this research is to propose a cryptocurrency wallet framework that provides a secure and usable way of managing crypto-tokens. In particular, we aim to achieve:

Self-Sovereignty: ensures that the user does not depend on the 3rd party's infrastructure, and the user does not share his secrets with anybody. Self-sovereign (i.e., non-custodial) wallets do not pose a single point of failure in contrast to server-side (i.e., custodial) wallets, which when compromised, resulted in huge financial loses [Wol18, Rac14, Reu16, MC13, Bin19].

Security: the insufficient security level of some self-sovereign wallets has caused significant financial losses for individuals and companies [Bun16, CSC16, CHH$^+$18, Par17b]. We argue that wallets should be designed with security in mind and in particular, we point out 2FA solutions, which have successfully contributed to the security of other environments [AZEH09, Sch05]. Our motivation is to provide a cheap security extension of the hardware wallets (i.e., the first factor) by using OTPs as the second factor in a fashion similar to Google Authenticator.

5.3.2 Threat Model

For a generic cryptocurrency, we assume an adversary $\mathcal{A}$ whose goal is to conduct unauthorized operations on the user's behalf or render the user's wallet unusable. $\mathcal{A}$ is able to eavesdrop on the network traffic as well as to participate in the underlying consensus protocol. However, $\mathcal{A}$ is unable to take over the cryptocurrency platform nor to break the used cryptographic primitives. We further assume that $\mathcal{A}$ is able to intercept and "override" the user's transactions, e.g., by launching a man-in-the-middle (MITM) attack or by creating a conflicting malicious transaction with a higher fee, which will incentivize miners to include $\mathcal{A}$'s transaction and discard the user's one; this attack is also referred to as *transaction front-running*. We assume three types of exclusively occurring attackers, each targeting one of the three components of our framework: (1) $\mathcal{A}$ with access to the user's private key hardware wallet $\mathbb{W}$, (2) $\mathcal{A}$ that tampers with the client $\mathbb{C}$, and for completeness we also assume (3) $\mathcal{A}$ with access to the authenticator $\mathbb{A}$. Next, we assume that the legitimate user correctly executes the proposed protocols and $h(.)$ is an instantiation of random oracle $\mathcal{RO}$.

5.3.3 Design Space

There are many types of wallets with different properties (see Section 5.2.1). In our context, to achieve self-sovereignty we identify smart-contract wallets as a promising category. These wallets manage crypto-tokens by the functionality of smart contracts, enabling users to have customized control over their wallets. The advantages of these solutions are that spending rules can be explicitly specified and then enforced by the cryptocurrency platform itself. Therefore, using this approach, it is possible to build a flexible wallet with features such as daily spending limits or transfer limits.

General OTPs

With spending rules encoded in a smart contract, it is feasible to design custom security features, such as OTP-based authentication serving as the second factor. In such a setting, the authenticator produces OTPs to authenticate transactions in the smart contract. However, in contrast to digital signatures, OTPs do not provide non-repudiation of data present in a transaction with an OTP; moreover, they can be intercepted and misused by the front-running or the MITM attacks. To overcome this limitation, we argue that a two-stage protocol $\Pi_O^{<G>}$ must be employed, enabling secure utilization of general OTPs in the context of blockchains. In the first stage of $\Pi_O^{<G>}$, an operation O, signed by the user $\mathbb{U}$, is submitted to the blockchain, where it obtains an identifier i. Then, in the second stage, O_i is executed on the blockchain upon the submission of OTP_i that is unambiguously associated with the operation initiated in the first stage.

Requirements of General and Air-Gapped OTPs

Based on the above, we define the necessary security requirements of general OTPs used in the blockchain as follows:

1. **Authenticity:** each OTP must be associated only with a unique authenticator instance.
2. **Linkage:** each OTP_i must be linked with exactly a single operation O_i, ensuring that OTP_i cannot be misused for the authentication of $O_j, i \neq j$.
3. **Independence:** OTP_i linked with the operation O_i cannot be derived from OTP_j of an operation O_j, where $i \neq j$, or an arbitrary set of other OTPs.

Nevertheless, in the air-gapped setting (important for a high usability and security), one more requirement comes into play: **the short length of OTPs**. Short OTPs allow the users to use a relatively small number of mnemonic words or a small QR code to transfer an OTP in an air-gapped fashion. This requirement is of high importance especially in the case when the authenticator is implemented as a resource-constrained embedded device with a small display (e.g., credit-card-shaped wallet, such as CoolBitX [Coo18]).

Analysis of Existing Solutions

We argue that not all solutions meet the requirements of air-gapped OTPs. Asymmetric cryptography primitives such as digital signatures or zero-knowledge proofs are inadequate in this setting, despite meeting all general OTP requirements. State-of-the-art signature schemes with reasonable performance overhead [BDL+12, JMV01] and short signature size produce a 48B-64B long output. The BLS signatures [BLS01] go even beyond the previous constructs and might produce signatures of size 32B. Nevertheless, BLS signatures are unattractive in the setting of the smart contract platforms that put high execution costs for BLS signature verification, which is $\sim$33 times more expensive than in the case of ECDSA with the equivalent security level [BWG+19]. Hence, we assume 48B as the minimal feasible OTP size for assymetric cryptography.

However, transferring even 48B in a fully air-gapped environment by transcription of mnemonic words [PRVB13] would lack usability for regular users – considering study from Dhakal et al. [DFKO18], transcription of 36 English words takes 42s on average, which is much longer than users are willing to "sacrifice." We note that the situation is better with QR code, but on the other hand it has two limitations: (1) when the authenticator is implemented as a simple embedded device, its display might be unable to fit a requested QR code with sufficient scanning properties (to preserve the maximal scanning distance of QR code, the "denser" QR code must be displayed in a larger image [QRS11]) and (2) occasionally, the users might not have a camera in their devices, thus, they can proceed only with a fallback method that uses mnemonics. Finally, most of the currently deployed asymmetric constructions are vulnerable to quantum computing [Ber09].

The problem of long signatures also exists in hash-based signature constructs [Lam79, DSS05, Mer89]. Lamport-Diffie one-time signatures (LD-OTS) [Lam79] produce an output of length $2|h(.)|^2$, which, for example in the case of $|h(.)| = 16B$ yields $4kB$-long signatures. The signature size of LD-OTS can be reduced by using one string of one-time key for simultaneous signing of several bits in the message digest (i.e., Winternitz one-time signatures (W-OTS) [DSS05]), but at the expense of exponentially increased number of hash computations (in the number of encoded bits) during a signature generation and verification. The extreme case minimizing the size of W-OTS to $|h(.)|$ (for simplicity omitting checksum) would require $2^{|h(.)|}$ hash computations for signature generation, which is unfeasible.

Approaches based on symmetric cryptography primitives produce much shorter outputs, but it is challenging to implement them with smart-contract wallets. Widely used one-time passwords like HOTP [MBH+05] or TOTP [MMPR11] require the user to share a secret key k with the authentication server. Then, with each authentication request the user proves that he possesses k by returning the output of an $F_k(.)$ computed with a nonce (i.e., HOTP) or the current timestamp (i.e., TOTP). This approach is insecure in the setting of the blockchain, as the user would have to share the secret k with a smart-contract wallet, making k publicly visible.

A solution that does not publicly disclose secret information and, at the same time, provides short enough OTPs (e.g., $16B \simeq 12$ mnemonic words $\simeq$ QR code v1), can be implemented by Lamport's hash chains [Lam81] or other single hash-chain-based constructs, such as T/Key [KMB17]. A hash chain enables the production of many OTPs by the consecutive execution of a hash function, starting from k that represents a secret key of the authenticator. Upon the initialization, a smart contract is preloaded with the last generated value $h^n(k)$. When the user wants to authenticate the ith operation, he sends the $h^{n-i}(k)$ to the smart contract in the second stage of $\Pi_O^{<G>}$. The smart contract then computes $h(.)$ consecutively i times and checks to ascertain whether the obtained value equals the stored value. However, the main drawback of this solution is that *each OTP can be trivially derived from any previous one*, and thereby this scheme does not meet the requirement of OTPs on independence. To detail an attack misusing this flaw, assume the MITM attacker possessing $SK_{\mathbb{U}}$ (i.e., the first factor) is able to initiate operations in the first stage of $\Pi_O^{<G>}$. The attacker $\mathcal{A}$ initiates operation O_i and waits for $\mathbb{U}$ to initiate and confirm an arbitrary follow-up operation $O_j, j > i$. When $\mathbb{U}$ sends OTP_j in the second stage of $\Pi_O^{<G>}$, $\mathcal{A}$ intercepts and "front-runs" the user's transaction by a malicious transaction with OTP_i computed as $h^{j-i}(OTP_j)$. Although one may argue that this scheme can be hardened by a modification denying to confirm older operations than the last initiated one, it would bring a race condition issue in which $\mathcal{A}$ might keep initiating operations in the first stage of $\Pi_O^{<G>}$ each time he intercepts a confirmation transaction from $\mathbb{U}$, causing the DoS attack on the wallet.

5.3.4 Proposed Approach

For a generic cryptocurrency with Turing-complete smart contract platform, we propose SmartOTPs, a 2FA against the blockchain, which consists of: (1) a client $\mathbb{C}$, (2) a private key hardware wallet $\mathbb{W}$ equipped with a display, (3) a smart-contract $\mathbb{S}$, and (4) an air-gapped authenticator $\mathbb{A}$ that might be implemented as an embedded device with limited resources. First, we explain the key idea of our approach, which enables us to construct $\mathbb{A}$ as a fully air-gapped device. Then, we present the base version of SmartOTPs, and finally, we describe modifications.

Design of an Air-Gapped Authenticator

In our approach, OTPs are generated by a pseudo-random function $F_k(.)$ and then aggregated by a Merkle tree, providing a single value, the root hash ($\mathcal{R}$). $\mathcal{R}$ is stored at $\mathbb{S}$ and serves as a PK for OTPs. Assuming the two stage protocol $\Pi_O^{<G>}$ (further denoted as Π_O), the user $\mathbb{U}$ might confirm the initiated operation O_{opID} by a corresponding OTP_{opID} (provided by $\mathbb{A}$) in the second stage of Π_O, whereby $\mathbb{S}$ verifies the correctness of OTP_{opID} with use of $\mathcal{R}$. A challenge of such an approach is the size of an OTP.

From Straw-Man to the Base Version. Using the straw-man version, a 2FA requires $\mathbb{A}$ to provide an OTP and its proof. However, in such a straw-man version, the user $\mathbb{U}$ has to transfer $\frac{(S+S\times H)}{8}$ bytes from $\mathbb{A}$ each time he confirms an operation, where S represents the bit-length of an OTP as well as the output of $h(.)$, and H represents the height of a Merkle tree with N leaves; hence $H = log_2(N)$. For example, if $S = 256$ and $H = 10$, then $\mathbb{U}$ would have to transfer 352B each time he confirms an operation, which has very low usability in an air-gapped setting utilizing transcription of mnemonic words [PRVB13] (i.e., 264 words) or scanning of several QR codes (e.g., 21 QR codes v1) displayed on an embedded device with a small display. Even further reduction of S to 128 bits would not

Algorithm 2: Smart contract $\mathbb{S}$ with 2FA

```
▷ VARIABLES AND FUNCTIONS OF ENVIRONMENT:
    tx: a current transaction processed by $\mathbb{S}$,
    balance: the current balance of a contract,
    transfer(r, v): transfer v crypto-tokens from a smart contract to r,
▷ DECLARATION OF TYPES:
    Operation { addr, param, pending, type ∈ {TRANSFER, ...} }
▷ DECLARATION OF FUNCTIONS:
function $constructor$(root, pk) public
    operations ← [];                                        ▷ An append-only list
    $PK_{\mathbb{U}}$ ← pk,  $\mathcal{R}$ ← root,  nextOpID ← 0;
    return $\mathbb{S}^{ID}$;                                ▷ Computed by a blockchain platform.
function initOp(a, p, type) public
    assert $\Sigma.verify(tx.\sigma, PK_{\mathbb{U}})$;        ▷ 1st factor of 2FA
    opID ← nextOpID++;
    operations[opID] ← new Operation(a, p, true, type);
function $confirmOp$(otp, $\pi$, opID) public
    assert operations[opID].pending;
    verifyOTP(otp, $\pi$, opID);                             ▷ 2nd factor of 2FA
    execOp(operations[opID]);
    operations[opID].pending ← false;
function $verifyOTP$(otp, $\pi_{opID}$, opID) private
    assert deriveRootHash(otp, $\pi_{opID}$, opID) = $\mathcal{R}$;
function execOp(oper) private
    if TRANSFER = oper.type then
        assert oper.param ≤ balance;
        transfer(oper.addr, oper.param);
```

help to resolve this issue, as the amount of user transferred data would be equal to 176B $\simeq$ 132 mnemonic words $\simeq$ 11 QR codes v1.

We make the observation that it is possible to decouple providing OTPs from providing their proofs. The only data that need to be kept secret are OTPs, while any node of a Merkle tree may potentially be disclosed – no OTP can be derived from these nodes. Therefore, we propose providing OTPs by $\mathbb{A}$, while their proofs can be constructed at $\mathbb{C}$ from stored hashes of OTPs. This modification enables us to fetch the nodes of the proof from the storage of $\mathbb{C}$, while $\mathbb{U}$ has to transfer only the OTP itself from $\mathbb{A}$ when confirming an operation (i.e, $S = 128 \simeq 12$ mnemonic words by default).

Base Version

Secure Bootstrapping. As common in other schemes and protocols, by default, we assume a secure environment for bootstrapping protocol Π_B^S (see Figure 5.1). First, $\mathbb{A}$ generates a secret seed k, which is stored as a recovery phrase by $\mathbb{U}$. $\mathbb{W}$ generates a key-pair $SK_{\mathbb{U}}, PK_{\mathbb{U}} \leftarrow \Sigma.KeyGen()$. Next, $\mathbb{U}$ transfers k from $\mathbb{A}$ to $\mathbb{C}$ in an air-gapped manner (i.e., transcribing a few mnemonic words or scanning a QR code). Then, $\mathbb{C}$ generates OTPs by computing $F_k(i) \mid i \in \{0, 1, \ldots, N-1\}$, where N is the number of leaves (equal to the number of OTPs in the base version). Next, $\mathbb{C}$ computes and stores the leaves of the tree – i.e., the hashes of the OTPs (i.e., $hOTPs$), which do not contain any confidential

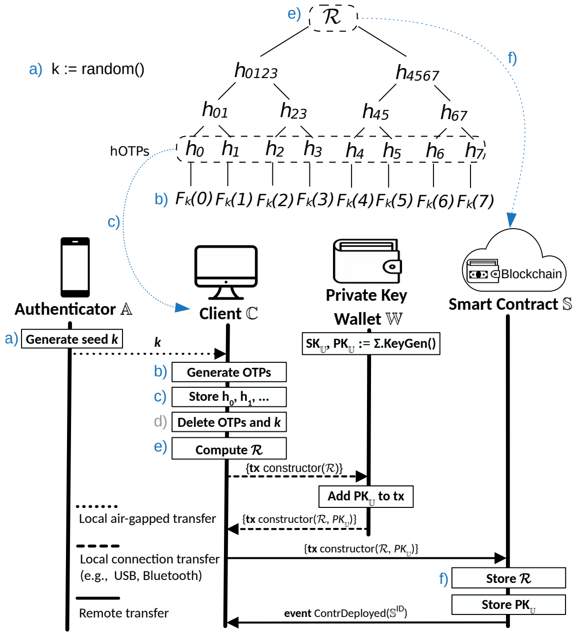


Figure 5.1: Bootstrapping of SmartOTPs in a secure environment (Π_B^S).

data.[2] After this step, k and the OTPs are deleted from $\mathbb{C}$, and $\mathbb{C}$ computes $\mathcal{R}$ from the stored hashes of the OTPs. Then, $\mathbb{C}$ creates a transaction containing constructor of $\mathbb{S}$ (see Algorithm 2) with $\mathcal{R}$ as the argument and passes it to $\mathbb{W}$ for appending $PK_{\mathbb{U}}$. Finally, $\mathbb{C}$ sends the transaction with the constructor to the blockchain where the deployment of $\mathbb{S}$ is made.[3] In the constructor, $\mathcal{R}$ with $PK_{\mathbb{U}}$ are stored and ID of $\mathbb{S}$ (i.e., $\mathbb{S}^{ID}$) is assigned by a blockchain platform and returned in a response.[4] Storing $\mathcal{R}$ and $PK_{\mathbb{U}}$ binds an instance of $\mathbb{S}$ with the user's authenticator $\mathbb{A}$ and the user's private key wallet $\mathbb{W}$, respectively. In detail, $PK_{\mathbb{U}}$ enables $\mathbb{S}$ to verify whether an arbitrary transaction was signed by the user who created $\mathbb{S}$, while $\mathcal{R}$ enables the verification whether the given OTP was produced by the user's $\mathbb{A}$.

Operation Execution. When the wallet framework is initialized, it is ready for executing operations by a two-stage protocol Π_O (see Figure 5.2):

1. **Initialization Stage**. When $\mathbb{U}$ decides to execute an operation with SmartOTPs, he enters the details of the operation into $\mathbb{C}$ that creates a transaction calling *initOp()*, which is provided with operation-specific parameters – the type of operation (e.g., transfer), a numerical parameter (e.g., amount or daily limit), and an address parameter (e.g., recipient). Then, $\mathbb{C}$ sends this transaction to $\mathbb{W}$, which displays the details of the transaction and prompts $\mathbb{U}$ to confirm signing by a hardware button. Upon confirmation, $\mathbb{W}$ signs the transaction by $SK_{\mathbb{U}}$ and sends it back to $\mathbb{C}$. $\mathbb{C}$ forwards

[2] To improve performance during provisioning of proofs, $\mathbb{C}$ might additionally store non-leaf nodes, increasing the requirement on $\mathbb{C}$'s storage 2x.

[3] $\mathbb{C}$ has the template of $\mathbb{S}$ and the deployment process is unnoticeable for the users.

[4] Note that $\mathbb{S}^{ID}$ represents a public identification of $\mathbb{S}$, which serves as a destination for sending cryptotokens to $\mathbb{S}$ by any party.

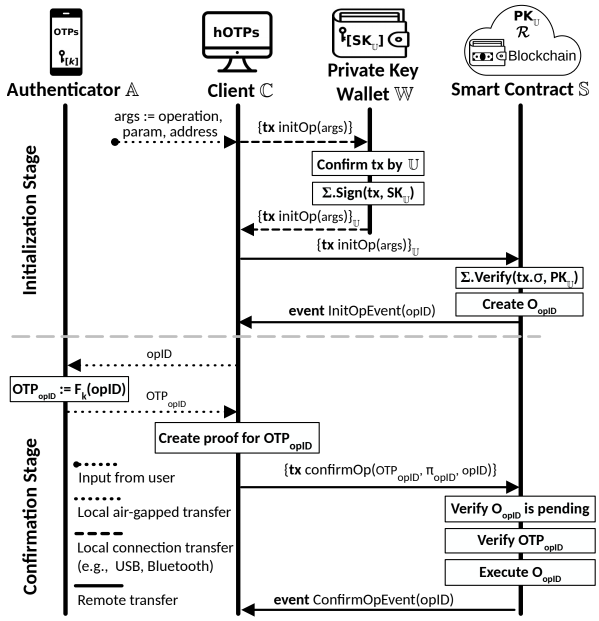

OTPs
[k]
Authenticator $\mathbb{A}$
hOTPs
Client $\mathbb{C}$
[SK$_\mathbb{U}$]
Private Key Wallet $\mathbb{W}$
PK$_\mathbb{U}$
$\mathcal{R}$
Blockchain
Smart Contract $\mathbb{S}$
Initialization Stage
args := operation, param, address
{tx initOp(args)}
Confirm tx by $\mathbb{U}$
Σ.Sign(tx, SK$_\mathbb{U}$)
{tx initOp(args)}$_\mathbb{U}$
{tx initOp(args)}$_\mathbb{U}$
Σ.Verify(tx.σ, PK$_\mathbb{U}$)
Create O$_{opID}$
event InitOpEvent(opID)
opID
OTP$_{opID}$:= F$_k$(opID)
OTP$_{opID}$
Create proof for OTP$_{opID}$
Confirmation Stage
Input from user
Local air-gapped transfer
Local connection transfer (e.g., USB, Bluetooth)
Remote transfer
{tx confirmOp(OTP$_{opID}$, π$_{opID}$, opID)}
Verify O$_{opID}$ is pending
Verify OTP$_{opID}$
Execute O$_{opID}$
event ConfirmOpEvent(opID)

Figure 5.2: Execution of an operation (Π_O).

the transaction to $\mathbb{S}$. In the function *initOp()*, $\mathbb{S}$ verifies whether the signature was created by $\mathbb{U}$ (the first factor), stores the parameters of the operation, and then assigns a sequential ID (i.e., $opID$) to the initiated operation. In the response from $\mathbb{S}$, $\mathbb{C}$ is provided with an $opID$.

2. **Confirmation Stage**. After the transaction (that initiated the operation) is persisted on the blockchain, $\mathbb{U}$ proceeds to the second stage of Π_O. $\mathbb{U}$ enters $opID$ to $\mathbb{A}$, which, in turn, computes and displays OTP_{opID} as $F_k(opID)$. Storing $hOTPs$ computed from OTPs at $\mathbb{C}$ enables $\mathbb{U}$ to transfer only the displayed OTP from $\mathbb{A}$ to $\mathbb{C}$, which can be accomplished in an air-gapped manner. Considering the mnemonic implementation [PRVB13], this means an air-gapped transfer of 12 words in the case of $O = 16$B. Then, $\mathbb{C}$ computes and appends the corresponding proof π_{opID} to the OTP. The proof of the OTP is computed from stored $hOTPs$ in the $\mathbb{C}$'s storage (or directly fetched from the storage if $\mathbb{C}$ stores all nodes of the Merkle tree). Next, $\mathbb{C}$ sends a transaction with OTP_{opID} and its proof π_{opID} to the blockchain, calling the function *confirmOp()* of $\mathbb{S}$, which handles the second factor. This function verifies the authenticity of the OTP (i.e., the first requirement of OTPs) and its association with the requested operation (i.e., the second requirement of OTPs), which together implies the correctness of the provided OTP.[5] In detail, upon calling the *confirmOp()* function with $opID$, OTP_{opID}, and π_{opID} as the arguments, $\mathbb{S}$ reconstructs the root hash from the provided arguments by the function *deriveRootHash()* that is presented in Appendix of [HBH+20a]. If the reconstructed value matches the stored value $\mathcal{R}$, the operation is executed (e.g., crypto-tokens are transferred).

In the following, we present extensions of SmartOTPs, improving its efficiency and usability, and introducing new features.

[5] Note that SmartOTPs meet the third requirement of OTPs by the design.

Bootstrapping in an Insecure Environment

The main advantage of $\Pi_B^{\mathcal{S}}$ described above is its high usability, requiring only an air-gapped transfer of k and connected $\mathbb{W}$. However, $\Pi_B^{\mathcal{S}}$ is not resistant against $\mathcal{A}$ tampering with $\mathbb{C}$; $\mathcal{A}$ might intercept k or forge $\mathcal{R}$ for $\mathcal{R}'$. Similarly, $\mathcal{A}$ might forge $PK_{\mathbb{U}}$ for $PK_{\mathcal{A}}$, while staying unnoticeable for $\mathbb{U}$ who expects that $\mathbb{S}^{ID}$ obtained is correct. Therefore, we propose an alternative bootstrapping protocol $\Pi_B^{\mathcal{I}}$ (see Appendix of [HBH+20a]), assuming that $\mathcal{A}$ can tamper with $\mathbb{C}$ during bootstrapping. In this protocol, first we protect SmartOTPs from the interception of k and then from forging $\mathcal{R}$ and $PK_{\mathbb{U}}$.

To avoid the interception of k, instead of transferring k, $\mathbb{U}$ performs a transfer of all leaves of the Merkle tree (i.e., $hOTPs$) from $\mathbb{A}$ to $\mathbb{C}$, which can be achieved with a microSD card. Note that the leaves are hashes of OTPs, hence they do not contain any confidential data. Next, to protect SmartOTPs from forging of $PK_{\mathbb{U}}$ and $\mathcal{R}$, we require a deterministic computation of $\mathbb{S}^{ID}$ by a blockchain platform using $PK_{\mathbb{U}}$ and $\mathcal{R}$, hence $\mathbb{S}^{ID}$ can be computed and displayed together with $\mathcal{R}$ in $\mathbb{W}$ before the deployment of $\mathbb{S}$. In detail, $\mathbb{S}^{ID}$ is computed as $h(PK_{\mathbb{U}} \parallel \mathcal{R})$, thus each pair consisting of a public key and a root hash maps to the only $\mathbb{S}^{ID}$. However, even with this modification, $\mathcal{R}$ can still be forged by $\mathbb{C}$. Therefore, when transaction with the constructor is sent to $\mathbb{W}$, $\mathbb{U}$ has to compare $\mathcal{R}$ displayed at $\mathbb{W}$ with the one computed and displayed by $\mathbb{A}$. In the case of equality, $\mathbb{U}$ records $\mathbb{S}^{ID}$ displayed in $\mathbb{W}$.

Increasing the Number of OTPs

A small number of OTPs can have negative usability and security implications. First, users executing many transactions[6] would need to create new OTPs often, and thus change their addresses. Second, an attacker possessing $SK_{\mathbb{U}}$ can flood $\mathbb{S}$ with initialized operations, rendering all the OTPs unusable. Therefore, we need to increase the number of OTPs to make the attack unfeasible. However, increasing the number of OTPs linearly increases the amount of data that $\mathbb{C}$ needs to preserve in its storage. For example, if the number of OTPs is 2^{20}, then $\mathbb{C}$ has to store $33.6MB$ of data (considering $S = 16B$ and $\mathbb{C}$ storing all leaves), which is feasible even on storage-limited devices. However, e.g., for 2^{32} OTPs, $\mathbb{C}$ needs to store $137.4GB$ of data, which might be infeasible even on PCs, especially when $\mathbb{C}$ handles multiple instances of SmartOTPs.

To resolve this issue, we modify the base approach by applying a time-space trade-off [Hel80] for OTPs. Namely, we introduce hash chains of which last items are aggregated by the Merkle tree. With such a construction, OTPs can be encoded as elements of chains and revealed layer by layer in the reverse order of creating the chains. This allows multiplication of the number of OTPs by the chain length without increasing the $\mathbb{C}$'s storage but imposing a larger number of hash computations on $\mathbb{S}$ and $\mathbb{A}$. Nonetheless, smart contract platforms set only a low execution cost for $h(.)$.

An illustration of this construction is presented in the bottom left part of Figure 5.3. A hash chain of length P is built from each OTP assumed so far. Then, the last items of

[6] E.g., several smart contracts in Ethereum have over 2^{20} transactions made.

all hash chains are used as the first iteration layer, which provides $\frac{N}{P}$ OTPs.[7] Similarly, the penultimate items of all the hash chains are used as the second iteration layer, etc., until the last iteration layer consisting of the first items of hash chains (i.e., outputs of $F_k(.)$) has been reached (see the middle part of Figure 5.3). We emphasize that introducing hash chains may cause a violation of the requirement on the independence of OTPs if implemented incorrectly; i.e., OTPs from upper iteration layers can be derived from lower layers. Therefore, to enforce this requirement, we invalidate all the OTPs of all the previous iteration layers by a sliding window at $\mathbb{S}$.

Furthermore, if a hash chain were to use the same hash function throughout the entire chain, it would be vulnerable to birthday attacks [HJP05]. To harden a hash chain against a birthday attack, a *domain separation* proposed by Leighton and Micali [LM95] can be used: a different hash function is applied in each step of a hash chain. Note that without domain separation, inverting the ith iterate of $h(.)$ is i times easier than inverting a single hash function (see the proof in [HN01]). Therefore, we use a different hash function for all but the last iteration layer $1 \leq i < P$ as follows:

$$h_{\mathcal{D}[i]}(x) \;\; = \;\; h(P - i + 1 \,||\, x), \tag{5.1}$$

where x represents the OTP from the next iteration layer.

Although domain separation hardens a single hash chain against the birthday attack, this attack is still possible within the current iteration layer, which is an inevitable consequence of using multiple hash chains. Therefore, the number of leaves $\mathcal{L}$ (i.e., N/P) is the parameter that must be considered when quantifying the security level of our scheme (see Section 5.3.5).

With this improvement, $\mathbb{A}$ is updated to provide OTPs by

$$getOTP(i) \;\; = \;\; h_{\mathcal{D}}^{\alpha(i)}\Bigg(F_k\Big(\beta(i)\Big)\Bigg), \tag{5.2}$$

where i is the operation ID, $\alpha(i)$ determines the index in a hash chain, and $\beta(i)$ determines the index in the last iteration layer of OTPs. We provide concrete expressions for $\alpha(i)$ and $\beta(i)$ in Equation 5.4, which involves all proposed improvements and optimizations. A derivation of $\mathcal{R}$ from the OTP at $\mathbb{S}$ needs to be updated as well (see Appendix of [HBH+20a]). In detail, $\mathbb{S}$ executes $P \, - \, \alpha(i) \, - \, 1 = \lfloor \frac{iP}{N} \rfloor$ hash computations, which is a complementary number to the number of hash computations at $\mathbb{A}$ with regard to P. Also, $\mathbb{C}$ has to be modified, requiring computation of a proof to use the leaf index relative to the current iteration layer of OTPs (i.e., $i \; \% \; \frac{N}{P}$).

With this improvement, given the number of leaves equal to 2^{20} and $P = 2^{12}$, $\mathbb{C}$ stores only $33.6MB$ of data and it has 2^{32} OTPs available. On the other hand, this modification implies, on average, the execution of additional $P/2$ hash computations at $\mathbb{S}$, imposing additional costs. However, our experiments show the benefits of this approach (see Section 5.3.6).

[7] For simplicity, we assume that $GCD(N, P) = P$.

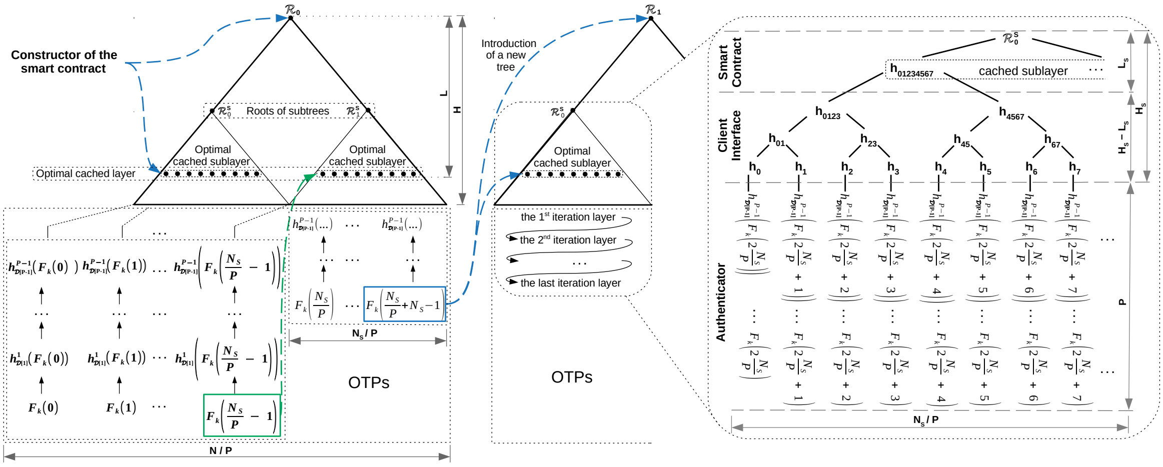


Figure 5.3: An overview of our approach and its improvements.

Depletion of OTPs

Even with the previous modification, the number of OTPs remains bounded, therefore they may be depleted. We propose handling of depleted OTPs by a special operation that replaces the current tree with a new one. To introduce a new tree securely, we propose updating $\mathcal{R}$ value while using the last OTP of the current tree for confirmation. Nevertheless, for this purpose we cannot use Π_O consisting of two stages, as $\mathcal{A}$ possessing $SK_{\mathbb{U}}$ could be "faster" than the user and might initialize the last operation and thus block all the user's funds. If we were to allow repeated initialization of this operation, then we would create a race condition issue.

Algorithm 3: Introduction of a new $\mathcal{R}$ in $\mathbb{S}$

```
$L_1 \leftarrow []$;                                   ▷ Items have form $< h(\mathcal{R}^{new} \parallel OTP) >$
$L_2 \leftarrow []$;                                   ▷ Items have form $< \mathcal{R}^{new} >$
function $1\_newRootHash$(hRootAndOTP) public
    assert $\Sigma.verify(tx.\sigma, PK_{\mathbb{U}})$;
    assert $nextOpID \ \% \ N = N - 1$;                 ▷ The last oper. of tree
    $L_1$.append(hRootAndOTP);
function $2\_newRootHash$($\mathcal{R}^{new}$) public
    assert $\Sigma.verify(tx.\sigma, PK_{\mathbb{U}})$;
    assert nextOpID $\% \ N = N - 1$;                   ▷ The last oper. of tree
    $L_2$.append($\mathcal{R}^{new}$);
function $3\_newRootHash$(otp, $\pi$) public
    assert nextOpID $\% \ N = N - 1$;                   ▷ The last oper. of tree
    verifyOTP(otp, $\pi$, nextOpID);
    if $L_1.len > LEN_{MAX} \mid L_2.len > LEN_{MAX}$ then
        $L_1, L_2 \leftarrow$ [], [];
        return;                                         ▷ To avoid $\mathcal{A}$ DoS-ing $\mathbb{S}$ by gas depletion.
    for $\{j \leftarrow 0;\ j < L_1.len;\ j{+}{+}\}$ do
        for $\{i \leftarrow 0;\ i < L_2.len;\ i{+}{+}\}$ do
            if $h(L_2[i] \parallel otp) = L_1[j]$ then
                $\mathcal{R} \leftarrow L_2[i]$;
                $L_1, L_2 \leftarrow$ [], [];
                nextOpID++;
```

To avoid this race condition issue, we propose a protocol Π_{NR} that replaces $\mathcal{R}$ during three stages of interaction with the blockchain, which requires two append-only lists L_1 and L_2 (see Algorithm 3):

1. $\mathbb{U}$ enters OTP_{N-1} to $\mathbb{C}$. $\mathbb{C}$ sends $h(OTP_{N-1} \parallel \mathcal{R}^{new})$ to $\mathbb{S}$, which appends it to L_1.
2. $\mathbb{C}$ sends $\mathcal{R}^{new}$ to $\mathbb{S}$, which appends it to L_2.
3. $\mathbb{C}$ passes OTP_{N-1} with π_{N-1} to $\mathbb{S}$, where the first matching entries of L_1 and L_2 are located to perform the introduction of $\mathcal{R}^{new}$. Finally, the lists are cleared for future updates.

Locating the first entries in the lists relies on the append-only feature of lists, hence no $\mathcal{A}$ can make the first valid pair of entries in the lists. Similarly as in Π_B, we propose two variants of Π_{NR} intended for secure (i.e., $\Pi^{\mathcal{S}}_{NR}$) and insecure environment (i.e., $\Pi^{\mathcal{I}}_{NR}$). In $\Pi^{\mathcal{I}}_{NR}$ (see Appendix of [HBH+20a] for detailed description), $\mathbb{A}$ must compute and display $h(OTP_{N-1} \parallel \mathcal{R}^{new})$ and $\mathcal{R}^{new}$ to enable protection against $\mathcal{A}$ that tampers with $\mathbb{C}$. Hence, $\mathbb{U}$ can verify the equality of items displayed at $\mathbb{W}$ with the ones displayed at $\mathbb{A}$ during the first and the second stage of $\Pi^{\mathcal{I}}_{NR}$, preventing $\mathcal{A}$ from forging the tree. To adapt this improvement at $\mathbb{C}$, $\mathbb{C}$ needs to store all nodes of the new tree. Therefore, $\mathbb{U}$ provides $\mathbb{C}$ with all nodes of the new tree, transferred from $\mathbb{A}$ on a microSD card. In the case of $\Pi^{\mathcal{S}}_{NR}$, the nodes of the new tree are transferred by a transcription of k from $\mathbb{A}$ to $\mathbb{C}$ and no values are displayed at $\mathbb{W}$ and $\mathbb{A}$ for $\mathbb{U}$'s verification.

Cost & Security Optimizations

Caching in the Smart Contract. With a high Merkle tree, the reconstruction of $\mathcal{R}$ from a leaf node may be costly. Although the number of hash computations stemming from the Merkle tree is logarithmic in the number of leaves, the cost imposed on the blockchain platform may be significant for higher trees. We propose to reduce this cost by caching an arbitrary tree layer of depth L at $\mathbb{S}$ and do proof verifications against a cached layer. Hence, every call of *deriveRootHash()* will execute L fewer hash computations in contrast to the version that reconstructs $\mathcal{R}$, while $\mathbb{C}$ will transfer by L fewer elements in the proof.

The minimal operational cost can be achieved by directly caching leaves of the tree, which accounts only for hash computations coming from hash chains, not a Merkle tree. However, storing such a high amount of cached data on the blockchain is too expensive. Therefore, this cost optimization must be viewed as a trade-off between the depth L of the cached layer and the price required for the storage of such a cached layer on the blockchain (see Section 5.3.6).

We depict this modification in the left part of Figure 5.3, and we show that an optimal caching layer can be further partitioned into caching sublayers of subtrees (introduced later). To enable this optimization, the cached layer of the Merkle tree must be stored in the constructor of $\mathbb{S}$. From that moment, the cached layer replaces the functionality of $\mathcal{R}$, reducing the size of proofs. During the confirmation stage of Π_O, an OTP and its proof are used for the reconstruction of a particular node in the cached layer, instead of $\mathcal{R}$. Then the reconstructed value is compared with an expected node of the cached layer. The index

Algorithm 4: Introduction of the next subtree at $\mathbb{S}$

```
currentSubLayer[];                                          ▷ Adjusted in the constructor
function nextSubtree(nextSubLayer, otp, $\pi_{otp}$, $\pi_{sr}$) public
    assert nextOpID % $N \neq N - 1$;                       ▷ Not the last op. of parent
    assert nextOpID % $N_S = N_S - 1$;                      ▷ The last op. of subtree
    assert currentSubLayer.len = nextSubLayer.len;
    assert deriveRootHash(otp, $\pi_{otp}$, nextOpID) = $\mathcal{R}$;
    currentSubLayer ← nextSubLayer;
    $\mathcal{R}^s$ ← reduceMT(currentSubLayer, currentSubLayer.len);
    assert subtreeConsistency($\mathcal{R}^s$, $\pi_{sr}$, $\mathcal{R}$);
    nextOpID++;                                             ▷ Accounts for this introduction of a subtree
```

of an expected node is computed as

$$idxInCache(i) \;=\; \left\lfloor \left(i \,\%\, \frac{N}{P} \right) \,/\, 2^{H-L} \right\rfloor, \tag{5.3}$$

where i is the ID of an operation.

Partitioning to Subtrees. The caching of the optimal layer minimizes the operational costs of SmartOTPs, but on the other hand, it requires prepayment for storing the cache on the blockchain. If the cached layer were to contain a high number of nodes, then the initial deployment cost could be prohibitively high, and moreover, the user might not deplete all the prepaid OTPs. On top of that, after revealing the first iteration layer of OTPs, the security of our scheme described so far is decreased by $log_2(N/P)$ bits due to the birthday attack (see Section 5.3.5) on OTPs. Hence, bigger trees suffer from higher security loss than smaller trees.

To overcome the prepayment issue and to mitigate the birthday attack, we propose partitioning an optimal cached layer to smaller groups having the same size, forming sublayers that belong to subtrees (see the left part of Figure 5.3). The obtained security loss is $log_2(N_S/P)$, $N_S \ll N$.

Starting with the deployment of $\mathbb{S}$, the cached sublayer of the first subtree and the "parent" root hash (i.e., $\mathcal{R}$) are passed to the constructor; the cached sublayer is stored on the blockchain and its consistency against $\mathcal{R}$ is verified. Then during the operational stage of Π_O, when confirmation of operation is performed, the passed OTP is verified against an expected node in the cached sublayer of the current subtree, saving costs for not doing verification against $\mathcal{R}$ (see Appendix of [HBH+20a]).

If the last OTP of the current subtree is reached, then no operation other than the introduction of the next subtree can be initialized (see the green dashed arrow in Figure 5.3). We propose a protocol Π_{ST} for the introduction of the next subtree (see Appendix of [HBH+20a] for the detailed description). Namely, $\mathbb{C}$ introduces the next subtree in a single step by calling a function *nextSubtree()* of $\mathbb{S}$ with the arguments containing: (1) the last OTP of the current subtree $OTP_{(N_S-1)+\delta N_S}$, $\delta \in \{1, \ \ldots, \ N/N_S - 1\}$, (2) its proof π_{otp}, (3) the cached sublayer of the next subtree, and (4) the proof π_{sr} of the next subtree's root; all items but OTP are computed by $\mathbb{C}$. The pseudo-code of the next subtree introduction at $\mathbb{S}$ is shown in Algorithm 4. The current subtree's cached sublayer is replaced by the new one, which is verified by the function $subtreeConsistency()$ against $\mathcal{R}$ with the use of the passed proof π_{sr} of the new subtree's root hash $\mathcal{R}^s$. Note that introducing a new subtree invalidates all initialized yet to be confirmed operations of the previous subtree.

At $\mathbb{A}$, this improvement requires accommodating the iteration over layers of hash chains in shorter periods. Hence, $\mathbb{A}$ provides OTPs by Equation 5.2 with the following expressions:

$$\begin{aligned} \alpha(i) = &\quad P - \left\lfloor \frac{(i \,\%\, N_S)P}{N_S} \right\rfloor - 1, \\ \beta(i) = &\quad \left\lfloor \frac{i}{N_S} \right\rfloor \frac{N_S}{P} + \left(i \,\%\, \frac{N_S}{P} \right), \end{aligned} \tag{5.4}$$

where i is an operation ID and N_S is the number of OTPs provided by a single subtree. We remark, that due to this optimization, the update of a new parent root $\mathcal{R}$ as well as the constructor of $\mathbb{S}$ requires, additionally to Algorithm 3 and Algorithm 2, the introduction of a cached sublayer of the first subtree (omitted here for simplicity).

5.3.5 Security Analysis

We analyze the security of SmartOTPs and its resilience to attacker models under the assumption of random oracle model $\mathcal{RO}$.

Security of OTPs

OTPs in our scheme are related to two cryptographic constructs: a list of hash chains and the Merkle tree aggregating their last values. In this subsection, we assume an adversary $\mathcal{A}$ who is trying to invert OTPs, and we give a concrete expressions for security of our scheme. Since we employ the hash domain separation technique [LM95] for hash chains, each hash execution can be seen as an execution of an independent hash function. For such a construction, Kogan et al. give the following upper bound (see Theorem 4.6 in [KMB17]) on the advantage of $\mathcal{A}$ breaking a chain:

$$Pr[\mathcal{A}\ breaks\ a\ chain] \leq \frac{2Q + 2P + 1}{2^S}, \tag{5.5}$$

where Q is the number of queries that $\mathcal{A}$ can make to $h(.)$, P is the chain length, and S is the bit-length of OTPs (and the output of $h(.)$). Kogan et al. [KMB17] proved that inverting a hash chain hardened by the domain separation imposes a loss of security equal to the factor of 2. Therefore, to make a hardened hash chain as secure as λ-$bit\ \mathcal{RO}$, it is enough to set $S = \lambda + 2$. E.g., to achieve 128-bit security, S should be equal to 130.

SmartOTPs without Subtrees. This scheme (see Section 22) uses a Merkle tree that aggregates $\mathcal{L} = \frac{N}{P}$ hash chains, where the chains are created independently of each other; they have the same length and the same number of OTPs. $\mathcal{A}$ can win by inverting any of the chains; hence, the probability that this scheme is secure is

$$Pr[Scheme\ is\ secure] = \left(1 - \frac{2Q + 2P + 1}{2^S}\right)^{\mathcal{L}}. \tag{5.6}$$

We can apply the alternative form of Bernoulli's inequality $(1 - x)^{\mathcal{L}} \geq 1 - x\mathcal{L}$, where $\mathcal{L} \geq 1$ and $0 \leq x \leq 1$ must hold. In our case, the input conditions hold since the number of hash chains is always greater than one and the probability that $\mathcal{A}$ breaks a single chain from Equation 5.5 fits the range of x (i.e., $0 \leq \frac{2Q+2P+1}{2^S} \leq 1$). Hence, we lower-bound the probability from Equation 5.6 as follows:

$$Pr[Scheme\ is\ secure] \geq 1 - \frac{\mathcal{L}(2Q + 2P + 1)}{2^S}. \tag{5.7}$$

Corollary 2. *To make SmartOTPs without partitioning into subtrees as secure as λ-bit $\mathcal{RO}$, it is enough to set $S = \lambda + 2 + log_2(\mathcal{L})$.*

For example, to achieve 128-bit security with $\mathcal{L} = 64$ and $P \geq 1$, S should be equal to 136, and thus an OTP can be transferred by one QR code v1 or 13 mnemonic words.

Full SmartOTPs. The full SmartOTPs scheme contains partitioning into subtrees, in which all leaves of the next subtree "are visible" only after depleting OTPs of the current subtree (and using OTPs from the 1st iteration layer of the next subtree). This improves the security of our scheme under the assumption that $\mathbb{C}$'s storage is not compromised by $\mathcal{A}$, which is true for $\mathcal{A}$ that possesses $PK_{\mathbb{U}}$ or $\mathbb{A}$. Therefore, we replace $\mathcal{L}$ in Equation 5.7 for $\mathcal{L}_{\mathcal{S}} = \frac{N_S}{P}$, $N_S \ll N$.

Corollary 3. *To make the full scheme of SmartOTPs as secure as λ-bit $\mathcal{RO}$, it is enough to set $S = \lambda + 2 + log_2(\mathcal{L}_{\mathcal{S}})$.*

Therefore, to achieve 128-bit security with $\mathcal{L} = \frac{N}{N_S}\mathcal{L}_{\mathcal{S}}$, $\mathcal{L}_{\mathcal{S}} = 64$, and $P \geq 1$, S should be equal to 136, and thus an OTP can be transferred by a QR code v1 or 13 mnemonic words. To achieve the same security with $\mathcal{L}_{\mathcal{S}} = 1024$, we need to set $S = 140$, and thus an OTP can be transferred in a QR code v2 or 13 mnemonic words.

The Attacker Possessing $SK_{\mathbb{U}}$

Theorem 1. *$\mathcal{A}$ with access to $SK_{\mathbb{U}}$ is able to initiate operations by Π_O but is unable to confirm them.*

Justification. The security of Π_O is achieved by meeting all requirements on general OTPs (see Section 5.3.3). In detail, the requirement on the *independence* of two different OTPs is satisfied by the definition of $F_k(.) \equiv h(k \parallel .)$, where $h(.)$ is instantiated by $\mathcal{RO}$. This is applicable when $P = 1$. However, if $P > 1$, then items in previous iteration layers of OTPs can be computed from the next ones. Therefore, to enforce this requirement, we employ an explicit invalidation of OTPs belonging to all previous iteration layers by a sliding window at $\mathbb{S}$ (see Section 26). The requirement on the *linkage* of each OTP_i

with operation O_i is satisfied due to (1) $\mathcal{RO}$ used for instantiation of $h(.)$ and (2) by the definition of the Merkle tree, preserving the order of its aggregated leaves. By meeting these requirements, $\mathcal{A}$ is able to initiate an operation O_j in the first stage of Π_O but is unable to use an OTP_i intercepted in the second stage of Π_O to confirm O_j, where $j \neq i$. Finally, the requirement on the *authenticity* of OTPs is ensured by a random generation of k and by anchoring $\mathcal{R}$ associated with k at the constructor of $\mathbb{S}$. □

Theorem 2. *Assuming $\delta \in \{0, \ldots, \frac{N}{N_S} - 2\}$, $\mathcal{A}$ with access to $SK_{\mathbb{U}}$ is unable to deplete all OTPs or misuse a stolen OTP that introduces the $(\delta + 1)th$ subtree by Π_{ST}.*

Justification. When all but one OTPs of the δth subtree are depleted, the last remaining operation $O_{(N_S-1)+\delta N_S}$, $\delta \in \{0, \ \ldots, \ \frac{N}{N_S} - 2\}$ is enforced by $\mathbb{S}$ to be the introduction of the next subtree. This operation is executed in a single transaction calling the function $nextSubtree()$ of $\mathbb{S}$ (see Algorithm 4) requiring the corresponding $OTP_{(N_S-1)+\delta N_S}$ that is under control of $\mathbb{U}$; hence $\mathcal{A}$ cannot execute the function to proceed with a further depletion of OTPs in the $(\delta + 1)th$ subtree. If $\mathcal{A}$ were to intercept $OTP_{(N_S-1)+\delta N_S}$ during the execution of Π_{ST} by $\mathbb{U}$, he could use the intercepted OTP only for the introduction of the next valid subtree since the function $nextSubtree()$ also checks a valid cached sublayer of the $(\delta + 1)$th subtree against the parent root hash $\mathcal{R}$. □

Theorem 3. *Assuming $\delta = \frac{N}{N_S} - 1$, $\mathcal{A}$ with access to $SK_{\mathbb{U}}$ is neither able to deplete all OTPs nor introduce a new parent tree nor render SmartOTPs unusable.*

Justification. In contrast to the adjustment of the next subtree, the situation here is more difficult to handle, since the new parent tree cannot be verified at $\mathbb{S}$ against any paramount field. If we were to use Π_O while constraining to the last initialized operation $O_{(N-1)+\eta N}$, $\eta \in \{0, 1, \ldots\}$ of the parent tree, then $\mathcal{A}$ could render SmartOTPs unusable by submitting an arbitrary $\mathcal{R}$ in $initOp()$, thus blocking all the funds of the user. If we were to allow repeated initialization of this operation, then we would create a race condition issue. Therefore, this operation needs to be handled outside of the protocol Π_O, using two unlimited append-only lists L_1 and L_2 that are manipulated in three stages of interaction with the blockchain (see Section 22). In the first stage, $h(\mathcal{R}^{new} \parallel OTP_{(N-1)+\eta N})$ is appended to L_1, hence $\mathcal{A}$ cannot extract the value of OTP. In the second stage, $\mathcal{R}^{new}$ is appended to L_2, and finally, in the third stage, the user reveals the OTP for confirmation of the first matching entries in both lists. Although $\mathcal{A}$ might use an intercepted OTP from the third stage for appending malicious arguments into L_1 and L_2, when he proceeds to the third stage and submits the intercepted OTP to $\mathbb{S}$, the user's entries will match as the first ones. □

The Attacker Tampering with the Client

Theorem 4. *If $\mathbb{C}$ is tampered with after Π_B, $\mathbb{U}$ can detect such a situation and prevent any malicious operation from being initialized.*

Justification. If we were to assume that $\mathbb{W}$ is implemented as a software wallet (or hardware wallet without a display), then $\mathcal{A}$ tampering with $\mathbb{C}$ might also tamper with the $\mathbb{W}$'s software running on the same machine. This would in turn enable a malicious operation to be initialized and further confirmed by $\mathbb{U}$, since $\mathbb{U}$ would be presented with a legitimate data in $\mathbb{C}$ and $\mathbb{W}$, while the transactions would contain malicious data. Therefore, we require that $\mathbb{W}$ is implemented as a hardware wallet with a display, which exposes only signing capabilities, while $SK_{\mathbb{U}}$ never leaves the device (e.g., [Tre18, Kee18, Bit18d, ELL19]). Due to it, $\mathbb{U}$ can verify the details of a transaction being signed in $\mathbb{W}$ and confirm signing only if the details match the information shown in $\mathbb{C}$ (for Π_O) or $\mathbb{A}$ (for $\Pi_{NR}^{\mathcal{I}}$). We refer the reader to the work of Arapinis et al. [AGKK19] for the security analysis of hardware wallets with displays. □

Theorem 5. *If $\mathbb{C}$ is tampered with during an execution of $\Pi_B^{\mathcal{I}}$, $\mathcal{A}$ can neither intercept k nor forge $\mathcal{R}$ nor forge $PK_{\mathbb{U}}$.*

Justification. When the protocol $\Pi_B^{\mathcal{I}}$ is used, instead of an air-gapped transfer of k from $\mathbb{A}$ to $\mathbb{C}$, $\mathbb{U}$ transfers leaves of the Merkle tree by microSD card. The leaves represent hashes of OTPs in the base version or the hashes of the last items of hash chains in the full version of SmartOTPs. In both versions, the transferred data do not contain any secrets, hence $\mathcal{A}$ cannot take advantage of intercepting them. The next option that $\mathcal{A}$ may seek for is to forge $\mathcal{R}$ for $\mathcal{R}'$ and $PK_{\mathbb{U}}$ for $PK_{\mathcal{A}}$, which results in different $\mathbb{S}^{ID}$ than in the case of $\mathcal{R}$ and $PK_{\mathbb{U}}$, since $\mathbb{S}^{ID}$ is computed as $h(PK_{\mathbb{U}} \parallel \mathcal{R})$. While $PK_{\mathbb{U}}$ is stored at $\mathbb{W}$, the authenticity of $\mathcal{R}$ needs to be verified by $\mathbb{U}$ who compares displays of $\mathbb{A}$ and $\mathbb{W}$. Only in the case of equality, $\mathbb{U}$ knows that $\mathbb{S}^{ID}$ displayed in $\mathbb{W}$ maps to legitimate $PK_{\mathbb{U}}$ and $\mathcal{R}$. □

The Attacker Possessing the Authenticator

It is trivial to see that $\mathcal{A}$ with access to $\mathbb{A}$ is unable to initialize any operation with SmartOTPs since he does not hold $PK_{\mathbb{U}}$.

Further Properties and Implications

Requirement on Block Confirmations. Most cryptocurrencies suffer from long time to finality, potentially enabling the accidental forks, which create parallel inconsistent blockchain views. On the other hand, this issue is not present at blockchain platforms with fast finality, such as Algorand [GHM$^+$17], HoneyBadgerBFT [MXC$^+$16], or StrongChain [SRHS19]. In blockchains with long time to finality, overly fast confirmation of an operation may be dangerous, as, if an operation were initiated in an "incorrect" view, an attacker holding $SK_{\mathbb{U}}$ would hijack the OTP and reuse it for a malicious operation settled in the "correct" view. To prevent this threat, the recommendation is to wait for several block confirmations to ensure that an accidental fork has not happened. For example, in Ethereum, the recommended number of block confirmations to wait is 12 (i.e., $\sim$3 minutes). Note that such waiting can be done as a background task of $\mathbb{C}$, hence $\mathbb{U}$ does not have to wait: (1) considering that $\mathcal{A}$ possesses $SK_{\mathbb{U}}$, $\mathbb{C}$ can detect such a fork during the wait and resubmit the $initOp()$ transaction, (2) in the case of $\mathcal{A}$ tampering with $\mathbb{C}$, no operation can be initialized since $\mathbb{U}$ never signs $\mathcal{A}$'s transaction (due to the hardware wallet), and (3) $\mathcal{A}$ possessing $\mathbb{A}$ cannot initialize any operation as well.

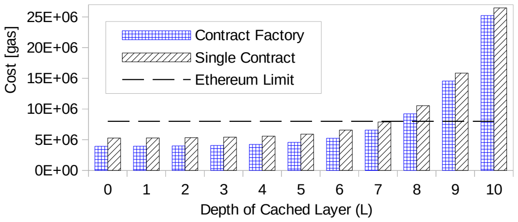


Figure 5.4: Deployment costs ($H = H_S$).

Attacks with a Post Quantum Computer. Although a resilience to quantum computing (QC) is not the focus of our work, it is of worthy to note that our scheme inherits a resilience to QC from the hash-based cryptography. The resilience of our scheme to QC is dependent on the output size of $h(.)$. A generic QC attack against $h(.)$ is Grover's algorithm [Gro96], providing a quadratic speedup in searching for the input of the black box function. As indicated by Amy et al. [ADMG$^+$16], using this algorithm under realistic assumptions, the security of SHA-3 is reduced from 256 to 166 bits. Applying these results to OTPs with 128-bit security from examples in Section 5.3.5, we obtain 98-bits post-QC security. Further, when assuming the example with $\mathcal{L} = 64$ from Section 5.3.5 and [ADMG$^+$16], to achieve 128-bits of post-QC security, we estimate the length of OTPs to 205-bits (i.e., 19 mnemonic words).

5.3.6 Implementation

We have selected the Ethereum platform and the Solidity language for the implementation of $\mathbb{S}$, HTML/JS for DAPP of $\mathbb{C}$, Java for smartphone App of $\mathbb{A}$, and Trezor T&One [Tre18] for $\mathbb{W}$. We selected $S = 128$ bits, which has practical advantages for an air-gapped $\mathbb{A}$, producing OTPs that are 12 mnemonic words long or a QR code v1 (with a capacity of 17B). Next, we used SHA-3 with truncated output to 128 bits as $h(.)$. We selected the size of k equal to 128 bits, fitting 12 mnemonic words $\simeq$ 1 QR code v1.

So far, we have considered only the crypto-token transfer operation. However, our proposed protocol enables us to extend the set of operations. For demonstration purposes, we extended the operation set by supporting daily limits and last resort information (see Appendix of [HBH+20a]). In addition, we made a hardware implementation of $\mathbb{A}$ using NodeMCU [Nod18] equipped with ESP8266 with the overall cost below \$5 (see Appendix of [HBH+20a]). The source code of our implementation and videos are available at `https://github.com/ivan-homoliak-sutd/SmartOTPs`.

Analysis of the Costs

In this section, we analyze the costs of our approach using the same bit-length S for $h(.)$ as well as for OTPs. S significantly influences the gas consumption for storing the cached layer on the blockchain. We remark that measured costs can also be influenced by EVM internals (e.g., 32B-long words/alignment). We assumed 8M as the maximum gas limit at the Ethereum main network, which, however, has already changed since this research was made.

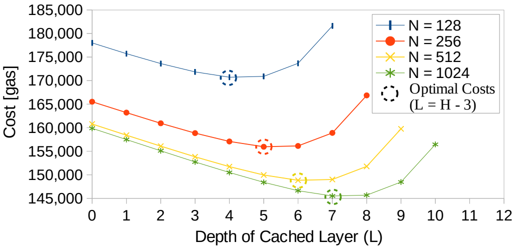
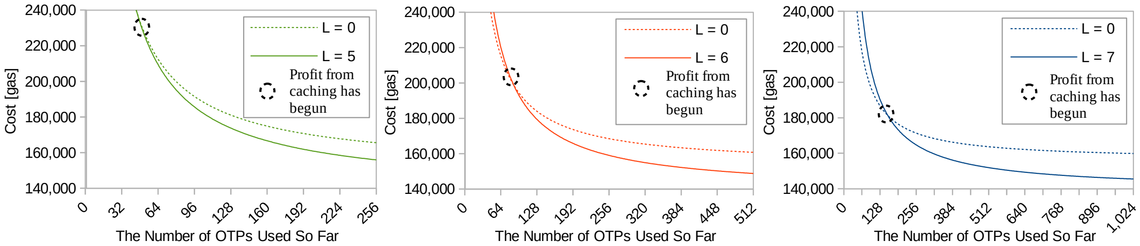


Figure 5.5: Average total cost per transfer ($H = H_S$).



(a) $H = 8$ (b) $H = 9$ (c) $H = 10$

Figure 5.6: Rolling average cost per transfer ($H = H_S$).

Costs Related to the Merkle Tree. First, we abstracted from the concept of subtrees and hashchains to analyze a single tree (i.e., $H = H_S$). The deployment costs of our scheme with respect to the depth L ($\equiv L_S$) of the cached layer are presented in Figure 5.4, which contains also the variant with the contract factory (saving $\sim 1.3M$, regardless of L_S). Although the cost of each operation supported by Π_O is similar, here we selected the transfer of crypto-tokens O^t, and we measured the total cost of O^t as follows:

$$\begin{aligned}
O^t\_cost(L, N, P) &= \overline{cost}\big(O^t(L, N, P)\big) + \frac{cost\big(O^d(L)\big)}{N},\\
\overline{cost}\big(O^t(L, N, P)\big) &= \frac{1}{N}\sum_{i=1}^{N} cost\big(O^t_i(L, N, P)\big),\\
cost\big(O^t_i(L, N, P)\big) &= cost\big(O_i^{t.init}\big) + cost\Big(O_i^{t.confirm}(L, N, P)\Big),
\end{aligned}$$

where *cost()* measures the cost of an operation in gas units, and O_d represents the deployment operation. As the purpose of the cached layer is to reduce the number of hash computations in *confirmOp()*, the size of an optimal cached layer is subject to a trade-off between the cost of storing the cached layer on the blockchain and the savings benefit of the caching. In Figure 5.5, we can see that the total average cost per transfer decreases with the increasing number of OTPs, as the deployment cost is spread across more OTPs. The optimal point depicted in the figure minimizes O^t by balancing $cost(O^d(L))$ and $\overline{cost}(O^t(L, N, P))$. We see that $L = H - 3$ for such an optimal point.

Next, we explored the number of transfer operations to be executed until a profit of the caching has begun (see Figure 5.6). We computed a rolling average cost per O^t, while distinguishing between the optimal caching layer and disabled caching. We measured the cost of introducing the next subtree within a parent tree depending on L_S, while we set $H = 20$ and $H_S = 10$ (see Figure 5.7). We found out that when subtrees (and their

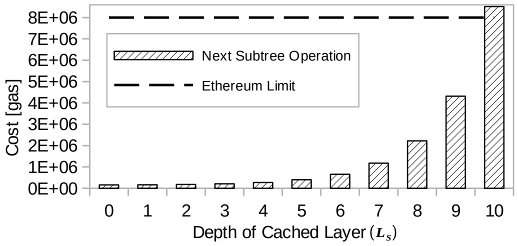
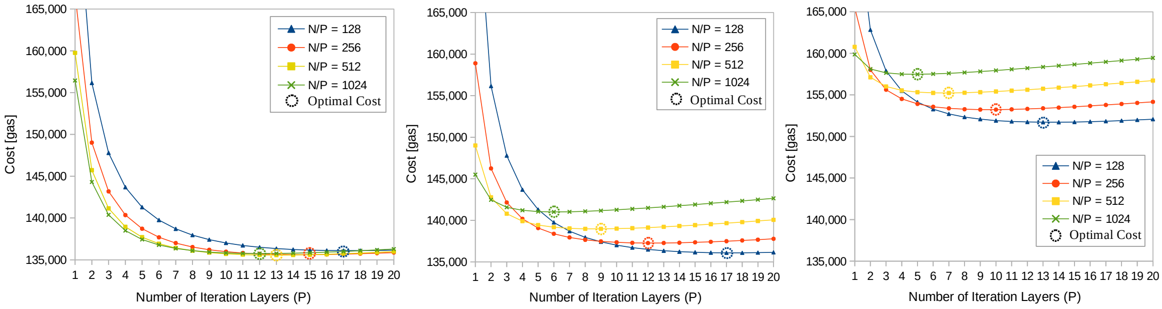


Figure 5.7: Cost of introducing the next subtree ($H = 20$, $H_S = 10$).



(a) $L = H$

(b) $L = 7$

(c) $L = 0$

Figure 5.8: Average total cost per transfer with regards to the length P of hash chains.

cached sublayers) are introduced within a dedicated operation, it is significantly cheaper compared to the introduction of a subtree during the deployment.

Costs Related to Hash Chains. Since each iteration layer of hash chains contributes to an average cost of $confirmOp()$ with around the same value, we measured this value on a few trees with P up to 512. Next, using this value and the deployment cost, we calculated the average total cost per transfer by adding layers of hash chains to a tree with $H = H_S$, thus increasing N by a factor of P until the minimum cost was found. As a result, the optimal caching layer shifted to the leaves of the tree (see Figure 5.8a), which would however, exceed the gas limit of Ethereum. To respect the gas limit, we adjusted $L = 7$, as depicted in Figure 5.8b. In contrast to the configurations with $L = 0$ and $P = 1$ (from Figure 5.5), we achieved savings of 27.80%, 19.61%, 14.95%, and 12.51% for trees with H equal to 7, 8, 9, and 10, respectively. For completeness, we calculated costs for $L = 0$ as well (see Figure 5.8c). Note that for $L = 0$ and $L = 7$, smaller trees are "less expensive," as they require less operations related to the proof verification in contrast to bigger trees; these operations consume substantially more gas than operations related to hash chains. Although we minimized the total cost per transfer by finding an optimal P, we highlight that increasing P contributes to the cost only minimally but on the other hand, it increases the variance of the cost. Hence, one may set this parameter even at higher values, depending on the use case.

5.4 Contributing Papers

The papers that contributed to this research direction are enumerated in the following, while highlighted papers are attached to this thesis in their original form.

[HBH+20a] **Ivan Homoliak, Dominik Breitenbacher, Ondrej Hujnak, Pieter Hartel, Alexander Binder, and Pawel Szalachowski. SmartOTPs: An air-gapped 2-factor authentication for smart-contract wallets. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies, AFT 2020, New York, NY, USA, October 21-23, 2020*. ACM, 2020.**

[HBH+20b] Ivan Homoliak, Dominik Breitenbacher, Ondrej Hujnak, Pieter Hartel, Alexander Binder, and Pawel Szalachowski. SmartOTPs: An Air-Gapped 2-Factor Authentication for Smart-Contract Wallets (Extended Version). *arXiv preprint arXiv:1812.03598*, 2020.

Chapter 6

Electronic Voting

In this chapter, we present our contributions to the area of decentralized applications for remote electronic voting using blockchain, which belongs to the application layer of our security reference architecture (see Chapter 3). In particular, this chapter is focused on privacy, scalability, security, and other practical aspects of blockchain-based e-voting, and it is based on the papers [HLS23, SH23, VSH23] (see also Section 6.4).

First, we review the desired properties of e-voting and then we introduce BBB-Voting, a blockchain-based boardroom voting that provides 1-out-of-k choices and the maximum voter privacy in the setting that outputs the full tally of votes, while additionally offering a mechanism for resolution of faulty participants (see Section 6.1).

Next, we aim to resolve the scalability limitation of the self-tallying approaches for boardroom voting (such as OVN [MSH17] and BBB-Voting [HLS23]) while maintaining security and maximum voter privacy. Therefore, we introduce SBvote [SH23], a decentralized blockchain-based e-voting protocol providing scalability in the number of participants by grouping them into voting booths instantiated as dedicated smart contracts that are controlled and verified by the aggregation smart contract. We base our work on BBB-Voting since it enables more than two vote choices and recovery of faulty participants in contrast to OVN. See details of SBvote in Section 6.2.

Finally, we identify two shortcomings in present governance systems for voting: (a) the inability of participants to change their vote between two consecutive elections (e.g., that might be a few years apart), and (b) a manipulation of participants via peak-end effect [DF18, HL14, Wle15]. As a response, we propose Always-on-Voting (AoV) framework [VSH23] that has four key features: (1) it works in repetitive epochs, (2) voters are allowed to change their vote anytime before the end of each epoch, (3) ends of epochs are randomized and unpredictable, and (4) only the supermajority of votes can change the previous winning vote choice. AoV uses public randomness and commitments to a future event to determine when the current epoch should end. See details of AoV in Section 6.3, where we analyze two different adversaries.

6.0.1 Properties of E-Voting

A voting protocol is expected to meet several properties. A list of such properties appears in the works of Kiayias and Yung [KY02], Groth [Gro04], and Cramer et al. [CGS97].

(1) Privacy of Vote: ensures the secrecy of the ballot contents [KY02]. Hence, a participant's vote must not be revealed other than by the participant herself upon her discretion (or through the collusion of all remaining participants). Usually, privacy is ensured by trusting authorities in traditional elections or by homomorphic encryption in some decentralized e-voting solutions (e.g., [KY02, HRZ10, MSH17, SGY20, DMMM18]).

(2) Perfect Ballot Secrecy: is an extension of the privacy of the vote, stating that a partial tally (i.e., prior to the end of voting) is available only if all remaining participants are involved in its computation.

(3) Fairness: ensures that a tally may be calculated only after all participants have submitted their votes. Therefore, no partial tally can be revealed to anyone before the end of the voting protocol [KY02].

(4a) Universal Verifiability: any involved party can verify that all cast votes are correct and they are correctly included in the final tally [KY02].

(4b) End-to-End (E2E) Verifiability: The verifiability of voting systems is also assessed by E2E verifiability [BRR$^+$15] as follows:

- ***cast-as-intended:*** a voter can verify the encrypted vote contains her choice of candidate,
- ***recorded-as-cast:*** a voter can verify the system recorded her vote correctly,
- ***tallied-as-recorded:*** any interested party is able to verify whether the final tally corresponds to the recorded votes.

A voting system that satisfies all these properties is end-to-end verifiable.

(5) Dispute-Freeness: extends the notion of verifiability. A dispute-free [KY02] voting protocol contains built-in mechanisms eliminating disputes between participants; therefore, anyone can verify whether a participant followed the protocol. Such a scheme has a publicly-verifiable audit trail that contributes to the reliability and trustworthiness of the scheme.

(6) Self-Tallying: once all the votes are cast, any involved party can compute the tally without unblinding the individual votes. Self-tallying systems need to deal with the fairness issues (see (3) above) because the last participant is able to compute the tally even before casting her vote. This can be rectified with an additional verifiable dummy vote [KY02].

(7) Robustness (Fault Tolerance): the voting protocol is able to recover from faulty (stalling) participants, where faults are publicly visible and verifiable due to dispute-freeness [KY02]. Fault recovery is possible when all the remaining honest participants are involved in the recovery.

(8) Resistance to Serious Failures: Serious failures are defined as situations in which voting results were changed either by a simple error or an adversarial attack. Such a change may or may not be detected. If detected in non-resistant systems, it is irreparable without restarting the entire voting [PSNR21].

(9) Receipt-Freeness: A participant is unable to supply a receipt of her vote after casting the vote. The goal of receipt-freeness is to prevent vote-selling. This property also prevents a post election coercion [BT94, HS00], where a participant may be coerced to reveal her vote.

(10) Dispute-Freeness. The protocol's design prevents any disputes among involved parties by allowing anyone to verify whether a participant followed the protocol.

6.1 BBB-Voting

6.1.1 System Model

Our system model of BBB-Voting [HLS23] has the following actor/components: (1) *a participant (P)* who votes, (2) *a voting authority (VA)* who is responsible for validating the eligibility of Ps to vote, their registration, and (3) *a smart contract (SC)*, which collects the votes, acts as a verifier of zero-knowledge proofs, enforces the rules of voting, and verifies the tally.

Adversary Model

The adversary $\mathcal{A}$ has bounded computing power, is unable to break used cryptographic primitives, and can control at most t of n participants during the protocol, where $t \leq n-2 \wedge n \geq 3$ (see the proof in the appendix of [VHLS20]). Any P under the control of $\mathcal{A}$ can misbehave during the protocol execution. $\mathcal{A}$ is also a passive listener of all communication entering the blockchain network but cannot block it or replace it with a malicious message since all transactions sent to the blockchain are authenticated by signatures of Ps or VA. Finally, VA **is only trusted in terms of identity management**, i.e., it performs identity verification of Ps honestly, and neither censor any P nor register any spoofed P. Nevertheless, no other trust in VA is required.

6.1.2 Overview of Proposed Approach

BBB-Voting scheme provides all properties mentioned in Section 6.0.1 but receipt-freeness. Similar to OVN [MSH17], BBB-Voting publishes the full tally at the output and uses homomorphic encryption to achieve privacy of votes and perfect ballot secrecy. In detail, we extend the protocol of Hao et al. [HRZ10] to support k choices utilizing the 1-out-of-k proof verification proposed by Kiayias and Yung [KY02], and we accommodate this approach to run on the blockchain. Additionally, we extend our protocol to support the robustness, based on Khader et al. [KSRH12], which enables the protocol to recover (without a restart) from faulty participants who did not submit their votes. As a consequence, robustness increases the resistance of our protocol to serious failures.

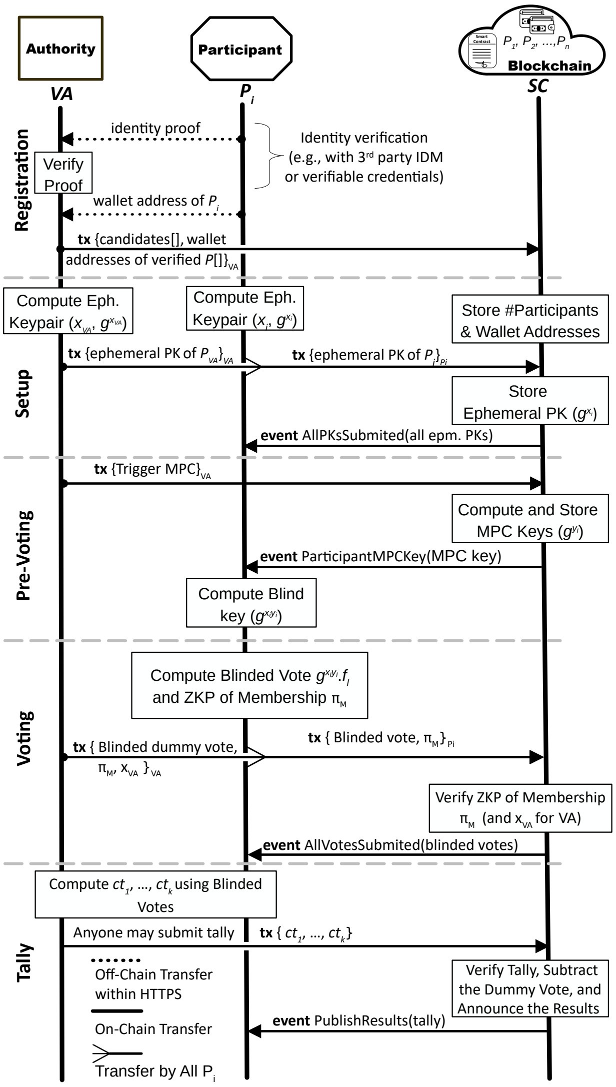

Authority
Participant
Smart Contract
$P_1, P_2, \ldots, P_n$
Blockchain
VA
P_i
SC
Registration
identity proof
Identity verification (e.g., with 3rd party IDM or verifiable credentials)
Verify Proof
wallet address of P_i
tx {candidates[], wallet addresses of verified P[]}$_{VA}$
Compute Eph. Keypair (x_{VA}, $g^{x_{VA}}$)
Compute Eph. Keypair (x_i, g^{x_i})
Store #Participants & Wallet Addresses
Setup
tx {ephemeral PK of P_{VA}}$_{VA}$
tx {ephemeral PK of P_i}$_{Pi}$
Store Ephemeral PK (g^{x_i})
event AllPKsSubmited(all epm. PKs)
tx {Trigger MPC}$_{VA}$
Pre-Voting
Compute and Store MPC Keys (g^{y_i})
event ParticipantMPCKey(MPC key)
Compute Blind key ($g^{x_i y_i}$)
Voting
Compute Blinded Vote $g^{x_i y_i}.f_l$ and ZKP of Membership π_M
tx { Blinded dummy vote, π_M, x_{VA} }$_{VA}$
tx { Blinded vote, π_M}$_{Pi}$
Verify ZKP of Membership π_M (and x_{VA} for VA)
event AllVotesSubmited(blinded votes)
Tally
Compute $ct_1, \ldots, ct_k$ using Blinded Votes
Anyone may submit tally
tx { $ct_1, \ldots, ct_k$}
Verify Tally, Subtract the Dummy Vote, and Announce the Results
event PublishResults(tally)
Off-Chain Transfer within HTTPS
On-Chain Transfer
Transfer by All P_i

Figure 6.1: Basic protocol of BBB-Voting.

6.1.3 Base Variant

The base variant of BBB-Voting (see Figure 6.1) does not involve a fault recovery and is divided into five stages: **registration** (identity verification, key ownership verification, enrollment at SC), a **setup** (an agreement on system parameters, submission of ephemeral public keys), **pre-voting** (computation of MPC keys), **voting** (vote packing, blinding, and verification), and **tally** phases. All faulty behaviors of Ps and VA are subject to deposit-based penalties. In detail, P who submitted her ephemeral key (in the setup phase) and then has not voted within the timeout will lose the deposit. To achieve fairness, VA acts as the last P who submits a "dummy" vote with her ephemeral private key[1] after all other Ps cast their vote (or upon the voting timeout expiration).

[1] Privacy for a dummy vote is not guaranteed since it is subtracted.

Phase 1 (Registration)

VA first verifies the identity proof of each P. For decentralized identity management (IDM), the identity proof is represented by the verifiable credentials (VC) [W3C19b] signed by the issuer, while in a centralized IDM the identity proof is interactively provided by a third-party identity provider (e.g., Google). First, VA verifies the issuer's signature on the identity proof. Next, VA challenges P to prove (using her VC) that she is indeed the owner of the identity. Further, each P creates her blockchain wallet address (i.e., the blockchain public key (PK)) and provides it to VA. The VA locally stores a bijective mapping between a P's identity and her wallet address.[2] Next, VA enrolls all verified Ps by sending their wallet addresses to SC.

Phase 2 (Setup)

Ps agree on system parameters that are universal to voting – the parameters for voting are publicly visible on SC (deployed by VA in a transaction). Therefore, any P may verify these parameters before joining the protocol. Note the deployment transaction also contains the specification of timeouts for all further phases of the protocol as well as deposit-based penalties for misbehavior of VA and Ps. The parameters for voting are set as follows:

1) VA selects a common generator $g \in \mathbb{F}_p^*$. The value of p is chosen to be a safe prime, i.e., $p = 2 \cdot q + 1$, where q is a prime. A safe prime is chosen to ensure the multiplicative group of order $p - 1 = 2 \cdot q$, which has no small subgroups that are trivial to detect.[3] Let $n < p - 1$.
2) Any participant P_i is later permitted to submit a vote $\{v_i \mid i \in \{1,2,...,k\}\}$ for one of k choices. This is achieved by selecting k independent generators $\{f_1,...,f_k\}$ in $\mathbb{F}_p^*$ (one for each choice). These generators for choices should meet a property described by Hao et al. [HRZ10] to preclude having two different valid tallies that fit Equation 6.4:

$$f_i = \begin{cases} g^{2^0} & \text{for choice 1,} \\ g^{2^m} & \text{for choice 2,} \\ \quad\dots & \\ g^{2^{(k-1)m}} & \text{for choice k,} \end{cases} \tag{6.1}$$

where m is the smallest integer such that $2^m > n$ (the number of participants).

Ephemeral Key Generation & Committing to Vote. Each P_i creates her ephemeral private key as a random number $x_i \in_R \mathbb{F}_p^*$ and ephemeral public key as g^{x_i}. Each P_i sends her ephemeral public key to SC in a transaction signed by her wallet, thereby, committing

[2] Note that the address of P must not be part of identity proof – avoiding VA to possess a proof of identity to blockchain address mapping (see Section 6.1.8).

[3] We use modular exponentiation by repeated squaring to compute $g^x \ mod \ p$, which has a time complexity of $\mathcal{O}((log \ x)\cdot(log^2 \ p))$ [Kob94].

to submit a vote later.[4] Furthermore, P_i sends a deposit in this transaction, which can be retrieved back after the end of voting. However, if P_i does not vote within a timeout (or does not participate in a fault recovery (see Section 6.1.4)), the deposit is lost, and it is split to the remaining involved parties. Ps who do not submit their ephemeral keys in this stage are indicating that they do not intend to vote; the protocol continues without them and they are not subject to penalties. Finally, each P obtains (from SC) the ephemeral public keys of all other verified Ps who have committed to voting. Ephemeral keys are one-time keys, and thus can be used only within one run of the protocol to ensure privacy of votes (other runs require fresh ephemeral keys).

Phase 3 (Pre-Voting)

This phase represents multiparty computation (MPC), which is run to synchronize the keys among all Ps and achieve the *self-tallying* property. However, no direct interaction among Ps is required since all ephemeral public keys are published at SC. The MPC keys are computed by SC, when VA triggers the compute operation via a transaction. The SC computes and stores the MPC key for each P_i as follows:

$$h = g^{y_i} = \prod_{j=1}^{i-1} g^{x_j} / \prod_{j=i+1}^{n} g^{x_j}, \tag{6.2}$$

where $y_i = \sum_{j<i} x_j - \sum_{j>i} x_j$ and $\sum_i x_i y_i = 0$ (see Hao et al. [HRZ10] for the proof). While anyone can compute g^{y_i}, to reveal y_i, all Ps $\setminus$ P_i must either collude or solve the DLP for Equation 6.2. As the corollary of Equation 6.2, the protocol preserves vote privacy if there are at least 3 Ps with at least 2 honest (see the proof in the extended version of our paper [VHLS20]).

Phase 4 (Voting)

In this phase, each P_i blinds and submits her vote to SC. These steps must ensure the recoverability of the tally, vote privacy, and well-formedness of the vote. Vote privacy is achieved by multiplying the P_i's blinding key with her vote choice. The blinded vote of the participant P_i is

$$B_i = \begin{cases} g^{x_i y_i} f_1 & \text{if } P_i \text{ votes for choice } 1, \\ g^{x_i y_i} f_2 & \text{if } P_i \text{ votes for choice } 2, \\ & \dots \\ g^{x_i y_i} f_k & \text{if } P_i \text{ votes for choice } k. \end{cases} \tag{6.3}$$

The P sends her choice within a blinded vote along with a 1-out-of-k non-interactive zero-knowledge (NIZK) proof of set membership to SC (i.e., proving that the vote choice $\in$

[4] In contrast to OVN [MSH17] (based on the idea from [HRZ10]), we do not require P_i to submit ZKP of knowledge of x_i to SC since P_i may only lose by submitting g^{x_i} to which she does not know x_i (i.e., a chance to vote + deposit).

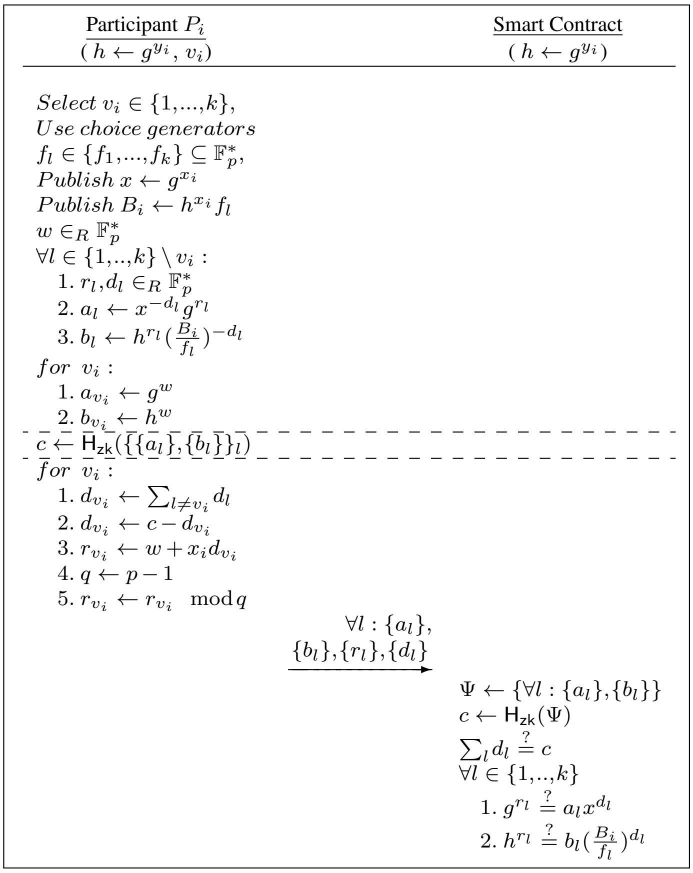

Participant P_i
$(h \leftarrow g^{y_i}, v_i)$
Smart Contract
$(h \leftarrow g^{y_i})$
$Select\ v_i \in \{1,...,k\}$,
$Use\ choice\ generators$
$f_l \in \{f_1,...,f_k\} \subseteq \mathbb{F}_p^*$,
$Publish\ x \leftarrow g^{x_i}$
$Publish\ B_i \leftarrow h^{x_i} f_l$
$w \in_R \mathbb{F}_p^*$
$\forall l \in \{1,..,k\} \setminus v_i :$
1. $r_l, d_l \in_R \mathbb{F}_p^*$
2. $a_l \leftarrow x^{-d_l} g^{r_l}$
3. $b_l \leftarrow h^{r_l} (\frac{B_i}{f_l})^{-d_l}$
$for\ \ v_i :$
1. $a_{v_i} \leftarrow g^w$
2. $b_{v_i} \leftarrow h^w$
$c \leftarrow \mathsf{H_{zk}}(\{\{a_l\},\{b_l\}\}_l)$
$for\ \ v_i :$
1. $d_{v_i} \leftarrow \sum_{l \neq v_i} d_l$
2. $d_{v_i} \leftarrow c - d_{v_i}$
3. $r_{v_i} \leftarrow w + x_i d_{v_i}$
4. $q \leftarrow p - 1$
5. $r_{v_i} \leftarrow r_{v_i} \mod q$
$\forall l : \{a_l\},$
$\{b_l\},\{r_l\},\{d_l\}$
$\Psi \leftarrow \{\forall l : \{a_l\},\{b_l\}\}$
$c \leftarrow \mathsf{H_{zk}}(\Psi)$
$\sum_l d_l \stackrel{?}{=} c$
$\forall l \in \{1,..,k\}$
1. $g^{r_l} \stackrel{?}{=} a_l x^{d_l}$
2. $h^{r_l} \stackrel{?}{=} b_l (\frac{B_i}{f_l})^{d_l}$

Figure 6.2: ZKP of set membership for 1-out-of-k choices.

$\{1,...,k\}$). We modified the approach proposed by Kiayias et al. [KY02] to the form used by Hao et al. [HRZ10], which is convenient for practical deployment on existing smart contract platforms. The verification of set membership using this protocol is depicted in Figure 6.2, where P_i is a prover and SC is the verifier. Hence, SC verifies the correctness of the proof and then stores the blinded vote. In this stage, it is important to ensure that no re-voting is possible, which is to avoid any inference about the final vote of P in the case she would change her vote choice during the voting stage. Such a re-voting logic can be enforced by SC, while user interface of the P should also not allow it. Moreover, to ensure fairness, VA acts as the last P who submits a dummy vote and her ephemeral private key.

Phase 5 (Tally)

When the voting finishes (i.e., voting timeout expires or all Ps and VA cast their votes), the tally of votes received for each of k choices is computed off-chain by any party and then submitted to SC. When SC receives the tally, it verifies whether Equation 6.4 holds, subtracts a dummy vote of VA, and notifies all Ps about the result. The tally is represented by vote counts $ct_i, \forall i \in \{1,...,k\}$ of each choice, which are computed using an

$\underline{Participant\ P_i}$		Smart Contract
$A \leftarrow g^{x_i}, B \leftarrow g^{x_j}, x_i$		$A \leftarrow g^{x_i},\ B \leftarrow g^{x_j}$
$Let\ w_i, \in_r \mathbb{F}_p$		
$C \leftarrow g^{x_i x_j}$		
$m_1 \leftarrow g^{w_i}$		
$m_2 \leftarrow B^{w_i}$		
$c \leftarrow \mathsf{H_{zk}}(A, B, m_1, m_2)$		
$r_i \leftarrow w_i + c x_i$		
	$\xrightarrow{C, r_i, m_1, m_2}$	$c \leftarrow \mathsf{H_{zk}}(A, B, m_1, m_2)$
		$g^r \overset{?}{=} m_1 A^c$
		$B^r \overset{?}{=} m_2 C^c$

Figure 6.3: ZKP verifying correspondence of $g^{x_i x_j}$ to public keys $A = g^{x_i}, B = g^{x_j}$.

exhaustive search fitting

$$\prod_{i=1}^{n} B_i = \prod_{i=1}^{n} g^{x_i y_i} f = g^{\sum_i x_i y_i} f = f_1{}^{ct_1} f_2{}^{ct_2} ... f_k{}^{ct_k}. \tag{6.4}$$

The maximum number of attempts is bounded by combinations with repetitions to $\binom{n+k-1}{k}$. Although the exhaustive search of 1-out-of-k voting is more computationally demanding in contrast to 1-out-of-2 voting [MSH17], [HRZ10], this process can be heavily parallelized. See time measurements in Section 6.1.6.

6.1.4 Variant with Robustness

We extend the base variant of BBB-Voting by a fault recovery mechanism. If one or more Ps stall (i.e., are faulty) and do not submit their blinded vote in the voting stage despite committing in doing so, the tally cannot be computed directly. To recover from faulty Ps, we adapt the solution proposed by Khader et al. [KSRH12], and we place the fault recovery phase immediately after the voting phase. All remaining honest Ps are expected to repair their vote by a transaction to SC, which contains key materials shared with all faulty Ps and their NIZK proof of correctness. SC verifies all key materials with proofs (see Figure 6.3), and then they are used to invert out the counter-party keys from a blinded vote of an honest P who sent the vote-repairing transaction to SC. Even if some of the honest (i.e., non-faulty) Ps would be faulty during the recovery phase (i.e., do not submit vote-repairing transaction), it is still possible to recover from such a state by repeating the next round of the fault recovery, but this time only with key materials related to new faulty Ps. To disincentivize stalling participants, they lose their deposits, which is split across remaining Ps as a compensation for the cost of fault recovery.

6.1.5 Implementation

We selected the Ethereum-based environment for evaluation due to its widespread adoption and open standardized architecture (driven by the Enterprise Ethereum Alliance

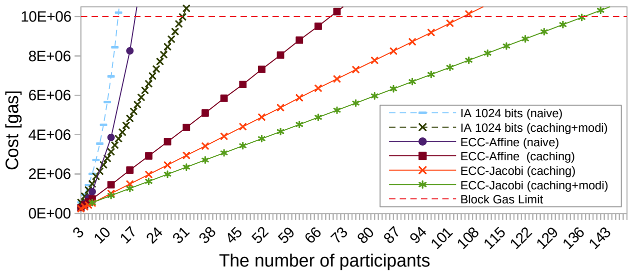
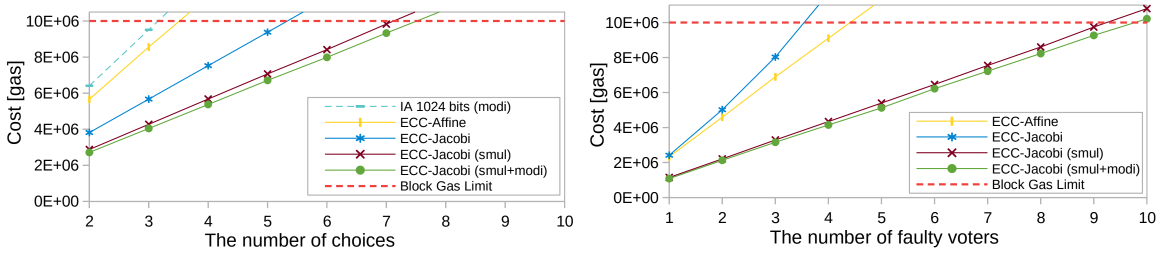


Figure 6.4: A computation of MPC keys by the authority.



(a) Vote submission by P_i.

(b) Vote repair by P_i.

Figure 6.5: Vote submission and vote repair (i.e., fault recovery) with various optimizations.

[ent17]), which is incorporated by many blockchain projects. We implemented SC components in Solidity, while VA and P components were implemented in Javascript. In this section, we analyze the costs imposed by our approach, perform a few optimizations, and compare the costs with OVN [MSH17]. In the context of this work, we assume 10M as the block gas limit. With block gas limit assumed, our implementation supports up to 135 participants (see Figure 6.4), up to 7 vote choices (see Figure 6.5a), and up to 9 simultaneously stalling faulty participants (see Figure 6.5b).[5]

We made two different implementations, the first one is based on DLP for integers modulo p (denoted as integer arithmetic (IA)), and the second one is based on the elliptic curve DLP (denoted as ECC). In the ECC, we used a standardized *Secp256k1* curve from existing libraries [Wit19], [MSH17]. In the case of IA, we used a dedicated library [Zer17] for operations with big numbers since EVM natively supports only 256-bit long words, which does not provide sufficient security level with respect to the DLP for integers modulo p.[6] We consider 1024 bits the minimal secure (library-supported) length of numbers in IA. As we will show below, IA implementation even with minimal 1024 bits is overly expensive, and thus in many cases does not fit the block gas limit by a single transaction. Therefore, in our experiments, we put emphasis on ECC implementation. The source code of our implementation is available at `https://github.com/ivan-homoliak-sutd/BBB-Voting`.

[5] The max. corresponds to a single recovery round but the total number of faulty participants can be unlimited since the fault recovery round can repeat.

[6] Since this DLP was already computed for 795-bit long safe prime in 2019 [BGG$^+$19], only values higher than 795-bit are considered secure enough.

6.1.6 Evaluation & Cost Optimizations

Since ECC operations in ZKP verifications and computation of MPC keys impose a high execution cost when running at the blockchain, we have made several cost optimizations.

(1) Caching in MPC Key Computation.

If implemented naively, the computation of all MPC keys in SC would contain a high number of overlapped additions, and hence the price would be excessively high (see series *"ECC-Affine (naive)"* in Figure 6.4).[7] Therefore, in the code of SC we accumulate and reuse the value of the left side of MPC key computation during iteration through all participants. Similarly, the sum at the right side can be computed when processing the first participant, and then in each follow-up iteration, it can be subtracted by one item. However, we found out that subtraction imposes non-negligible costs since it contains one affine transformation (which we later optimize). In the result, we found pre-computation of all intermediary right items in the expression during the processing of the first participant as the most optimal. The resulting savings are depicted as *"ECC-Affine (caching)"* series in Figure 6.4. We applied the same optimization for IA; however, even after adding a further optimization (i.e., pre-computation of modular inverses; see Section 6.1.6.4), the costs were still prohibitively high (see *"IA 1024 bits (caching+modi)"* series in Figure 6.4), with the max. number of participants fitting the gas limit only 29.

(2) Affine vs. Jacobi Coordinates.

In the ECC libraries employed [Wit19] [MSH17], by default, all operations are performed with points in Jacobi coordinates and then explicitly transformed to affine coordinates. However, such a transformation involves one modular inversion and four modular multiplications over 256-bit long integers, which is costly. Therefore, we maximized the utilization of internal functions from the Witnet library [Wit19], which do not perform affine transformation after operation execution but keep the result in Jacobi coordinates. This is possible only until the moment when two points are compared – a comparison requires affine coordinates. Hence, a few calls of the affine transformation are inevitable. This optimization is depicted in Figure 6.4 (series *"ECC-Jacobi (caching)"*) and Figure 6.5a (series *"ECC-Jacobi"*). In the case of computation of MPC keys, this optimization brought improvement of costs by 23% in contrast to the version with affine coordinates and caching enabled. Due to this optimization, up to 111 participants can be processed in a single transaction not exceeding the block gas limit. In the case of vote submission, this optimization brought improvement of costs by 33% in contrast to the version with affine coordinates.

[7] The same phenomenon occurs in IA (see Equation 6.2) but with overlapped multiplications (see series *"IA 1024 bits (naive)"*).

(3) Multiplication with Scalar Decomposition.

The most expensive operation on an elliptic curve is a scalar multiplication; based on our experiments, it is often 5x-10x more expensive than the point addition since it involves several point additions (and/or point doubling). The literature proposes several ways of optimizing the scalar multiplication, where one of the most significant ways is w-NAF (Non-Adjacent Form) scalar decomposition followed by two simultaneous multiplications with scalars of halved sizes [HMV05]. This approach was also adopted in the Witnet library [Wit19] that we base on. The library boosts the performance (and decreases costs) by computing the sum of two simultaneous multiplications $kP + lQ$, where $k = (k_1 + k_2\lambda)$, $l = (l_1 + l_2\lambda)$, and λ is a specific constant to the endomorphism of the elliptic curve. To use this approach, a scalar decomposition of k and l needs to be computed beforehand. Nevertheless, such **a scalar decomposition can be computed off-chain (and verified on-chain)**, while only a simultaneous multiplication is computed on-chain. However, to leverage the full potential of the doubled simultaneous multiplication, one must have the expression $kP + lQ$, which is often not the case. In our case, we modified the check at SC to fit this form. Alike the vote submission, this optimization can be applied in vote repair. We depict the performance improvement brought by this optimization as series *"ECC-Jacobi (smul)"* in Figure 6.5.

(4) Pre-Computation of Modular Inversions.

Each affine transformation in the vote submission contains one operation of modular inversion – assuming previous optimizations, ZKP verification of one item in 1-out-of-k ZKP requires three affine transformations (e.g., for $k = 5$, it is 15). Similarly, the ZKP verification of correctness in the repair vote requires two affine transformations per each faulty participant submitted. The modular inversion operation runs the extended Euclidean algorithm, which imposes non-negligible costs. However, all modular inversions **can be pre-computed off-chain, while only their verification can be made on-chain** (i.e., modular multiplication), which imposes much lower costs. We depict the impact of this optimization as *"ECC-Jacobi (smul+modi)"* series in Figure 6.5 and "...modi..." series in Figure 6.4. In the result, it has brought 5% savings of costs in contrast to the version with the simultaneous multiplication.

Tally Computation

In Table 6.1, we provide time measurements of tally computation through the entire search space on 1 core vs. all cores of the i7-10510U CPU laptop.[8] We see that for $n \leq 100$ and $k \leq 6$, the tally can be computed even on a commodity PC in a reasonable time. However, for higher n and k, we recommend using a more powerful machine or distributed computation across all Ps. One should realize that our measurements correspond to the upper bound, and if some ranges of tally frequencies are more likely than other ones, they can

[8] In some cases we estimated the time since we knew the number of attempts.

Voters	**Choices**			
(n)	$k = 2$	$k = 4$	$k = 6$	$k = 8$
20	$0.01s$	$0.01s$	$0.07s$	$0.07s$
30	$0.01s$	$0.01s$	$0.53s$	$13.3s$
40	$0.01s$	$0.04s$	$02.6s$	$112s$
50	$0.01s$	$0.08s$	$10.0s$	$606s$
60	$0.01s$	$0.16s$	$28.2s$	$2424s$
70	$0.01s$	$0.48s$	$69.6s$	$\sim 2.1h$
80	$0.01s$	$0.82s$	$160s$	$\sim 5.8h$
90	$0.01s$	$1.08s$	$320s$	$\sim 14.2h$
100	$0.01s$	$1.2s$	$722s$	$\sim 33h$

(a) 1 core

Voters	**Choices**	
(n)	$k = 6$	$k = 8$
20	$0.01s$	$0.01s$
30	$0.08s$	$2.0s$
40	$0.39s$	$16.8s$
50	$1.5s$	$90.9s$
60	$4.44s$	$267s$
70	$11.85s$	$773s$
80	$19.46s$	$2210s$
90	$44.02s$	$\sim 2.7h$
100	$108.3s$	$\sim 4.9h$

(b) 8 cores

Table 6.1: Upper time bound for tally computation.

be processed first – in this way, the computation time can be significantly reduced. Moreover, we emphasize that an exhaustive search for tally computation is not specific only to our scheme but to homomorphic-encryption-based schemes providing perfect ballot secrecy and privacy of votes (e.g., [KY02], [HRZ10], [MSH17]).

Cost Comparison

In Table 6.2, we made a cost comparison of BBB-Voting (using ECC) with OVN [MSH17], where we assumed two choices and 40 participants (the same setting as in [MSH17]). We see that the total costs are similar but BBB-Voting improves P's costs by 13.5% and VA's cost by 0.9% even though using more complex setting that allows 1-out-of-k voting. We also emphasize that the protocol used for vote casting in BBB-Voting **contains more operations** than OVN but regardless of it, the costs are close to those of OVN, which is mostly caused by the proposed optimizations.[9] Next, we found that OVN computes tally on-chain, which is an expensive option. In contrast, BBB-Voting computes tally off-chain and SC performs only verification of its correctness, which enables us to minimize the cost of this operation. Another gas saving optimization of BBB-Voting in contrast to OVN (and Hao et al. [HRZ10]) is that we do not require voters to submit ZKP of knowl-

[9] To verify 1-out-of-k ZKP in vote casting, BBB-Voting computes $5 \cdot k$ multiplications and $3 \cdot k$ additions on the elliptic curve – i.e., 10 multiplications and 6 additions for $k = 2$. In contrast, OVN computes only 8 multiplications and 5 additions for $k = 2$.

	Gas Paid by	**OVN**	**BBB-Voting**
Deployment of Voting SC	VA	3.78M	4.8M
Deployment of Cryptographic SCs	VA	2.44M	2.15M (1.22M+0.93M)
Enroll voters	VA	2.38M (2.15M+0.23M)	1.93M
Submit Ephemeral PK	P	0.76M	0.15M
Cast Vote	P	2.50M	2.72M
Tally	VA $(or\ P)$	0.75M	0.39M
Total Costs for P		3.26M	2.87M
Total Costs for VA		9.35M	9.27M

Table 6.2: A normalized cost comparison of BBB-Voting with OVN for $n = 40$ and $k = 2$.

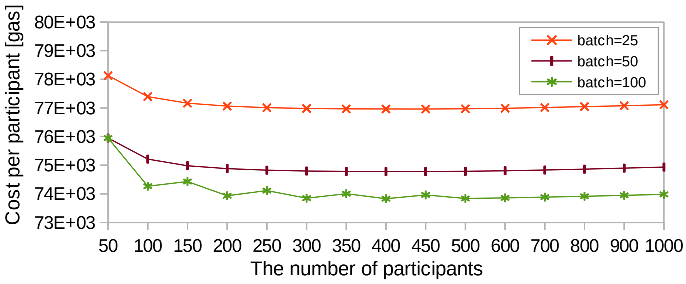


Figure 6.6: A computation of MPC keys by the authority VA using various batch sizes and the most optimized ECC variant.

edge of x_i in g^{x_i} during the registration phase to SC since P_i may only lose by providing incorrect ephemeral public key g^{x_i} – she might lose the chance to vote and her deposit. Finally, we note that we consider the deployment costs of our SC equal to 4.8M units of gas; however, our SC implementation contains a few auxiliary view-only functions for a pre-computation of modular inverses, with which, the deployment costs would increase to 7.67M due to code size. Nevertheless, these operations can be safely off-chained and we utilized them on-chain only for simplicity.

6.1.7 Limitations & Extensions

In this section, we discuss the extensions addressing the scalability and performance limitations of BBB-Voting.

Scalability Limitation & Extension

The limitation of BBB-Voting (like in OVN) is a lack of scalability, where the block gas limit might be exceeded with a high number of Ps. Therefore, we primarily position our solution as boardroom voting; however, we will show in this section that it can be extended even to larger voting. Our voting protocol (see Section 6.1.3) has a few platform-specific bottlenecks due to the block gas limit of Ethereum, when either k or n reach particular values. Nevertheless, transaction batching can be introduced for the elimination of all bottlenecks. To realize a batching of pre-voting, voting, and fault-recovery phases, the additional integrity preservation logic across batches needs to be addressed while the verification of integrity has to be made by SC. For demonstration purposes, we addressed the bottleneck of the pre-voting stage (see Figure 6.6) and setup stage, which further improves the vote privacy in BBB-Voting (see Section 6.1.8) and causes only a minimal cost increase due to the overhead of integrity preservation (i.e., 1%). With this extension, n is limited only by the expenses paid by VA to register Ps and compute their MPC keys, and the computing power to obtain the tally.[10] If a certain combination of high n and k would make computation of a tally overly computationally expensive (or the cost of its

[10] E.g., for $n = 1000$, $k = 2$ (and $k = 4$), it takes 0.15s (and $\sim$ 4h) to obtain the tally on a commodity PC with 8 cores, respectively.

verification by SC), it is further possible to partition Ps into multiple groups (i.e., voting booths), each managed by an instance of BBB-Voting, while the total results could be summed across instances. Scalability extension is a subject of our next work [SH23b], which will be described in Section 6.2.

Cost and Performance Limitations

Although we thoroughly optimized the costs of our implementation (see Section 6.1.6), the expenses imposed by a public permissionless smart contract platform might be still high, especially during peaks of the gas price and/or in the case of a larger voting than boardroom voting (see Section 6.1.7). Besides, the transactional throughput of such platforms might be too small for such larger voting instances to occur in a specified time window. Therefore, to further optimize the costs of BBB-Voting and its performance, it can run on a public permissioned Proof-of-Authority platforms, e.g., using Hyperledger projects (such as safety-favored Besu with Byzantine Fault Tolerant (BFT) protocol). Another option is to use smart contract platforms utilizing the trusted computing that off-chains expensive computations (e.g., Ekiden [CZK$^+$19b], TeeChain [LNE$^+$19], Aquareum [HS20]), or other partially-decentralized second layer solutions (e.g., Plasma [Eth22], Polygon Matic [Nat22], Hydra [CCF$^+$20]). Even though these solutions might preserve most of the blockchain features harnessed in e-voting, availability and decentralization might be decreased, which is the security/performance trade-off.

6.1.8 Security Analysis

We first analyze security of BBB-Voting with regard to the voting properties specified in Section 6.0.1. Next, we analyze blockchain-specific security & privacy issues and discuss their mitigations. Also, we compare voting and other features of BBB-Voting with a few related works in Table 6.3.

Properties of Voting

(1) Privacy in BBB-Voting requires at least 3 Ps, out of which at least 2 are honest (see Section 6.1.3). Privacy in BBB-voting is achieved by blinding votes using ElGamal encryption [Gam85], whose security is based on the decisional Diffie-Hellman assumption. Unlike the conventional ElGamal algorithm, a decryption operation is not required to unblind the votes. Instead, we rely on the self-tallying property of the voting protocol. The ciphertext representing a blinded vote is a tuple $(c_1{,}c_2)$, where $c_1 = g^{xy}.f$ and $c_2 = g^y$, where the purpose of c_2 is to assist with the decryption. Decryption involves computing $(c_2)^{-x} \cdot c_1$ to reveal f, which unambiguously identifies a vote choice. As a result, the blinding operation for participant P_i in Equation 6.3 is equivalent to ElGamal encryption involving the computation of c_1 but not the decryption component c_2. Furthermore, the blinding keys are ephemeral and used exactly once for encryption (i.e., blinding) of the

Approach	**Privacy of Votes**	**Perfect Ballot Secrecy**	**Fairness**	**Self-Tallying**	**Robustness**	**Uses Blockchain**	**Uni. Verifiability**	**E2E Verifiability**	**Open Source**	**Choices**
Hao et al. [HRZ10]	✓	✓	✗	✓	✗	✗	✓	✓	✗	2
Khader et. [KSRH12]	✓	✓	✓	✓	✓	✗	✓	✓	✗	2
Kiayias and Yung [KY02]	✓	✓	✓	✓	✓	✗	✓	✓	✗	$2/k$
McCorry et. [MSH17] (OVN)	✓	✓	✓	✓	✗	✓	✓	✓	✓	2
Seifelnasr et al. [SGY20] (sOVN)	✓	✓	✓	✓	✗	✓	✓	✓	✓	2
Li et al. [LSY+20]	✓	✓	✓	✓	✓	✓	✓	✓	✗	2
Baudron et al. [BFP+01]	✓	✗	✗	✗	✓	✗	✓	✗	✗	k
Groth [Gro04]	✓	✓	✓	✓	✗	✗	✓	✓	✗	k
Adida [Adi08] (Helios)	✓*	✗	✗	✗	✓	✗	✓	✓	✓	k
Matile et al. [MRSS19] (CaIV)	✓	✗	✗	✗	✓	✓	✗	✗	✓	k
Killer [KRS+20] (Provotum)	✓	✗	✓	✗	✓	✓	✓	✓	✓	2
Dagher et al. [DMMM18] (BroncoVote)	✓	✓*	✓*	✗	✗	✓	✗	✗	✗	k
Kostal et al. [KBRK19]	✓*	✓*	✗	✓	✓*	✓	✓	✓	✗	k
Zagorski et al. [ZCC+13] (Remotegrity)	✓*	✗	✗	✗	✓	✗	✗	✓	✓	k
Yu et al. [YLS+18]	✓	✓*	✓*	✓	✓*	✓	✓	✗	✗	k
BBB-Voting	✓	✓	✓	✓	✓	✓	✓	✓	✓	k

Table 6.3: A comparison of various remote voting protocols. *Assuming a trusted VA.

vote within a single run of voting protocol[11] – i.e., if the protocol is executed correctly, there are no two votes f_l and f_m encrypted with the same ephemeral blinding key of P_i, such that

$$\frac{(g^{xy} \cdot f_l)}{(g^{xy} \cdot f_m)} = \frac{f_l}{f_m}, \tag{6.5}$$

from which the individual votes could be deduced. For the blockchain-specific privacy analysis, see also Section 6.1.8.

(2) Ballot Secrecy. It is achieved by blinding the vote using ElGamal homomorphic encryption [CGS97], and it is not required to possess a private key to decrypt the tally because of the self-tallying property ($g^{\sum_i x_i y_i} = 1$). Therefore, given a homomorphic encryption function, it is possible to record a sequence of encrypted votes without being able to read the votes choices. However, if all Ps are involved in the recovery of a partial tally consisting of a recorded set of votes, these votes can be unblinded (as allowed by ballot secrecy). Even a subset of $n - 2$ Ps[12] who have already cast their votes cannot recover a partial tally that reveals their vote choices because of the self-tallying property ($g^{\sum_i x_i y_i} = 1$) has not been met.

(3) Fairness. If implemented naively, the last voting P can privately reveal the full tally by solving Equation 6.4 before she casts her vote since all remaining blinded votes are already recorded on the blockchain (a.k.a., the last participant conundrum). This can be resolved by VA who is required to submit the final dummy vote including the proof of her vote choice, which is later subtracted from the final tally by SC.[13]

[11] As a consequence, BBB-Voting can be utilized in a repetitive voting [VSH23] with a limitation of a single vote within an epoch.

[12] Note that at least 2 Ps are required to be honest (see Section 6.1.1).

[13] Note that If VA were not to execute this step, the fault recovery would exclude VA's share from MPC

(4a) Universal Verifiability. Any involved party can check whether all recorded votes in the blockchain are correct and are correctly included in the final tally [KY02]. Besides, the blinded votes are verified at SC, which provides correctness of its execution and public verifiability, relying on the honest majority of the consensus power in the blockchain.

(4b) E2E Verifiability. To satisfy E2E verifiability [PR18]: (I) each P can verify whether her vote was cast-as-intended and recorded-as-cast, (II) anyone can verify whether all votes are tallied-as-recorded. BBB-Voting meets (I) since each P can locally compute her vote choice (anytime) and compare it against the one recorded in the blockchain. BBB-Voting meets (II) since SC executes the code verifying that the submitted tally fits Equation 6.4 that embeds all recorded votes.

(5) Dispute-Freeness. Since the blockchain acts as a tamper-resistant bulletin board (see Section 6.1.8), and moreover it provides correctness of code execution (i.e., on-chain execution of verification checks for votes, tally, and fault recovery shares) and verifiability, the election remains dispute-free under the standard blockchain assumptions about the honest majority and waiting the time to finality.

(6) Self-Tallying. BBB-Voting meet this property since in the tally phase of our protocol (and anytime after), all cast votes are recorded in SC; therefore any party can use them to fit Equation 6.4, obtaining the final tally.

(7) Robustness (Fault Tolerance). BBB-Voting is robust since it enables to remove (even reoccurring) stalling Ps by its fault recovery mechanism (see Section 6.1.4). Removing of stalling Ps involves SC verifiability of ZKP submitted by Ps along with their counterparty shares corresponding to stalling Ps.

(8) Resistance to Serious Failures. The resistance of BBB-Voting to serious failures relies on the integrity and append-only features of the blockchain, which (under its assumptions Section 6.1.8) does not allow the change of already cast votes.

(9) Receipt-Freeness. BBB-Voting does not meet this property since any P can, using her ephemeral private key, recreate the blinded vote with the original vote choice, which can be compared to the recorded blinded vote. Moreover, P can reveal her ephemeral private key to the coercer, who can then verify how P voted.

(10) Dispute-Freeness. BBB-Voting provides dispute-freeness, since the all protocol stages of BBB-Voting are executed on the blockchain, and thus following of the protocol is self-enforcing.

Blockchain-Specific Aspects and Issues

In the following, we focus on the most important blockchain-specific aspects and issues related to BBB-Voting.

(1) Bulletin Board vs. Blockchains. The definition of a bulletin board [KY02] assumes its immutability and append-only feature, which can be provided by blockchains that

keys, and the protocol would continue.

moreover provide correct execution of code. CAP theorem [Bre00] enables a distributed system (such as the blockchain) to select either **c**onsistency or **a**vailability during the time of network **p**artitions. If the system selects consistency (e.g., Algorand [GHM$^+$17], BFT-based blockchains such as [Hyp17]), it stalls during network partitions and does not provide liveness (i.e., the blocks are not produced) but provides safety (i.e., all nodes agree on the same blocks when some are produced). On the other hand, if the system selects availability (e.g., Bitcoin [Nak09], Ethereum [Woo14b]), it does not provide safety but provides liveness, which translates into possibility of creating *accidental forks* and eventually accepting one as valid. Many public blockchains favor availability over consistency, and thus do not guarantee immediate immutability. Furthermore, blockchains might suffer from *malicious forks* that are longer than accidental forks and are expensive for the attacker. Usually, their goal is to execute double-spending or selfish mining [ES18], violating the assumptions of the consensus protocol employed – more than 51% / 66% of honest nodes presented in PoW / BFT-based protocols. To prevent accidental forks and mitigate malicious forks in liveness-favoring blockchains, it is recommended to wait for a certain number of blocks (a.k.a., block confirmations). Another option to cope with forks is to utilize safety-favoring blockchains (e.g., [GHM$^+$17, Hyp17]).

Considering BBB-Voting, we argue that these forks are not critical for the proposed protocol since any transaction can be resubmitted if it is not included in the blockchain after a fork. Waiting for the time to finality (with a potential resubmission) can be done as a background task of the client software at Ps' devices, so Ps do not have to wait. Finally, we emphasize that the time to finality is negligible in contrast to timeouts of the protocol phases; therefore, there is enough time to make an automatic resubmission if needed.

(2) Privacy of Votes. In BBB-Voting, the privacy of vote choices can be "violated" only in the case of unanimous voting by all Ps, assuming $\mathcal{A}$ who can link the identities of Ps (approximated by their IP addresses) to their blockchain addresses by passive monitoring of network traffic. However, this is the acceptable property in the class of voting protocols that provide the full tally of votes at the output, such as BBB-Voting and other protocols (e.g., [MSH17, KY02, HRZ10, KRS$^+$20, YLS$^+$18]). Moreover, $\mathcal{A}$ can do deductions about the probability of selecting a particular vote choice by Ps. For example, in the case that the majority m of all participants n voted for a winning vote choice, then $\mathcal{A}$ passively monitoring the network traffic can link the blockchain addresses of Ps to their identities (i.e., IP addresses), and thus $\mathcal{A}$ can infer that each P from the group of all Ps cast her vote to the winning choice with the probability equal to $\frac{m}{n} > 0.5$. However, it does not violate the privacy of votes and such an inferring is not possible solely from the data publicly stored at the blockchain since it stores only blinded votes and blockchain addresses of Ps, not the identities of Ps. To mitigate these issues, Ps can use anonymization networks or VPN services for sending transactions to the blockchain. Moreover, neither $\mathcal{A}$ nor VA can provide the public with the indisputable proof that links P's identity to her blockchain address.

(3) Privacy of Votes in Larger Voting. The privacy issue of unanimous and majority voting (assuming $\mathcal{A}$ with network monitoring capability) are less likely to occur in the larger voting than boardroom voting since the voting group of Ps is larger and potentially more divergent. We showed that BBB-Voting can be extended to such a large voting by integrity-preserving batching in Section 6.1.7. We experimented with batching up to 1000 Ps, which is a magnitude greater voting than the boardroom voting. We depict the gas expenses paid by VA (per P) in Figure 6.6, where we distinguish various batch sizes. In sum, the bigger the batch size, the lower the price per P.

6.2 SBvote

We introduce SBvote [SH23], a blockchain-based self-tallying e-voting protocol that enables scalability in the number of voters and is based on BBB-Voting protocol. SBvote introduces multiple voting smart contracts booths that are managed and aggregated by the main smart contract. Our extended solution maintains the most properties of decentralized e-voting, including public verifiability, perfect ballot secrecy, and fault tolerance (but excluding receipt-freeness). Moreover, it improves the privacy of voters within booths.

6.2.1 System Model

We focus on a decentralized e-voting that provides desired properties of e-voting schemes mentioned in Section 6.0.1 as well as scalability in the number of the participants. We assume a centralized authority that is responsible for the enrollment of the participants and shifting the stages of the protocol. However, the authority can neither change nor censor the votes of the participants, and it cannot compromise the privacy of the votes. We assume that a public bulletin board required for e-voting is instantiated by a blockchain platform that moreover supports the execution of smart contracts. We assume that all participants of voting have their thin clients that can verify the inclusion of their transactions in the blockchain as well as the correct execution of the smart contract code.

Adversary Model

We consider an adversary that passively listens to a communication on the blockchain network. The adversary cannot modify or replace any honest transactions since she does not hold the private keys of the participants. Next, we assume that the adversary cannot block an honest transaction due to the censorship-resistance property of the blockchain. The adversary can link a voter's IP address to her blockchain address. However, she does not possess the computational resources to break the cryptographic primitives used in the blockchain platform and the voting protocol. The adversary cannot access or compromise the voter's device or the user interface of the voting application. We assume that in each voting group of n participants, at most t of them can be controlled by the adversary and disobey the voting protocol, where $t \leq n - 2$ and $n \geq 3$. This eliminates the possibility of *full collusion* against a single voter [HRZ10].

6.2.2 Proposed Approach

Involved Parties. Our proposed approach has the following actors and components: (1) *a participant* $\mathbb{P}$ (*a voter*) who chooses a candidate (i.e., a voting choice) and casts a vote, (2) *a voting authority* $\mathbb{VA}$ responsible for the registration of participants and initiating actions performed by smart contracts, (3) *a booth contract* $\mathbb{BC}$, which is replicated into multiple instances, where each instance serves a limited number of participants. New instances might be added on-demand to provide scalability. (4) *The main contract* $\mathbb{MC}$, which assigns participants to voting booths, deploys booth contracts, and aggregates the final tally from booth contracts.

Protocol

We depict our protocol in Figure 6.7. SBvote follows similar phases as BBB-Voting but with several alterations that enable better scalability. The registration phase requires $\mathbb{VA}$ to authenticate users and generate a list of eligible voters. In BBB-Voting, the setup phase of the protocol allows users to submit their ephemeral public keys. However, in contrast to BBB-Voting, SBvote requires additional steps to set up the booth contracts. First, eligible voters are assigned to voting groups and then $\mathbb{BC}$ is deployed for each voting group. Once the setup is finished, voters proceed to submit their ephemeral public keys during a sign-up phase. These keys are further used to compute multi-party computation (MPC) keys within each voting group during a pre-voting phase. In the voting phase, voters cast their blinded votes along with corresponding NIZK proofs. The NIZK proof allows $\mathbb{BC}$ to verify that a blinded vote correctly encrypts one of the valid candidates. If some of the voters who submitted their ephemeral public keys have failed to cast their vote, the remaining active voters repair their votes in the subsequent fault recovery phase. This is achieved by removing the key material of stalling voters from the encryption of the correctly cast votes. The key material has to be provided by each active voter along with NIZK proof of correctness. After the repair of votes, the tallies for individual voting groups are computed during the tally phase of a booth. Then, partial tally results are aggregated to obtain the final tally by $\mathbb{MC}$.

In the following, we describe the phases of our protocol in more detail. Phases 2–6 are executed independently (thus in parallel) within each of the voting groups/booth contracts.

Registration. In this phase, the participants interact with $\mathbb{VA}$ to register as eligible voters. A suitable identity management (IDM) system is required, allowing $\mathbb{VA}$ to verify participants' identities and eligibility to vote.[14] Each participant creates her blockchain wallet address and registers it with $\mathbb{VA}$ that stores a mapping between a participant's identity and her wallet address.

[14] The details of IDM are out-of-scope for this work.

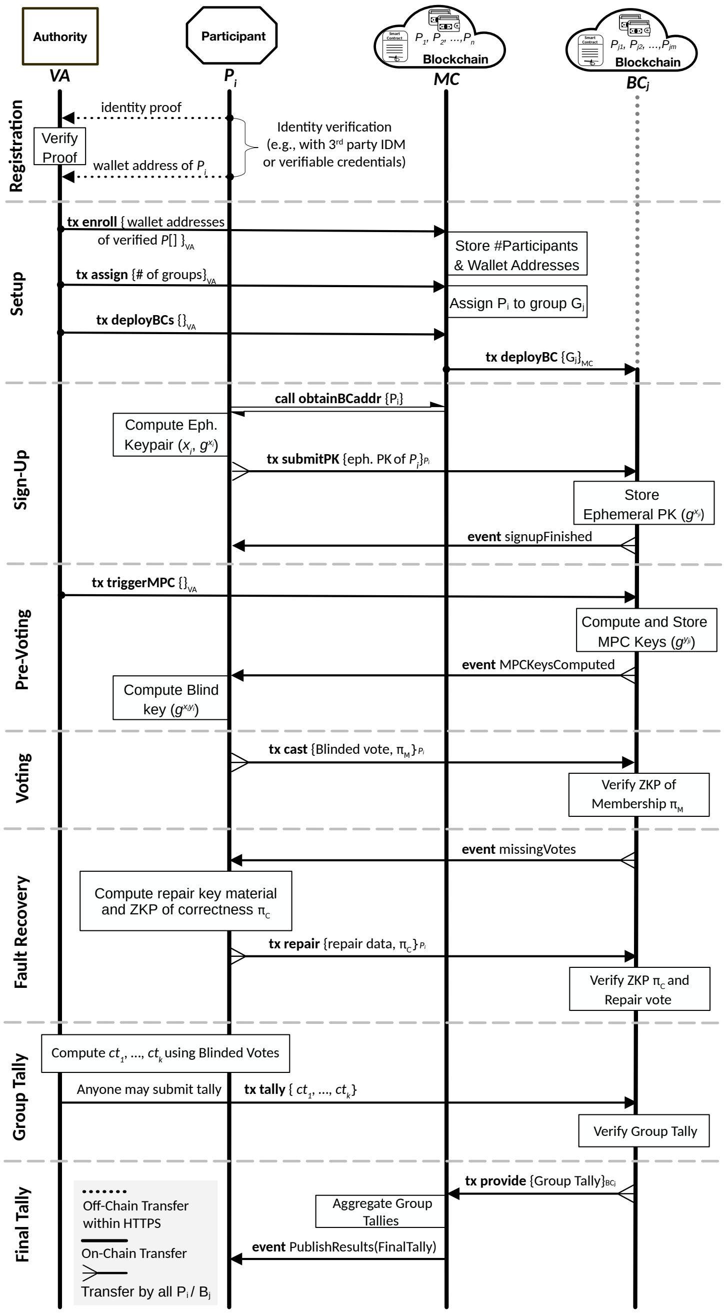

Authority
VA
Participant
Pi
P1, P2, ..., Pn
Blockchain
MC
Pj1, Pj2, ..., Pjm
Blockchain
BCj
Registration
identity proof
Verify Proof
wallet address of Pi
Identity verification (e.g., with 3rd party IDM or verifiable credentials)
Setup
tx enroll {wallet addresses of verified P[] }VA
Store #Participants & Wallet Addresses
tx assign {# of groups}VA
Assign Pi to group Gj
tx deployBCs {}VA
tx deployBC {Gj}MC
Sign-Up
call obtainBCaddr {Pi}
Compute Eph. Keypair (xi, g^xi)
tx submitPK {eph. PK of Pi}Pi
Store Ephemeral PK (g^xi)
event signupFinished
Pre-Voting
tx triggerMPC {}VA
Compute and Store MPC Keys (g^yi)
event MPCKeysComputed
Compute Blind key (g^xiyi)
Voting
tx cast {Blinded vote, πM}Pi
Verify ZKP of Membership πM
Fault Recovery
event missingVotes
Compute repair key material and ZKP of correctness πC
tx repair {repair data, πC}Pi
Verify ZKP πC and Repair vote
Group Tally
Compute ct1, ..., ctk using Blinded Votes
Anyone may submit tally
tx tally { ct1, ..., ctk}
Verify Group Tally
Final Tally
tx provide {Group Tally}BCj
Aggregate Group Tallies
event PublishResults(FinalTally)
Off-Chain Transfer within HTTPS
On-Chain Transfer
Transfer by all Pi / Bj

Figure 6.7: Overview of SBvote protocol.

Phase 1 (Setup). First, $\mathbb{VA}$ deploys $\mathbb{MC}$ to the blockchain. Then, $\mathbb{VA}$ enrolls the wallet addresses of all registered participants to $\mathbb{MC}$ within a transaction.[15] Once all the registered participants have been enrolled, $\mathbb{VA}$ triggers $\mathbb{MC}$ to pseudo-randomly distribute enrolled participants into groups whose size is pre-determined and ensures a certain degree of privacy. Note that distributed randomness protocols such as RoundHound [SJK$^+$17] might be used for this purpose, however, in this work we assume a trusted randomness

[15] Note that in practice this step utilizes transaction batching to cope with the limits of the blockchain platform (see Section 6.2.3).

source that is agreed upon by all voters (e.g., a hash of a Bitcoin block).

In every group, the participants agree on the parameters of the voting. Let n be the number of participants in the group and k the number of candidates. We specify the parameters of voting as follows:

1) a common generator $g \in \mathbb{F}_p^*$, where $p = 2 \cdot q + 1$, q is a prime and $n < p - 1$.
2) k independent generators $\{f_1,...,f_k\}$ in $\mathbb{F}_p^*$ such that $f_i = g^{2^{(i-1)m}}$, where m is the smallest integer such that $2^m > n$.

Then, $\mathbb{VA}$ deploys a booth contract $\mathbb{BC}$ for each group of participants with these previously agreed upon voting parameters. $\mathbb{MC}$ stores a mapping between a participant's wallet address and the group she was assigned to.

Phase 2 (Sign-Up). Eligible voters enrolled in the setup phase review the candidates and the voting parameters. Each voter who intends to participate obtains the address of $\mathbb{BC}$ she was assigned to by $\mathbb{MC}$. From this point onward, each participant interacts only with her $\mathbb{BC}$ representing the group she is part of. Every participant P_i creates her ephemeral key pair consisting of a private key $x_i \in_R \mathbb{F}_p^*$ and public key g^{x_i}. The P_i then sends her public key to $\mathbb{BC}$. By submitting an ephemeral public key, the participant commits to cast a vote later. Furthermore, participants are required to send a deposit within this transaction. If the voter does not cast her vote or later does not participate in the potential fault recovery phase, she will be penalized by losing the deposit. Voters who participate correctly retrieve their deposit at the end of the voting.

Phase 3 (Pre-Voting). In this step, each $\mathbb{BC}$ computes synchronized multi-party computation (MPC) keys from the participants' ephemeral public keys submitted in the previous step. To achieve scalability, the MPC keys are computed independently in each $\mathbb{BC}$ over the set of ephemeral public keys within the group. The MPC key for participant P_i is computed as follows:

$$g^{y_i} = \prod_{j=1}^{i-1} g^{x_j} / \prod_{j=i+1}^{n} g^{x_j}, \tag{6.6}$$

where $y_i = \sum_{j<i} x_j - \sum_{j>i} x_j$ and $\sum_i x_i y_i = 0$ (see Hao et al. [HRZ10] for the proof). The computation of MPC keys is triggered by $\mathbb{VA}$ in each $\mathbb{BC}$. After the computation, each participant obtains her MPC key from $\mathbb{BC}$ and proceeds to compute her ephemeral blinding key as $g^{x_i y_i}$ using her private key x_i.

Phase 4 (Voting). Before participating in this phase of the protocol, each voter must create her blinded vote and a NIZK proof of its correctness. The blinded vote of the participant P_i is $B_i = g^{x_i y_i} f_j$, where $f_j \in f_1,...,f_k$ represents her choice of a candidate. The participant casts the blinded vote by sending it to $\mathbb{BC}$ in a transaction $cast(B_i, \pi_M)$, where π_M is a 1-out-of-k NIZK proof of set membership. This proof allows $\mathbb{BC}$ to verify that the vote contains one of the candidate generators from $f_1,...,f_k$ without revealing the voter's choice. $\mathbb{BC}$ performs a check of the proof's correctness and accepts well-formed votes. Construction and verification of the NIZK proof are depicted in Figure 6.2 (from BBB-Voting).

Phase 5 (Fault-Recovery). The use of synchronized MPC keys ensures that a vote cast by each voter contains the key material shared with all voters within the group. If some of the voters within a group stall during the voting phase, the tally cannot be computed from the remaining data. Therefore, we include a fault-recovery phase, where remaining voters provide $\mathbb{BC}$ with the key material they share with each stalling voter, enabling $\mathbb{BC}$ to repair their votes. In detail, for a stalling voter P_j and an active voter P_i $(i \neq j)$, the shared key material $g^{x_i x_j}$ consists of the stalling voter's ephemeral public key g^{x_j} (previously published in $\mathbb{BC}$) and the active voter's ephemeral private key x_i. The active voters send the shared key material to $\mathbb{BC}$ along with a NIZK proof depicted in Figure 6.3 (from BBB-Voting). The NIZK proof allows $\mathbb{BC}$ to verify that the shared key material provided by the voter corresponds to the ephemeral public keys g^{x_i} and g^{x_j}.

Suppose some of the previously active voters become inactive during the fault-recovery phase (i.e., do not provide the shared key material needed to repair their votes). In that case, the fault-recovery phase can be repeated to exclude these voters. Note that this phase takes place in groups where all the voters who committed to vote during the sign-up phase have cast their votes.

Phase 6 (Booth Tallies). At first, the tally has to be computed for each group separately. Computation of the result is not performed by $\mathbb{BC}$ itself. Instead, $\mathbb{VA}$ (or any participant) obtains the blinded votes from $\mathbb{BC}$, computes the tally, and then sends the result back to $\mathbb{BC}$, which verifies whether a provided tally fits

$$\prod_{i=1}^{n} B_i = \prod_{i=1}^{n} g^{x_i y_i} f_j = g^{\sum_i x_i y_i} f_j = {f_1}^{ct_1} {f_2}^{ct_2} ... {f_k}^{ct_k}, \tag{6.7}$$

where $ct_j \in ct_1,...,ct_k$ denotes the vote count for each candidate.

Phase 7 (Final Tally). Once $\mathbb{BC}$ obtains a correctly computed tally, it sends it to $\mathbb{MC}$. $\mathbb{MC}$ collects and summarizes the partial tallies from individual booths and announces the final tally once all booths have provided their results. The participants can also review the partial results from already processed booths without waiting for the final tally since the booth tallies are processed independently.

6.2.3 Design Choices and Optimizations

We introduce several specific features of SBvote, which allow us to achieve the scalability and privacy properties.

Algorithm 5: Pre-computation of right side values from Equation 6.6.

```
Inputs:
  • $n$: # of voters
  • $mpc\_batch$ : batch size for MPC computation
  • $voterPKs$ : array of voters' ephemeral public keys
Outputs:
  • $right\_markers$ : pre-computed right side values

$right\_tmp \leftarrow 0$
if $n \bmod mpc\_batch \neq 0$ then
    $right\_markers$.push($right\_tmp$)
for $i \leftarrow 0$ to $n$ do
    if $n \bmod mpc\_batch = (i-1) \bmod mpc\_batch$ then
        $right\_markers$.push($right\_tmp$)
    $right\_tmp \leftarrow right\_tmp * voterPKs[n-i]$
```

Storage of Voters' Addresses. If we were to store the voters' wallet addresses in the booth contracts, it would cause high storage overhead and thus high costs. However, we proposed to store these addresses only in $\mathbb{MC}$, while booth contracts can only query $\mathbb{MC}$ whenever they require these addresses (i.e., when they verify whether a voter belongs to the booth's group). As a result, this eliminates the costs of transactions when deploying booth contracts, and moreover saving the blockchain storage space.

Elimination of Bottlenecks. The main focus of our proposed approach is to eliminate the bottlenecks that limit the number of voters and thus the size of the voting groups. In particular, passing the necessary data within a single transaction could potentially exceed the block gas limit.

The scalability of the Setup phase is straightforward to resolve since it does not involve any transient integrity violation checks (excluding duplicity checks). In all these cases, $\mathbb{VA}$ splits the data into multiple independent transactions. Similarly, each active voter can send the key material required to repair her vote in several batches in the Fault-Recovery phase, allowing the system to recover from an arbitrary number of stalling participants.

In contrast to the Setup and Fault-Recovery phases, batching in the Pre-Voting phase is not trivial since it requires transient preservation of integrity between consecutive batches of the particular voting group. Therefore, we designed a custom batching mechanism, which eliminates this bottleneck while also optimizing the cost of the MPC computation.

MPC Batching and Optimization. If computed independently for each participant, the computation of MPC keys leads to a high number of overlapping multiplications. Therefore, we optimize this step by dividing the computation into two parts, respecting both sides of the expression in Equation 6.6 and reusing accumulated values for each side.

First, we pre-compute the right part (i.e., divisor) of Equation 6.6, which consists of a product of ephemeral public keys of voters with a higher index than the current voter's one (i.e., i in Equation 6.6). The product is accumulated and saved in the contract's storage at regular intervals during a single iteration over all ephemeral public keys. The

Algorithm 6: Computation of a batch of MPC keys.

```
Inputs:
  • $voterPKs$ : array of voters' ephemeral public keys
  • $mpc\_batch$ : batch size for MPC computation
  • $start, end$ : start and end index of the current batch
  • $right\_marker$ : pre-computed right side value for the first index in a batch
  • $act\_left$ : left side value from the previous batch
Outputs:
  • $act\_left$ : left side value at the last index of the current batch
  • $mpc\_keys$ : array of MPC keys for the current batch

Compute right side values for the batch:
$right\_tab[mpc\_batch - 1] \leftarrow right\_marker$
for $i \leftarrow 0$ to $mpc\_batch$ do
    $j \leftarrow mpc\_batch - i$
    $right\_tab[j-1] \leftarrow right\_tab[j] * voterPKs[i-1]$
Compute the current batch of MPC keys:
for $i \leftarrow start$ to $end$ do
    $act\_left \leftarrow act\_left * voterPKs[i-1]$
$mpc\_keys[i] \leftarrow act\_left \div right\_tab[i] \bmod mpc\_batch$
```

size of these intervals corresponds to the batch size chosen for the computation of the remaining (left side) of the equation. We refer to these saved values as *right markers* (see Algorithm 5). We only choose to save the right markers in the storage of $\mathbb{BC}$ instead of saving all accumulated values due to the high cost of storing data in the smart contract storage. Though the intermediate values between right markers have to be computed again later, they are only kept in memory (not persistent between consecutive function calls). Therefore, they do not significantly impact the cost of the computation.

The second part of the computation is processed in batches. First, the right-side values for all voters within the current batch are obtained using the pre-computed right marker corresponding to this batch (see lines 1–4 of Algorithm 6). Then, the left part of Equation 6.6 is computed for each voter within the batch, followed by evaluating the entire equation to obtain the MPC key (lines 6–8 of Algorithm 6). This left-side value is not discarded; therefore, computing the left side for the next voter's MPC key only requires single multiplication. The last dividend value in the current batch is saved in the contract's storage to allow its reuse for the next batch.

6.2.4 Evaluation

To evaluate the scalability of SBvote, we created the proof of concept implementation that builds on BBB-Voting [HLS23]. We used the Truffle framework and Solidity programming language to implement the smart contract part and Javascript for the client API of all other components. We also utilized the Witnet library [Wit19] for on-chain elliptic curve operations on the standardized *Secp256k1* curve [SEC00]. Although Solidity was primarily intended for Ethereum and its Ethereum Virtual Machine (EVM), we have not selected Ethereum for our evaluation due to its high operational costs and low transactional throughput, which is contrary to our goal of improving scalability. However, there are many other smart contract platforms supporting Solidity and EVM, out of which we

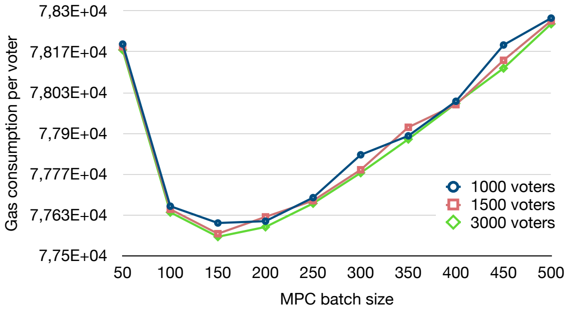


Figure 6.8: Per voter cost of the MPC key computation w.r.t. the batch size.

selected Gnosis[16] and Harmony[17] due to their low costs and high throughput.

Throughout our evaluation, we considered the following parameters of the chosen platforms: 30M block gas limit with 5 second block creation time on Gnosis and 80M block gas limit with 2 second block creation time on Harmony.

MPC Batch Size. The MPC keys in the Pre-Voting phase are computed in batches (see Section 6.2.3). In detail, there is a pre-computed value available for the first voter in each batch. Using a small batch size imposes many transactions and high execution costs due to utilizing fewer pre-computed values. In contrast, using a large batch size requires more expensive pre-computation and storage allocation, which results in a trade-off. This trade-off is illustrated in Figure 6.8, depicting how the batch size affects the cost of the computation per voter. We can see that the best value for our setup is 150 voters per batch.

The Number of Candidates. The number of candidates our voting system can accommodate remains limited. This is mainly caused by the block gas limit of a particular platform. In detail, we can only run voting with a candidate set small enough so that the vote-casting transaction does not exceed the underlying platform's block gas limit. Such transaction must be accompanied by a NIZK proof of set membership (i.e., proof that the voter's encrypted choice belongs to the set of candidates), and the size of the candidate

[16] https://developers.gnosischain.com

[17] https://www.harmony.one

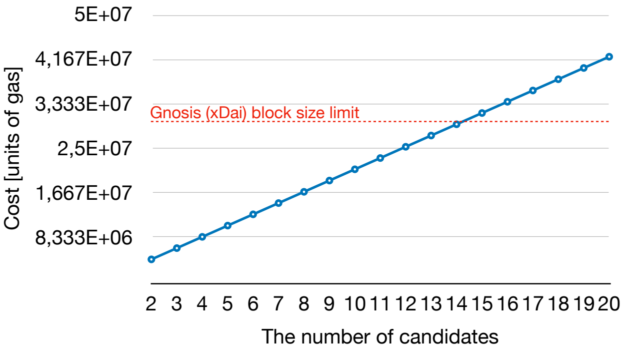


Figure 6.9: The cost of vote casting w.r.t. the number of candidates.

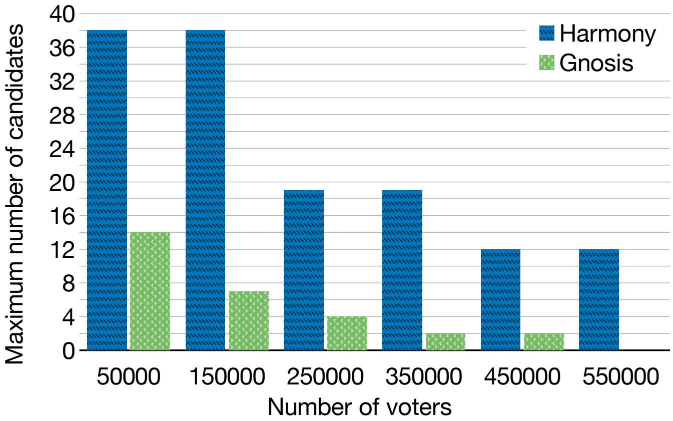
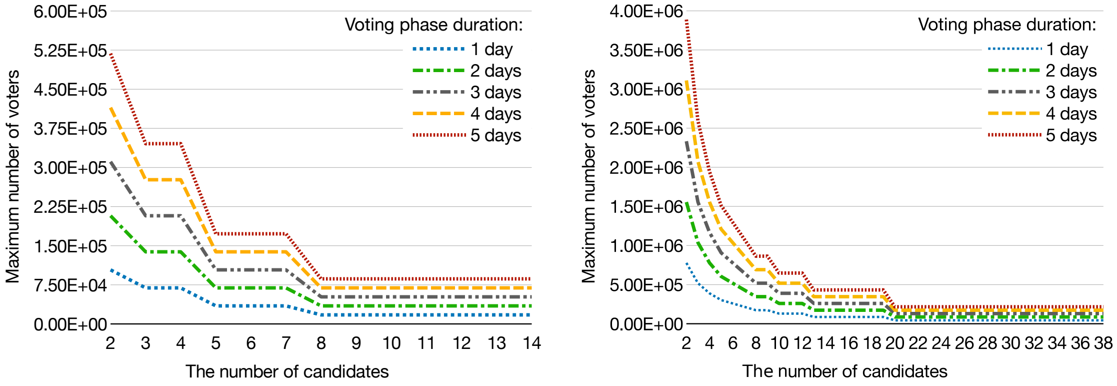


Figure 6.10: The maximum number of candidates our approach can process during a fixed 5-day voting interval, assuming various numbers of voting participants.



(a) Gnosis.

(b) Harmony.

Figure 6.11: The maximum number of voters that our approach can accommodate w.r.t. the number of candidates.

set determines its execution complexity. Figure 6.9 illustrates this dependency. Our experiments show that the proposed system can accommodate up to 38 and 14 candidates on Harmony and Gnosis, respectively.

The Total Number of Participants. The time period over which the voters can cast their ballots typically lasts only several days in realistic elections. The platform's throughput over a restricted time period and the high cost of the vote-casting transactions result in a trade-off between the number of voters and the number of candidates. We evaluated the limitations of the proposed voting protocol on both Harmony and Gnosis, as shown in Figure 6.10 and Figure 6.11. Note that in these examples, we considered only the most expensive phase of the protocol (i.e., voting phase) to be time-restricted.

We determined that with two candidates, the proposed system can accommodate 1.5M voters over a 2-day voting period and up to 3.8M voters over a 5-day voting period on the Harmony blockchain. On the other side of the trade-off, with the maximum number of 38 candidates on Harmony, maximally 216K voters can participate within a 5-day period.

6.2.5 Security Analysis and Discussion

We discuss the properties and scenarios affecting the security and privacy of SBvote.

Privacy. Within each voting group, SBvote maintains perfect ballot secrecy. The adversary, as defined in Section 6.2.1, cannot reveal a participant's vote through a collusion of all remaining participants since adversary can control at most $n - 2$ participants. The privacy of votes can be violated only if all participants in a voting group vote for the same candidate. However, this is a natural property of voting protocols, which output the tally rather than only the winning candidate. SBvote mitigates this problem by implementing transaction batching, which allows the authority to maintain a sufficiently large size of the voting groups to lower the probability of a unanimous vote within the groups. This probability is further decreased in SBvote by the smart-contract-based pseudo-random assignment of participants to the groups. We refer the reader to the work of Ullrich [Ull17] that addresses the issue of unanimous voting and the probability of its occurrence.

Deanonymization & Linking Addresses. In common blockchains, the network-level adversary might be able to link the participant's address with her IP address. Such an adversary can also intercept the participant's blinded vote; however, she cannot extract the vote choice due to the privacy-preserving feature of our voting protocol. Therefore, even if the adversary were to link the IP address to the participant's identity, the only information she could obtain is whether the participant has voted. Nevertheless, to prevent the linking of addresses, participants can use VPNs or anonymization services such as Tor.

Re-Voting. It is important to ensure that no re-voting is possible, which is to avoid any inference about the final vote of a participant in the case she would reuse her ephemeral blinding key to change her vote during the voting stage. Such a re-voting logic can be easily enforced by the smart contract, while the user interface of the participant should also not allow re-voting. Also, note that ephemeral keys are one-time keys and thus are intended to use only within one instance of e-voting protocol to ensure the security and privacy of the protocol. If a participant were to vote in a different instance of e-voting, she would generate new ephemeral keys.

Forks in Blockchain. Blockchains do not guarantee immediate immutability due to possible forks. This differentiates blockchains from public bulletin boards, as defined in [KY02]. However, since our protocol does not contain any two-phase commitment scheme with revealed secrets, its security is not impacted by accidental or malicious forks. Temporary forks also do not impact the voting stage since the same votes can be resubmitted by client interfaces.

Self-Tallying Property. The self-tallying property holds within each voting group since the correctness of obtained tallies can be verified by anybody. Consequently, this property holds for the whole voting protocol since the main contract aggregates the booth tallies of the groups in a verifiable fashion.

Verifiability. SBvote achieves both individual and universal verifiability. By querying the booth contract, each voter can verify her vote has been recorded. Each voter (and any interested party) can verify the booth tally since it satisfies the self-tallying property, i.e., the Equation 6.7 would not hold should any vote be left out. Any party can verify the final tally aggregated in the main contact by querying all the booth contracts to obtain individual booth tallies.

Platform-Dependent Limitations. Although our system itself does not limit the number of participants, the required transactions are computationally intensive, which results in high gas consumption. Therefore, large-scale voting using our system might be too demanding on the underlying smart contract platform. As a potential solution, public permissioned blockchains dedicated to e-voting might be utilized.

Adversary Controlling Multiple Participants in the Fault Recovery. One issue that needs to be addressed in the fault recovery is the adversary controlling multiple participants and letting them stall one by one in each fault recovery round. Even though the fault recovery mechanism will eventually finish with no new stalling participants, such behavior might increase the costs paid by remaining participants who are required to submit counter-party shares in each round of the protocol. For this reason, similar to the voting stage, we require the fault recovery stage to penalize stalling participants by losing the deposit they put into the smart contract at the beginning of our protocol. On the other hand, the adversary can cause a delay in the voting protocol within a particular booth. However, it does not impact other booths. To further disincentivize the adversary from such a behavior, the fault-recovery might require additional deposits that could be increased in each round, while all deposits could be redeemed at the tally stage.

Tally computation. Tallying the results in individual booths requires an exhaustive search for a solution of Equation 6.7 with $\binom{n+k-1}{k-1}$ possible combinations [HRZ10], where n is the number of votes and t is the number of candidates. Therefore, the authority should select the size of the voting groups accordingly to the budget and available computational resources (see Table 6.1 of BBB-Voting for the evaluation).

6.3 Always on Voting

Verifiable Delay Function (VDF). Given a time delay t, a VDF must satisfy the following conditions: for any input x, anyone equipped with commercial hardware can find y = VDF(x,t) in t sequential steps, but an adversary with p parallel processing units must not distinguish y from a random number in significantly fewer steps (see also Section 2.9). For our purposes, the value of t is fixed once it is determined. Therefore, we use VDF(x) instead of VDF(x,t) in the remaining text.

6.3.1 System Model

Our model has the following main actors and components: $i)$ A *participant* (P) who partakes in governance by casting a vote for her choice or candidate. $ii)$ *Election Authority* (EA) is responsible for validating the eligibility of participants to vote in elections, registering them, and shifting between the phases of the voting. $iii)$ A *smart contract* (SC) collects the votes, acts as a verifier of zero-knowledge proofs, enforces the rules of the voting and verifies the tallies of votes. $iv)$ *Bitcoin Puzzle Oracle* (BPO) provides an off-chain data feed from the Bitcoin network and supplies the requested Bitcoin block header (BH) when it is available on the Bitcoin network. $v)$ A *VDF prover* is any benign party in the voting ecosystem who computes the output of VDF and supplies proof of its correctness to SC.

Adversary Model

There are two types of adversaries: the network adversary Adv_{net} and a Bitcoin mining adversary Adv_{min}. Both adversaries are static and have bounded computing power, i.e., they are unable to break used cryptographic primitives under their standard security assumptions. Adv_{net} is a passive listener of communication entering the blockchain network but cannot block it. Her objective is to derive statistical inferences determining the voting patterns of participants (including the voting intervals in which they voted). Adv_{min} can mine on the Bitcoin blockchain. Her goal is to find a solution to the Bitcoin PoW puzzle that also triggers the end of the current voting interval, thereby influencing the end time of epoch. Our voting framework uses a function of the Bitcoin block header (BH) inclusive of its PoW solution s, i.e., $f(BH)$ to trigger the end of the current voting interval. Such a manipulation would potentially enable Adv_{min} to prematurely finish the current interval and start the next one. Finally, we assume that EA verifies identities honestly and supply addresses of only verified participants to SC.

6.3.2 Design Goals

The AoV framework has the following main design goals.

1. **Repeated voting epochs**: Participants are allowed to continuously vote and change elected candidates or policies without waiting for the next election. Participants are permitted to privately change their vote at any point in time, while the effect of their change is considered rightful at the end of each epoch. The duration of such epochs is shorter than the time between the two main elections.

2. **Randomized time epochs**: The end of each epoch is randomized and made unpredictable. In contrast to fixed-length time epochs, the proposed randomized time epochs are used to thwart the peak-end-effect.
3. **Plug & play voting protocols**: The AoV framework is designed to "plug & play" new or existing voting protocols. As a result, AoV inherits the properties of the underlying protocol chosen. However, in the interest of vote confidentiality on a blockchain, we recommend protocols providing secret ballots whose correctness can be publicly verified by SC without leaking any information, e.g., [YLS$^+$18, MRSS19, KRS$^+$20, BFP$^+$01]. Also, due to the repetitive nature of AoV, e-voting protocols with expensive on-chain computations and required fault recovery (due to stalling participants) may be less appealing but still acceptable with some limitations, e.g., [MSH17, LSY$^+$20, SGY20, VHLS21, SH23b].
4. **Privacy of participants**: Revoting by a participant P may enable Adv_{net} to link her blockchain wallet addresses or link the IP address of P across her multiple voting transactions. Therefore, it is important to achieve maximum voter privacy (anonymity) in the presence of adversary Adv_{net} (see Section 6.3.3).
5. **Privacy implications of booth sharding**: The AoV framework supports booth sharding to distribute voting overheads, streamline operations, and respect hierarchical structure of elections. In detail, instead of having a single booth smart contract for all participants, they are split into a number of smaller booth contracts. For multi-booth elections with limited participants per booth and a high candidate win probability, there is a small chance that all participants in a booth voted for the same candidate. Since the EA knows the identity of all the participants in the booth, it can trivially determine whom they voted for. We discuss the effects of booth size and candidate winning probability to prevent such incidents in Section 6.3.4.

6.3.3 Proposed Approach

Always-on-Voting (AoV) is a framework for a blockchain-based e-voting, in which voting does not end when the votes are tallied and the winners are announced. Instead, participants can continue voting for their previous vote choice or change their vote. A possible outcome of such repetitive voting is transitioning from a previous winning candidate to a new winner. To achieve this, the whole time interval between two regularly scheduled elections is unpredictably divided into several intervals, denoted as voting epochs. Participants may change their vote anytime before the end of a voting epoch (i.e., before a tally of the epoch is computed); however, they do not know beforehand when the end occurs. Any vote choice that transitioned into the supermajority threshold of votes is declared as the new winner of the election, and it remains a winning choice until another vote choice reaches a supermajority threshold.

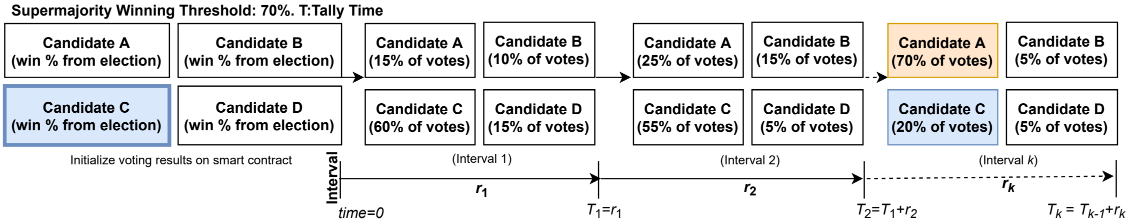

Supermajority Winning Threshold: 70%. T:Tally Time
Candidate A (win % from election)
Candidate B (win % from election)
Candidate C (win % from election)
Candidate D (win % from election)
Initialize voting results on smart contract
Candidate A (15% of votes)
Candidate B (10% of votes)
Candidate C (60% of votes)
Candidate D (15% of votes)
Candidate A (25% of votes)
Candidate B (15% of votes)
Candidate C (55% of votes)
Candidate D (5% of votes)
Candidate A (70% of votes)
Candidate B (5% of votes)
Candidate C (20% of votes)
Candidate D (5% of votes)
Interval
(Interval 1)
(Interval 2)
(Interval k)
r_1
r_2
r_k
time=0
$T_1=r_1$
$T_2=T_1+r_2$
$T_k = T_{k-1}+r_k$

Figure 6.12: The time between two regular elections is divided into the fixed number ft of intervals (a.k.a., epochs). First, the ratios of votes for all vote choices (i.e., candidates) are initialized from the last election. Next, repeated voting within k epochs results in a winning vote choice transition (from C to A). The new winner A is declared when she obtains a supermajority of total votes (i.e., 70%) at interval k; $k \leq ft$ (see Section 6.3.3). Note that r_1,..., r_k are randomized times that determine the length of the intervals 1,...,k. The tally is computed at the end of each interval.

Underlying Voting Protocol

AoV provides the option to plug & play any suitable e-voting protocol. To provide the baseline security and privacy of votes (with on-chain verifiability), we assume the voting protocol plugged into AoV allows participants to blind or encrypt their votes whose correctness is verified on-chain by SC. However, AoV does not deal with other features supported by the plugged-in voting protocol (such as end-to-end verifiability [JMP13], coercion-resistance [YLS$^+$18], receipt-freeness [KY02], and fairness [KY02]).

Example of Operation

Figure 6.12 illustrates a scenario with 4 candidates A-D, where C is the present winner of the election. For example, the supermajority threshold of 70% votes is set for future winnings, which is a tunable parameter that may be suitably tailored to the situation. All candidates are initialized to their winning percentages of obtained votes from the last election. Over time, the individual tally is observed to shift as the supermajority of participants decided to change their vote in favor of another candidate by voting in the epoch intervals. Through k intervals, the winner-ship is seen to transition from candidate C to A. At the k^{th} interval, A obtains the 70% threshold of votes and is declared as the new winner. Note that the supermajority is required only in the voting epochs between two regularly scheduled elections. The regular elections are also executed in AoV, and they repeat every M months/years, while requiring only a majority of votes (i.e., >50%) to declare a winner. Hence, in contrast to existing electoral systems, we only propose changes between regularly scheduled elections and enable new candidates to be added or removed.

Justification for Supermajority. A supermajority of 70% was chosen (see Appendix of [VSH23] for background) to help the incumbent carry out reforms without the risk of losing when there is still sufficient support from participants. On the other hand, the main purpose of this threshold is to block (or repeal) policies that are unpopular or negatively affecting a vast majority of participants. Additionally, we aim to avoid the quorum para-

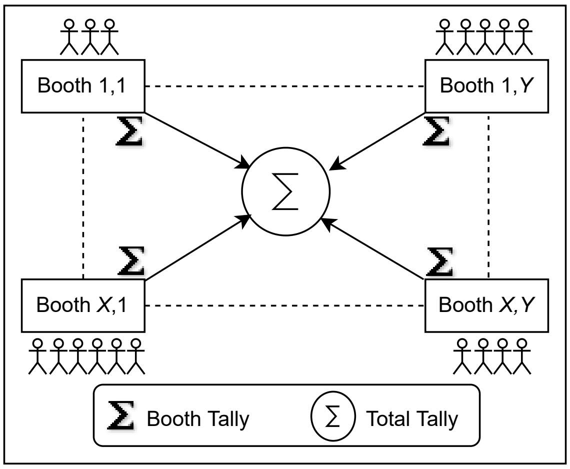


Figure 6.13: When the tally computation is triggered, each booth computes the sum of all votes cast at the booth (referred to as booth tally). Each booth tally is further summed up to determine the total tally. Pictorially, the booths are numbered 1 to X along the rows and 1 to Y along the columns. There are a total of $X \cdot Y$ booths.

dox (see Appendix of [VSH23]) by setting a minimum participation requirement of 70% from the just concluded main election.

Overview of AoV Phases

Once the setup phase (that ensures participants agree upon all system parameters) is completed, electronic voting frameworks typically consist of three phases: (1) a registration phase to verify voter credentials and add them to the voting system, (2) a voting phase, in which participants cast their vote via a secret ballot, and (3) a tally phase, where the total votes for each candidate are counted and revealed to participants. The voting protocol plugged-in with the AoV framework may contain additional phases, but we omit them here for brevity.

The architecture of AoV is shown in Figure 6.14. In AoV, participants (in step 1) register their wallet address with the EA, who then (in step 2) verifies and updates it on the booth smart contract[18]. This is followed by the voting phase (in step 3), where participants publicly cast their secret ballots (i.e, not revealing the vote choice nor identity). The BPO (step 4) supplies the validator contract and VDF prover with the $target$, recent Bitcoin block header BH and its block height. The VDF prover[19] (in step 5) computes and submits $VDF(BH)$ and a proof of sequential work (π) to the validator contract. The validator contract (in step 6) verifies the VDF proof and checks whether the supplied nonce s (included in the block header) is a valid solution to the Bitcoin PoW puzzle of the supplied header. If both verifications pass, the validator contract finalizes the epoch

[18] Participants are randomly grouped and assigned to booths $\in \{1,2,...,X \cdot Y\}$ (see Figure 6.13), represented by a booth smart contract.

[19] A VDF prover is any benign user in the voting ecosystem with commercial hardware to evaluate the input of VDF, i.e., y = VDF(BH) and supply a proof π.

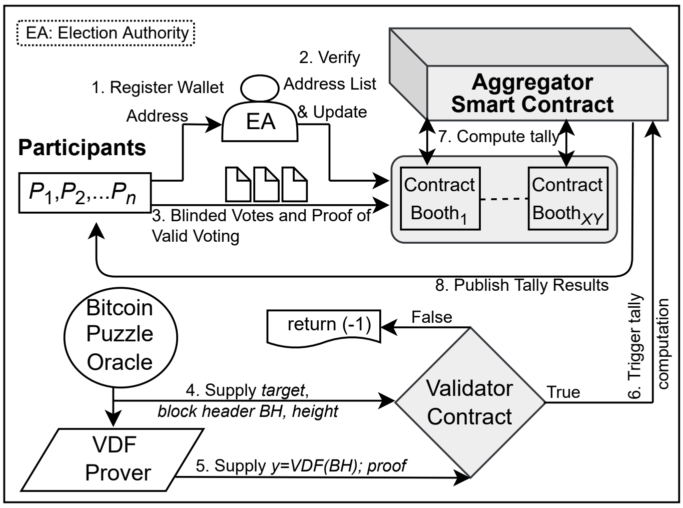

EA: Election Authority
1. Register Wallet Address
2. Verify Address List & Update
EA
Aggregator Smart Contract
Participants
P_1,P_2,...P_n
7. Compute tally
Contract Booth$_1$
Contract Booth$_{XY}$
3. Blinded Votes and Proof of Valid Voting
8. Publish Tally Results
Bitcoin Puzzle Oracle
return (-1)
False
Validator Contract
True
6. Trigger tally computation
4. Supply *target*, *block header BH, height*
VDF Prover
5. Supply *y=VDF(BH); proof*

Figure 6.14: Interaction among participants (Ps), election authority (EA), smart contracts, the Oracle, and VDF prover. (1) Registering wallet addresses of participants and (2) their identity verification are made by the EA. (3) Participants send a blinded vote and its zero-knowledge proof of correctness to their assigned booth contract. The booth contract verifies the validity of the vote. (4) The Bitcoin Puzzle Oracle (BPO) provides the latest Bitcoin block header (BH) and (5) VDF prover sends a proof of sequential work with y (the output of VDF(BH)) to the validator contract. (6) The validator contract finishes the epoch and shifts the state of the elections to the tally upon meeting the required conditions. (7) The aggregator contract is responsible for totaling individual booth tallies and (8) publicly announcing the total tally. The on-chain components of AoV are depicted in gray.

and triggers the tally computation for the epoch. Otherwise, it waits for the next block header submission from the BPO and the proof of sequential work from the VDF prover. When the tally computation is triggered, each booth contract $\{1{,}2{,}...{,}X \cdot Y\}$, sums up all its local vote counts and sends them to the aggregator contract (step 7). Then, the aggregator contract totals the votes from each booth contract and publishes the final tally (step 8). In AoV, the EA is authorized to register/remove participants and candidates in a future interval. Nevertheless, candidates can also be managed by other means, and AoV does not mandate how it should be done. When there are no other changes in the next interval, revoting repeats with step 2 and ends with step 8. From the initialization of AoV until the next regular elections, the validator smart contract accepts all future Bitcoin block headers. The new block headers (as part of their blocks) arriving every 10 minutes on average are appended to the Bitcoin blockchain. The BPO is responsible for timely supplying[20] each new block header to the VDF prover and validator contract. The VDF prover computes the VDF on each of those block headers after they are supplied.

[20] To respect the finality of the Bitcoin network, we assume that BPO supplies only the block headers that contain at least 6 confirmations on top of them. As a consequence, the probability that such a confirmed block will be reverted is negligible. Note that this does not influence the chances of ADV_{min} to succeed since she is already “delayed” by VDF in finding multiple PoW solutions at the same height; therefore, she prefers to work on top of the chain with her new attempts.

Calculating the Epoch Tally Time

Due to concerns that Bitcoin nonces are a weak entropy source, additional steps are taken to make it cryptographically secure (see details in Section 6.3.4). Our notion of randomness relies on Bitcoin Proof-of-Work to generate valid nonces.[21] The validator contract awaits future block headers yet to be mined on the Bitcoin network. When new Bitcoin block headers arrive, they are sent to the validator contract and the VDF prover via the BPO. The VDF ensures that a mining adversary cannot find more than one valid nonce to the block at a given height and test if the nonce is favorable within 10 minutes. The VDF is computed with the block header at the input by the VDF prover, who then submits the VDF output and the proof of sequential work to the validator contract. The choice of VDF depends on its security properties, speed of verification, and a size of the proof [Pie19]. Let BH be the Bitcoin block header. Once VDF prover computes $y = VDF(BH)$, a small proof (π) is used to trivially verify its correctness using $VDF\_Verify(y,\pi)$. Wesolowski's construction [Wes20] is known for its fast verification and a short proof: Let TL be the number of sequential computations. Prover claims

$$y \ = \ BH^{2^{TL}}$$

and computes a proof

$$\pi = BH^{\lfloor \frac{2^{TL}}{B} \rfloor},$$

where B = Blake256(BH || y || TL) hash. Verifier checks whether

$$\pi^B \cdot BH^{2^{TL} mod\ B} \stackrel{?}{=} y.$$

Since we employ VDF, Adv_{min} does not know the value of y before evaluating the VDF and is forced to wait for a given amount of time to see if the output is in her favor (before trying again). However, since Bitcoin mining is a lottery, other miners can solve the puzzle and append a block by propagating the solution to the Bitcoin network, rendering any withheld or attempted solution by the adversary that was not published useless.

Interactions of BPO, VDF Prover, and Validator. Let $TotalTime$ be the time in minutes between 2 regular elections. The BPO (see step 4 in Figure 6.14) feeds the block header BH of every future Bitcoin block (when it is available) to the validator contract and VDF prover. Further, BPO provides validator contrast also with the value of $target$ when it changes; i.e., every 2016 blocks.

Upon obtaining data from BPO, the VDF prover computes VDF output

$$y \ = \ VDF(BH) \tag{6.8}$$

with the VDF proof π and sends them to the validator contract (see step 5 in Figure 6.14). Next, the validator contract verifies the following conditions:

$$VDF\_Verify(y,\pi) \stackrel{?}{=} True, \tag{6.9}$$

[21] If the nonce overflows, a parameter called *extraNonce* (part of the coinbase transaction) is used to provide miners with the extra entropy needed to solve the PoW puzzle.

$$SHA256(BH) < target. \tag{6.10}$$

The first verification checks whether the VDF output y and supplied proof π (i.e., Equation 6.9) correspond to the BPO-supplied block header BH. The second verification (i.e., Equation 6.10) checks whether the nonce received from BPO[22] is a valid solution to the Bitcoin PoW puzzle. Once both checks pass, the validator contract proceeds to compute

$$a = SHA(y), \tag{6.11}$$

where SHA(.) is SHA-X-256[23] hash. The goal of Equation 6.11 is to consolidate the entropy by passing it through a compression function that acts as a randomness extractor (see Section 6.3.4). Using a, the validator contract computes

$$b = a \ (mod \ BHsInInterval), \tag{6.12}$$

where the expected number of block headers are

$$BHsInInterval \ = \ \frac{IntervalTime}{BlockTime}. \tag{6.13}$$

and the time interval is found as

$$IntervalTime \ = \ \frac{TotalTime}{ft}. \tag{6.14}$$

As seen in Figure 6.12, ft is the number of intervals (epochs) that the total time ($TotalTime$) between 2 regular elections is divided into. $BlockTime$ is the average time of block generation (i.e., 10 minutes in Bitcoin). The computation of tally for the current interval is triggered when the output of the validator contract is $True$ (see step 6 in Figure 6.14):

$$VC_{output} = \begin{cases} \text{True}, & \text{if } b = 0 \\ \text{False}, & \text{otherwise.} \end{cases} \tag{6.15}$$

Example. Let $TotalTime$ = 4 years = $525600 \cdot 4$ minutes and $ft = 8$; then $IntervalTime$ = $(525600 \cdot 4)/(8) = 262800$ minutes ≈ 182.5 days and $BHsInInterval$ = $262800/10$ = 26280 blocks. Therefore, the BPO will send on average 26280 block headers (BH values) to the validator contract within assumed 182.5 days long epoch (assuming 10 minutes block creation interval), i.e., $1/8$ of the total time. We expect the tally will be triggered on average once in every 182.5 days because of the Poisson probability distribution of this event. Therefore, ft expresses the expected number of epochs, while ft might differ across the regular elections iterations.

[22] We note that the BPO may be replaced by a quorum to improve decentralization. The validator contract will then accept the input from BPO only when $2/3$ (and more) of the quorum is in agreement.

[23] X denotes a suitable hash function such as SHA-3, and 256 is the output length in bits.

Anonymizing Identity

In our scheme, we employ wallet addresses to keep track of authorized participants. The map between participant P and her wallet, recorded by the EA is to prevent sybil attack (preventing any unauthorized person from voting) and double voting. For this reason, EA is trusted to keep this mapping private. Only voters corresponding to white-listed wallets by EA are allowed to vote and everyone else is blocked from voting on the smart contract. The wallet address corresponds to a unique random string associated with the voter. Its knowledge provides no additional information to the EA, since the EA knows voters' identities.

The AoV framework permits participants to change their vote at any time. The effect of the change is manifested at the end of the epoch when the tally is computed. However, Adv_{net} can observe the vote transactions on SC even though the vote choice remains confidential (preserving the privacy of votes). A participant might vote in one interval and then vote again in a future interval. Adv_{net} cannot distinguish whether the participant voted again for the same candidate or changed her vote to a different candidate due to assumed confidentiality-preserving properties of plugged voting protocols (see Section 6.3.3) However, both votes may be mapped to the same participant's blockchain wallet address if utilized naively, hence Adv_{net} can determine how many times a participant voted.

To break the map between participant P and her blockchain wallet address, we use the idea presented in type 2 deterministic wallets[24]. The objective is to synchronize a practically unlimited number of wallet public keys PKs (one per vote) between EA and P such that this PK list can be regenerated only by these two parties, while the corresponding private keys SKs can be computed only by P. As mentioned in Section 6.3.1, EA is assumed to verify identities honestly, and it supplies their corresponding wallet addresses to SC. The wallet address is generated as a function f of the elliptic curve public key. Once the public key is available, it is straightforward to compute the corresponding wallet address. Let BP be the base point on the elliptic curve. Further, let PK be the blockchain wallet public key corresponding to a private key SK. Here, SK is chosen as a random positive integer whose size is bounded to the order of BP on the chosen elliptic curve modulo a prime number. Note that Equation 6.16 – Equation 6.20 are computed off-chain. As an illustration, let the first PK be computed as

$$PK_0 = SK_0 \cdot BP, \tag{6.16}$$

and the next PK be

$$PK_1 = PK_0 + SK_1 \cdot BP. \tag{6.17}$$

From Equation 6.16 and Equation 6.17, we observe that

$$(SK_0 + SK_1) \cdot BP = PK_1. \tag{6.18}$$

The following steps ensue:

[24] See `https://en.bitcoin.it/wiki/BIP_0032`.

Algorithm 7: VDF Add.

```
Def VdfAdd ($y$,$\pi$,$blockheight$):
    writeState(“$vdfadd$” || $blockheight$,$y$ || $\pi$)
```

Algorithm 8: BPO Add.

```
Def BpoAdd ($Target$,$BH$,$blockheight$):
    writeState(“$bpoadd$” || $blockheight$,$BH$ || $Target$)
    writeState(“$blockheightStored$”,$blockheight$)
    writeState(“$blockheader$” || $blockheight$,$BH$)
```

i) During the identity verification, P sends to EA: a) wallet public key PK_0, b) a random shared secret key hk, and c) parameters (g,p), where g is a randomly chosen generator in F_p^* (i.e., a prime field) and p is a large prime. The wallet address is a public function of the wallet public key. Hence, EA computes $W_0 = f(PK_0)$ and stores it.

ii) The private key of P at any future voting epoch $e = \{1,2,3,...,2^{128}-1\}$ is generated by P as

$$SK_e = SK_0 + HMAC_{hk}(g^e), \tag{6.19}$$

where $g^e \in F_p^*$ is the output of pseudo-random number generator (PRNG) in epoch e, HMAC$(.)$ is HMAC-X-256 using shared secret key hk between EA and P, which is unknown to Adv_{net} and serves for stopping her from mapping P's wallet addresses.

iii) The corresponding PK of P for epoch e is

$$PK_e = PK_0 + HMAC_{hk}(g^e) \cdot BP. \tag{6.20}$$

EA and P can compute PK_e but SK_e is held only by P. This effectively separates PKs from their SKs and, at the same time, maps it to P's first wallet public key, i.e., PK_0. At any voting iteration e, the public key PK_e, and the corresponding wallet address can be computed by both EA and P. Since the shared secret hk used with HMAC is known only to EA and P, no third party, including Adv_{net}, is able to compute any future PKs. Hence, for a sequence of wallet addresses of P given by W_e = $f(PK_e)$, the map between the wallet address and P is broken for all other parties other than EA and P.

Batching the Requests. It is important to note that if a participant wishes to change her vote within the same voting interval, she should submit the request with the new wallet address to EA who will approve the request in batches (aggregating multiple such requests) in order to improve the resistance against mapping of former wallet addresses to new ones. In detail, the EA will mark all former addresses as *invalid* and approve the new ones within a single transaction. In the extreme case, when a batch contains only one participant (who wanted to change her vote), Adv_{net} can map the two wallets. Therefore, such a participant should value her wallet privacy and vote only once during the next interval (or alternatively change her address again and be a part a bigger batch). Using VPN/dVPNs may further limit the Adv_{net}'s ability to map participant IP addresses.

Algorithm 9: Trigger mechanism.

```
Def VerifyTrigger ($y$,$\pi$,$T$,$BH$,$params\_struct$):
    $b$ = -1
    $tt$ = readState($params\_struct.totaltime$)
    $ft$ = readState($params\_struct.ft$)
    $seed$ = readState($params\_struct.key$)
    $IntervalTime$ = $tt/ft$
    $BHsInInterval$ = $IntervalTime/10$
    if $SHA256(BH) < T$ then
        if $Verify\_VDF(y,\pi) == True$ then
            $a = SHA(y)$
            $b = a(mod\ BHsInInterval)$
    if $b$==$0$ then
        return $True$
    else
        return $False$
```

Functionality

The high-level functionality of the AoV framework and its smart contracts is shown in Algorithm 10, and the trigger mechanism is presented in Algorithm 9. Algorithm 10 comprises of 5 main functions — *setup, registration, voting, tally* and *revote*. The system parameters agreed upon are added by EA using the *setup* function of the smart contract. The EA is also responsible for adding the list of valid participants' wallet addresses to the contract through the *registration* function. The *voting* function is supplied by a participant's wallet address, blinded vote, and its proof of correctness. This information is signed with P_i's private key and sent to the contract. The *voting* function carries out the necessary verifications and adds her vote. The participant wallet address is set to "voted" to disallow its reuse. Before invoking the *tally* function, the VDF prover and BPO store their respective data to the contract (see Algorithm 7 and Algorithm 8). Next, the *tally* function is called by EA or any authorized participant. The *tally* function carries out two main tasks. First, it checks whether the condition to trigger the interval tally is satisfied (see Algorithm 9). The second task (when triggered) is to tally the votes and return the results. When a participant wishes to vote again, she sends her next wallet address (synchronized with EA) to the *revote* function. The EA will verify the new wallet address offline and call the *registration* function to set the new address to valid (preferably in batches as mentioned in Section 6.3.3). Next, a participant may call the function *Voting* and vote using her new wallet address.

6.3.4 Security Analysis

Mining Adversary

The goal of Adv_{min} is to find a valid nonce s that solves the Bitcoin puzzle such that b in Equation 6.12 is 0. When these two conditions are met, and the new epoch is about to start, the validator contract triggers the tally computation of votes. We set the difficulty

Algorithm 10: Always on Voting Framework.

Input: Set1: $\forall$ participants P_i, wallet addresses (WA_{ij}), blinded vote BV_{ij} (by P_i for her j^{th} voting occurence, $j = 0,1,2,3,...$) & zero knowledge proof of vote correctness (ZKP_{ij}), $booth_{no}$. Set2: system parameters $init\_params$, BTC blockheader BH, $VDF(BH)$, proof π, BTC target T, BTC blockheight.

Output: Total tally of votes in the interval.

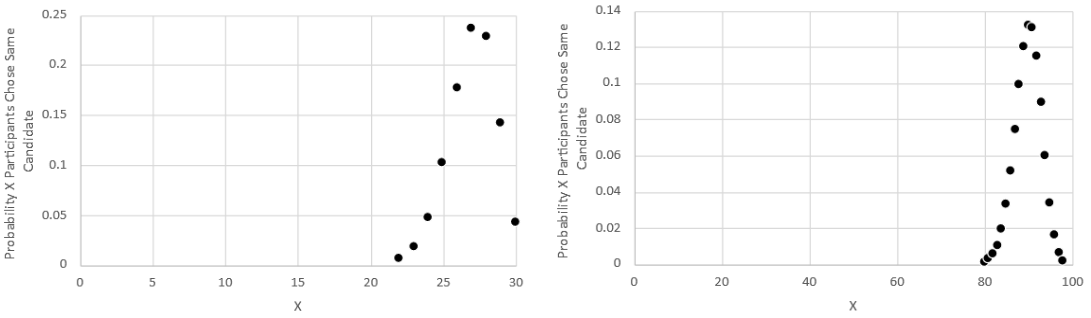
```
Function Setup($init\_params$):
    writeState($params\_struct$,$init\_params$) // add system parameters as key-value pairs into
        $params\_struct$.

Function Registration($msg1 = WA_{ij}$,$msg2 = valid\_flag$,$EA\_signed\_msg$):
    $msg = msg1 \mid\mid msg2$ // concatenate message parts.
    $EA\_pubkey$= readState($params\_struct.EA\_public\_key$) // get $EA$ public key.
    if $VerifySig(msg, EA\_signed\_msg, EA\_pubkey) = True$ then
        if $valid\_flag == True$ then
            writeState($WA_{ij}$, "$valid$") // set wallet address to valid.
        else
            writeState($WA_{ij}$, "$invalid$") // set wallet address to invalid.

Function Voting($msg1 = WA_{ij}$,$msg2 = BV_{ij}$,$msg3 = ZKP_{ij}$,$P_i\_signed\_msg$):
    $msg = msg1 \mid\mid msg2 \mid\mid msg3$
    $wallet\_status$= readState($WA_{ij}$)
    $P_i\_pubkey$= readState($params\_struct.P_i\_public\_key$)
    $sig\_flag$ = VerifySig($msg$,$P_i\_signed\_msg$,$P_i\_pubkey$)
    $zkp\_flag$ = VerifyZKP($BV_{ij}$,$ZKP_{ij}$)
    if ($sig\_flag$ and $zkp\_flag$) == $True$ and $wallet\_status$ == "$valid$" then
        writeState("$vote$" || $WA_{ij}$,$BV_{ij}$) // The latest wallet address of $P_i$ is mapped to
            her private vote.  The key in (key,value) is prefixed with 'vote' tag
            to identify valid votes w.r.t.  wallet addresses.
        writeState($WA_{ij}$, "$voted$") // set $WA$ to voted & prevent voting from that address
            again.

Function Tally($blockheight$):
    $total\_tally = -1$
    $stored\_blockheight$ = readState("$blockheightStored$")// $blockheightStored$ is from
        Algorithm 8.
    $BH$ =readState("$blockheader$" || $blockheight$)// $blockheight$ is the argument passed to
        the function Tally.
    $y$,$\pi$ = readState("$vdfadd$" || $blockheight$)// $vdfadd$ is read from Algorithm 7.
    $BH$,$Target$ = readState("$bpoadd$" || $blockheight$)
    $trigger\_flag$=VerifyTrigger($y$,$\pi$,$Target$,$BH$,$params\_struct$)// Call Algorithm Algorithm 9.
    if ($stored\_blockheight$ == $blockheight$) and $trigger\_flag$ == $True$ then
        $total\_tally = \sum_{no=1}^{X \cdot Y} local\_tally(booth_{no})$
    return $total\_tally$

Function Revote($msg = WA_{ij}$,$P_i\_signed\_msg$):
    $P_i\_pubkey$= readState($params\_struct.P_i\_public\_key$) // get $P_i$ public key
    if $VerifySig(msg, P_i\_signed\_msg, P_i\_pubkey) = True$ then
        writeState($WA_{ij}$, "$pending$") // set wallet address to pending verification by
            $EA$.
    // Next, $EA$ calls $Registration()$, where it sets a new wallet address of $P_i$ to
        valid.
    // Further, $P_i$ calls $Voting()$ to re-vote using the new wallet address.
```

VDF Prover	Time (minutes)	VDF Prover	Time (minutes)
1	0-100	1	100-199
2	10-110	2	110-209
3	20-120	3	120-219
4	30-130	4	130-229
5	40-140	5	140-239
6	50-150	6	150-249
7	60-160	7	160-259
8	70-170	8	170-269
9	80-180	9	180-279
10	90-190	10	190-289

Table 6.4: Scheduling 10 VDF provers without queuing. Note the VDF computations on a VDF prover machine are not parallelized. It is the scheduling alone that is in parallel. The start time is based on the job arrival time at the VDF prover, where it will run for 100 minutes. Once completed, it is ready to take on the next job. In column 2, the start times are 10 minutes apart and correspond to the average BTC interblock (job) arrival time. The largest idle time in column 1 is for VDF Prover 10 at 90 minutes, waiting for the job to start. Beyond this, all VDF prover machines are continuously occupied since a new job is available to start immediately after the current job ends.

for the benign VDF prover (with commercial hardware) to take 100 minutes[25] to solve VDF(BH). Based on A_{max} limit, we assume Adv_{min} to take at least 10 minutes to solve the VDF. As a result, Adv_{min} is restricted to a maximum of 1 try (considering 10 minutes as an average Bitcoin block creation time), excluding the Proof-of-Work required to solve the Bitcoin mining puzzle. However, since a Bitcoin block header is generated on average once every 10 minutes and the benign VDF prover is occupied for 100 minutes, the question is – how many VDF provers are required to prevent the block headers from queuing up? We can see in Table 6.4 that VDF prover 1 runs a task for time 0-100 minutes, and she picks up the next task to run for time 100-199 minutes. Similarly, all other provers pick up the next task after completing the present one. Hence, 10 VDF provers are sufficient to prevent block headers from queuing up because $A_{max} = 10$. On the other hand, a benign VDF prover might reduce A_{max} of VDF computation by using specialized hardware instead of commercial hardware (depending on the cost-to-benefit ratio). However, we emphasize that the VDF can be computed only after solving the PoW mining puzzle, which is prohibitively expensive. Moreover, the puzzle difficulty increases proportionally to the mining power of the Bitcoin network. Hence, the proposed serial combination of solving the Bitcoin mining puzzle followed by the computation of VDF output improves the aggregate security against Adv_{min} from choosing a favorable nonce. The estimated requirement of $A_{max} = 10$ might be further increased as more studies to efficiently solve VDFs on ASICs are carried out. However, if A_{max} will increase in the future, our solution can cope with it by employing more VDF provers.

[25] We consider $A_{max} = 10$, i.e., what is solved by a benign VDF prover in 10 units of time, while in the case of Adv_{min} it is in 1 unit.



(a) Booth with 30 participants, winning $p = 0.9$. (b) Booth with 100 participants, winning $p = 0.9$.

Figure 6.15: Binomial probability distribution function of X booth participants voting for their favorite candidate whose winning probability is p.

Implications of VDF Prover Synchronization and Optimizing Frequency of Supplied Block Headers

Several VDF provers are synchronised to supply the VDF proofs to the validator contract in sequence. However, there are no adverse effects when the proofs are generated and supplied out of sequence. The validator smart contract stores the latest *block height* for which the VDF proof was last accepted. It only allows proof verification for stored *block height+1* on the contract and any out-of-order proofs have to be resent. Once the order is corrected, a handful of VDF proofs may appear in quick succession at the validator contract. However, the tally for the interval is only triggered when VC_{output} in Equation 6.15 is *True*.

In terms of gas consumption, it can be costly to process every single Bitcoin block header (supplied to the VDF prover and the validator contract by the BPO). We suggest optimizing this by choosing a coarser time granularity of the block header supply, independent of the Bitcoin block interval (e.g., every x-th block). We modify the example from Section 6.3.3 by considering the processing of every 100^{th} Bitcoin block header.[26] $TotalTime$ = 4 years = $525600 \cdot 4$ minutes and the total number of intervals $ft = 8$. Then, $IntervalTime = (525600 \cdot 4)/(8) = 262800$ minutes and $BHsInInterval = 262800/(10 \cdot 100) = 262.8$. On average, the oracle will send 262.8 block headers (BH values) to the validator contract within 182.5 days instead of the 26280 block headers required in the original example. In this case, we only need 1 VDF prover instead of 10, and it provides similar security guarantees as before.

Privacy Implications of Booth Sharding

In this section, we look at the implications of booth sharding. Further, we recommend booth sizes to protect participant votes from being revealed to the EA (in the case when all participants in a booth voted for the same candidate). The map between a participant's

[26] Note that this would need another condition to be met, i.e., the block height in BH (mod 100) should be equal to 0.

wallet and her vote transaction is broken for all parties (including Adv_{net}) but the EA (see Section 6.3.3). The EA is aware of the participants' current wallet address used for voting, and hence it has an advantage over Adv_{net} in statistical inference attacks. Certain scenarios revealing vote choice are possible under some circumstances. In particular, if all participants in a booth voted for the same candidate or the winning probability of one candidate is much higher than the others.

To demonstrate it, in Figure 6.15 we provide the probability that X participants in a booth voted for the same candidate, depending on the candidate winning probability p. Figure 6.15(a) represents a booth with 30 participants and candidate winning probability $p = 0.9$. The probability that all participants voted for the same candidate is $P(30, X = 30, p = 0.9) \cong 0.0423$. Figure 6.15(b) represents a booth with 100 participants and candidate winning probability $p = 0.9$. The probability that all participants voted for the same candidate is $P(100, X = 100, p = 0.9) \cong 0.00003$, which demonstrates that number of participants in a booth, influences p in indirect proportion, favoring the booths with higher sizes.

Even though the probabilities in the booth with 100 participants are very low, this may not be sufficient depending on the total number of participants. For example, consider the elections with 1 million participants. First, let number of participants in a booth be 30 and the number of booths $M = \lceil(1\ 000\ 000/30)\rceil = 33334$. The number of booths where all participants likely voted for the same candidate is $0.0423 \cdot 33334 \cong 1410$. For booths with 100 participants each and $M = \lceil(1\ 000\ 000/100)\rceil = 10000$, the number of booths where all participants likely voted for the same candidate is reduced to $0.00003 \cdot 10000 \cong 0.3$. Therefore, a suitable number of participants per booth should be determined based on the extreme estimations of the tally results and the total number of voters.

Randomness of Bitcoin Nonces & AoV Entropy

We decided to utilize a single public source of randomness instead of a distributed randomness due to the low computation cost and synchronization complexity. Bonneau et al. [BCG15] showed that if the underlying hash function used to solve the Bitcoin PoW puzzle is secure, then each block in the canonical chain has a computational min-entropy of at least d bits, representing the mining difficulty. I.e., d consecutive 0 bits must appear in the hash of the block header.[27] Hence, $\lfloor \frac{d}{2} \rfloor$ near-uniform bits can be securely extracted. Nevertheless, empirical evaluation has shown that Bitcoin nonces have visible white spaces (non-uniformity) in its scatter-plot [Bit19]. A possible explanation is that some miners are presetting some of the bits in the 32-bit $nonce$ field and using the $extraNonce$ to solve the PoW puzzle. We use the entire block header as the initial source of entropy instead of the 32-bit nonce alone to avoid such biases. To reduce the probability of Adv_{min} biasing the solution in her favor, the block header is passed through a verifiable delay function (see Equation 6.8). The output of VDF is hashed (see Equation 6.11) to consolidate the entropy.

[27] At the time of writing, $d \approx 76$.

6.4 Contributing Papers

The papers that contributed to this research direction are enumerated in the following, while highlighted papers are attached to this thesis in their original form.

[HLS23] **Ivan Homoliak, Zengpeng Li, and Pawel Szalachowski. BBB-Voting: 1-out-of-k blockchain-based boardroom voting. In *IEEE International Conference on Blockchain, Blockchain 2023, Hainan, China, December 17-21, 2023*. IEEE, 2023.**

[SH23] **Ivana Stančíková and Ivan Homoliak. SBvote: Scalable self-tallying blockchain-based voting. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, pages 203–211, 2023.**

[VSH23] **Sarad Venugopalan, Ivana Stančíková, and Ivan Homoliak. Always on Voting: A framework for repetitive voting on the blockchain. *IEEE Transactions on Emerging Topics in Computing*, 2023.**

Chapter 7

Secure Logging

In this chapter, we present our contributions to the area of secure logging using blockchain, which belongs to the application layer of our security reference architecture (see Chapter 3) and its category of data provenance. In particular, this chapter is focused on the security, privacy, and scalability, of blockchain-based secure logging, which we further extend to the use case of a global Central Bank Digital Currency (CBDC). Note that we base this chapter on not yet published papers [HS20] [HPH$^+$23] (see also Section 7.3), and therefore we consider this chapter as additional to the thesis.

First, we present Aquareum [HS20], a novel framework for centralized ledgers removing their main limitations. By combining a trusted execution environment with a public blockchain platform, Aquareum provides publicly verifiable, non-equivocating, censorship-evident, private, and high-performance ledgers. Aquareum ledgers are integrated with a Turing-complete virtual machine, allowing arbitrary transaction processing logics, including tokens or client-specified smart contracts (see details in Section 7.1).

Next, we aim at interoperability for the environment of Central Bank Digital Currency (CBDC) containing multiple instances of centralized ledgers (based on Aquareum) that either represent central banks of more countries or retail banks (with a single central bank) of a single country. In detail, we present CBDC-AquaSphere [HPH$^+$23] a practical blockchain interoperability protocol that integrates important features of Digital Euro Association manifesto [Dig22], such as strong value proposition for the end users, the highest degree of privacy, and interoperability. On top of the above-mentioned features, our work also provides proof-of-censorship and atomicity (see details in Section 7.2).

7.1 Aquareum

We depict overview of components that Aquareum consists of in Figure 7.1. In the following, we elaborate on the system model and principles of our approach. For background related to integrity preserving data structures and trusted computing, we refer the reader to Section 2.8 and Section 2.11.

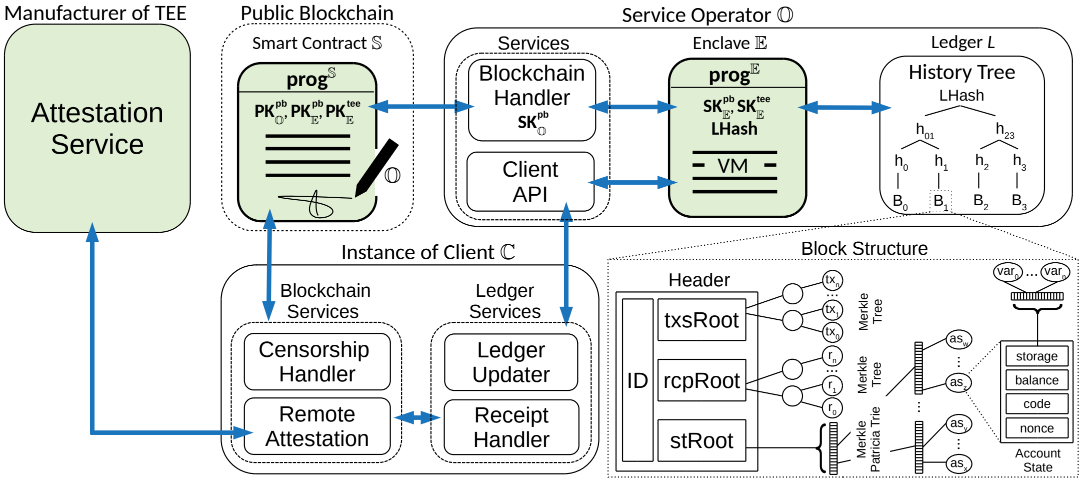


Figure 7.1: Aquareum components. Trusted components are depicted in green.

Notation

By $\{msg\}_{\mathbb{U}}$, we denote the message msg digitally signed by $\mathbb{U}$, and by $msg.\sigma$ we refer to a signature; $h(.)$ stands for a cryptographic hash function; $\|$ is the string concatenation; $\%$ represents modulo operation over integers; $\Sigma_p.\{KeyGen, Verify, Sign\}$ represents a signature (and encryption) scheme of the platform p, where $p \in \{pb, tee\}$ (i.e., public blockchain platform and trusted execution environment platform); and $SK^p_{\mathbb{U}}$, $PK^p_{\mathbb{U}}$ is the private/public key-pair of $\mathbb{U}$, under Σ_p. Then, we use π^s for denoting proofs of various data structures $s \in \{mk, mem, inc\}$: π^{mk} denotes the inclusion proof in the Merkle tree, π^{mem} and π^{inc} denote the membership proof and the incremental proof in the history tree, respectively.

7.1.1 System Model

In Aquareum, an *operator* is an entity that maintains and manages a ledger containing chronologically sorted transactions. *Clients* interact with the ledger by sending requests, such as queries and transactions to be handled. We assume that all involved parties can interact with a blockchain platform supporting smart contracts (e.g., Ethereum). Next, we assume that the operator has access to a TEE platform (e.g., Intel SGX). Finally, we assume that the operator can be malicious and her goals are as follows:

- **Violation of the ledger's integrity** by creating its internal inconsistent state – e.g., via inserting two conflicting transactions or by removing/modifying existing transactions.
- **Equivocation of the ledger** by presenting at least two inconsistent views of the ledger to (at least) two distinct clients who would accept such views as valid.
- **Censorship of client queries** without leaving any audit trails evincing the censorship occurrence.

Next, we assume that the adversary cannot undermine the cryptographic primitives used, the underlying blockchain platform, and the TEE platform deployed.

Desired Properties

We target the following security properties for Aquareum ledgers:

Verifiability: clients should be able to obtain easily verifiable evidence that the ledger they interact with is internally *correct* and *consistent*. In particular, it means that none of the previously inserted transaction was neither modified nor deleted, and there are no conflicting transactions. Traditionally, the verifiability is achieved by replicating the ledger (like in blockchains) or by trusted auditors who download the full copy of the ledger and sequentially validate it. However, this property should be provided even if the operator does not wish to share the full database with third parties. Besides, the system should be *self-auditable*, such that any client can easily verify (and prove to others) that some transaction is included in the ledger, and she can prove the state of the ledger at the given point in time.

Non-Equivocation: the system should protect from forking attacks and thus guarantee that no concurrent (equivocating) versions of the ledger exist at any point in time. The consequence of this property is that whenever a client interacts with the ledger or relies on the ledger's logged artifacts, the client is ensured that other clients have ledger views consistent with her view.

Censorship Evidence: preventing censorship in a centralized system is particularly challenging, as its operator can simply pretend unavailability in order to censor undesired queries or transactions. Therefore, this property requires that whenever the operator censors client's requests, the client can do a resolution of an arbitrary (i.e., censored) request publicly. We emphasize that proving censorship is a non-trivial task since it is difficult to distinguish "pretended" unavailability from "genuine" one. Censorship evidence enables clients to enforce potential service-level agreements with the operator, either by a legal dispute or by automated rules encoded in smart contracts.

Besides those properties, we intend the system to provide *privacy* (keeping the clients' communication confidential), *efficiency* and *high performance*, not introducing any significant overhead, *deployability* with today's technologies and infrastructures, as well as *flexibility* enabling various applications and scenarios.

7.1.2 High-Level Overview

Aquareum ledger is initialized by an operator ($\mathbb{O}$) who creates an internal ledger (L) that will store all transactions processed and the state that they render. Initially, L contains an empty transaction set and a null state. During the initialization, $\mathbb{O}$ creates a TEE enclave ($\mathbb{E}$) whose role is to execute updates of L and verify consistency of L before each update. Initialization of $\mathbb{E}$ involves the generation of two public private key pairs – one for the signature scheme of TEE (i.e., $PK_{\mathbb{E}}^{tee}$,$SK_{\mathbb{E}}^{tee}$) and one for the signature scheme of the

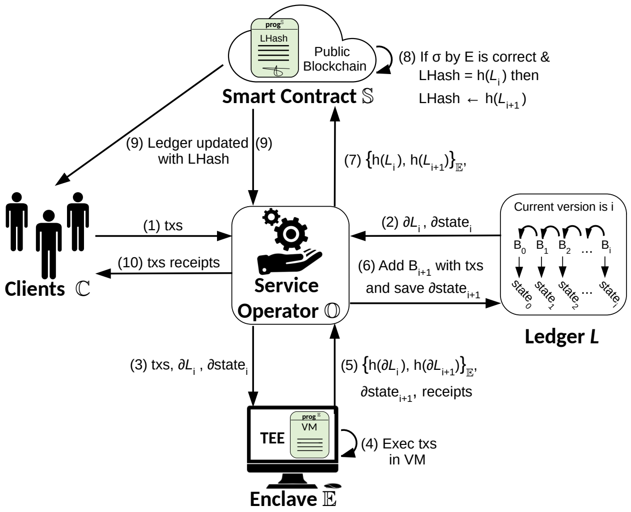

prog$^{\mathbb{S}}$
LHash
Public Blockchain
(8) If σ by E is correct & LHash = h(L_i) then LHash ← h(L_{i+1})
Smart Contract $\mathbb{S}$
(9) Ledger updated with LHash
(9)
(7) $\{h(L_i), h(L_{i+1})\}_{\mathbb{E}}$,
Current version is i
B_0 B_1 B_2 ... B_i
$state_0$ $state_1$ $state_2$ ... $state_i$
(1) txs
(2) ∂L_i , $\partial state_i$
(10) txs receipts
(6) Add B_{i+1} with txs and save $\partial state_{i+1}$
Clients $\mathbb{C}$
Service Operator $\mathbb{O}$
Ledger *L*
(3) txs, ∂L_i , $\partial state_i$
(5) $\{h(\partial L_i), h(\partial L_{i+1})\}_{\mathbb{E}}$, $\partial state_{i+1}$, receipts
TEE
prog$^{\mathbb{E}}$
VM
(4) Exec txs in VM
Enclave $\mathbb{E}$

Figure 7.2: Operation procedure of Aquareum ledger.

public blockchain (i.e., $PK_{\mathbb{E}}^{pb}$,$SK_{\mathbb{E}}^{pb}$).[1] The code of $\mathbb{E}$ is public (see Algorithm 12 and Algorithm 13), and it can be remotely attested with the TEE infrastructure by any client.

Next, $\mathbb{O}$ generates her public-private key pair (i.e., $PK_{\mathbb{O}}$,$SK_{\mathbb{O}}$) and deploys a special smart contract $\mathbb{S}$ (see Algorithm 11) initialized with the empty L represented by its hash $LHash$, the operator's public key $PK_{\mathbb{O}}$, and both enclave public keys $PK_{\mathbb{E}}^{tee}$ and $PK_{\mathbb{E}}^{pb}$. After the deployment of $\mathbb{S}$, an instance of L is uniquely identified by the address of $\mathbb{S}$. A client ($\mathbb{C}$) wishing to interact with L obtains the address of $\mathbb{S}$ and performs the remote attestation of $\mathbb{E}$ using the $PK_{\mathbb{E}}^{tee}$.

Whenever $\mathbb{C}$ sends a transaction to $\mathbb{O}$ (see Figure 7.2), $\mathbb{E}$ validates whether it is authentic and non-conflicting; and if so, $\mathbb{E}$ updates L with the transaction, yielding the new version of L. The $\mathbb{C}$ is responded with a *receipt* and *"a version transition of L"*, both signed by $\mathbb{E}$, which prove that the transaction was processed successfully and is included in the new version of L. For efficiency reasons, transactions are processed in batches that are referred to as *blocks*. In detail, $\mathbb{O}$ starts the update procedure of L (see Figure 7.2) as follows:

a) $\mathbb{O}$ sends all received transactions since the previous update to $\mathbb{E}$, together with the current partial state of L and a small subset of L's data ∂L_i, such that $h(\partial L_i) = h(L_i)$, which is required to validate L's consistency and perform its incremental extension.
b) $\mathbb{E}$ validates and executes the transactions in its virtual machine, updates the current partial state and partial data of L, and finally creates a blockchain transaction[2] $\{h(\partial L_i),$ $h(\partial L_{i+1})\}_{\mathbb{E}}$ signed by $SK_{\mathbb{E}}^{pb}$, which represents a version transition of the ledger from version i to its new version $i+1$, also referred to as the **version transition pair**.
c) The blockchain transaction with version transition pair is returned to $\mathbb{O}$, who sends this transaction to $\mathbb{S}$.
d) $\mathbb{S}$ accepts the second item of the version transition pair iff it is signed by $SK_{\mathbb{E}}^{pb}$ and the

[1] Note that neither of the private keys ever leaves $\mathbb{E}$.

[2] Note that $\{h(\partial L_i),h(\partial L_{i+1})\}_{\mathbb{E}} = \{h(L_i),h(L_{i+1})\}_{\mathbb{E}}$

Algorithm 11: The program $prog^{\mathbb{S}}$ of the smart contract $\mathbb{S}$

```
▷ DECLARATION OF TYPES AND CONSTANTS:
    CensInfo { $etx$,$equery$,$status$,$edata$ },
    $msg$: a current transaction that called $\mathbb{S}$,
▷ DECLARATION OF FUNCTIONS:
function $Init(PK^{pb}_{\mathbb{E}}, PK^{tee}_{\mathbb{E}}, PK_{\mathbb{O}})$ public
    $PK^{tee}_{\mathbb{E}}[].add(PK^{tee}_{\mathbb{E}})$;        ▷ PK of enclave $\mathbb{E}$ under $\Sigma_{tee}$.
    $PK^{pb}_{\mathbb{E}}[].add(PK^{pb}_{\mathbb{E}})$;        ▷ PK of enclave $\mathbb{E}$ under $\Sigma_{pb}$.
    $PK^{pb}_{\mathbb{O}} \leftarrow PK_{\mathbb{O}}$;        ▷ PK of operator $\mathbb{O}$ under $\Sigma_{pb}$.
    $LRoot_{pb} \leftarrow \perp$;        ▷ The most recent root hash of L synchronized with $\mathbb{S}$.
    $censReqs \leftarrow []$;        ▷ Request that $\mathbb{C}$s wants to resolve publicly.
function $PostLRoot(root_A, root_B, \sigma)$ public
        ▷ Verify whether a state transition was made within $\mathbb{E}$.
    assert $\Sigma_{pb}.verify((\sigma, PK^{pb}_{\mathbb{E}}[-1]), (root_A, root_B))$;
        ▷ Verify whether a version transition extends the last one.
    if $LRoot_{pb} = root_A$ then
        $LRoot_{pb} \leftarrow root_B$;        ▷ Do a version transition of L.
function $ReplaceEnc(PKN^{pb}_{\mathbb{E}}, PKN^{tee}_{\mathbb{E}}, r_A, r_B, \sigma, \sigma_{msg})$ public
        ▷ Called by $\mathbb{O}$ in the case of enclave failure.
    assert $\Sigma_{pb}.verify((\sigma_{msg}, PK^{pb}_{\mathbb{O}}), msg)$;        ▷ Avoiding MiTM attack.
    $PostLRoot(r_A, r_B, \sigma)$ ;        ▷ Do a version transition.
    $PK^{tee}_{\mathbb{E}}.add(PKN^{tee}_{\mathbb{E}})$;        ▷ Upon change, $\mathbb{C}$s make remote attestation.
    $PK^{pb}_{\mathbb{E}}.add(PKN^{pb}_{\mathbb{E}})$;
function $SubmitCensTx(etx, \sigma_{msg})$ public
        ▷ Called by $\mathbb{C}$ in the case her TX is censored.
    accessControl($\sigma_{msg}$,$msg.PK^{pb}_{\mathbb{C}}$);
    $censReqs$.add(CensInfo($etx$,$\perp$,$\perp$,$\perp$));
function $ResolveCensTx(idx_{req}, status, \sigma)$ public
        ▷ Called by $\mathbb{O}$ to prove that $\mathbb{C}$'s TX was processed.
    assert $idx_{req} < |censReqs|$;
    $r \leftarrow censReqs[idx_{req}]$;
    assert $\Sigma_{pb}.verify((\sigma, PK^{pb}_{\mathbb{E}}[-1]),\ (h(r.etx), status))$;
    $r.status \leftarrow status$;
function $SubmitCensQry(equery, \sigma_{msg})$ public
        ▷ Called by $\mathbb{C}$ in the case its read query is censored.
    accessControl($\sigma_{msg}$,$msg.PK^{pb}_{\mathbb{C}}$);
    $censReqs$.add(CensInfo($\perp$,$equery$,$\perp$,$\perp$));
function $ResolveCensQry(idx_{req}, status, edata, \sigma)$ public
        ▷ Called by $\mathbb{O}$ as a response to the $\mathbb{C}$'s censored read query.
    assert $idx_{req} < |censReqs|$;
    $r \leftarrow censReqs[idx_{req}]$;
    assert $\Sigma_{pb}.verify((\sigma, PK^{pb}_{\mathbb{E}}[-1]), (h(r.equery), status, h(edata)))$;
    $r.\{edata \leftarrow edata, status \leftarrow status\}$;
```

current hash stored by $\mathbb{S}$ (i.e., $LHash$) is equal to the first item of the pair.

After the update of L is finished, clients with receipts can verify that their transactions were processed by $\mathbb{E}$ (see details in [HS20]). The update procedure ensures that the new version of L is: (1) **internally correct** since it was executed by trusted code of $\mathbb{E}$, (2) a **consistent** extension of the previous version – relying on trusted code of $\mathbb{E}$ and a witnessed version transition by $\mathbb{S}$, and (3) **non-equivocating** since $\mathbb{S}$ stores only hash of a single version of L (i.e., $LHash$) at any point in time.

Algorithm 12: The program $prog^{\mathbb{E}}$ of enclave $\mathbb{E}$

```
▷ DECLARATION OF TYPES AND FUNCTIONS:
    Header { $ID$, $txsRoot$, $rcpRoot$, $stRoot$};
    $\#(r) \rightarrow v$: denotes the version $v$ of $L$ having $LRoot = r$,
▷ VARIABLES OF TEE:
    $SK^{tee}_{\mathbb{E}}$,$PK^{tee}_{\mathbb{E}}$: keypair of $\mathbb{E}$ under $\Sigma_{tee}$,
    $SK^{pb}_{\mathbb{E}}$,$PK^{pb}_{\mathbb{E}}$: keypair of $\mathbb{E}$ under $\Sigma_{pb}$,
    $hdr_{last} \leftarrow \perp$: the last header created by $\mathbb{E}$,
    $LRoot_{pb} \leftarrow \perp$: the last root of $L$ flushed to PB,
    $LRoot_{cur} \leftarrow \perp$: the root of $L \cup blks_p$ (not flushed to PB),
    $ID_{cur} \leftarrow 1$: the current version of $L$ (not flushed to PB),
▷ DECLARATION OF FUNCTIONS:
function $Init()$ public
    $(SK^{pb}_{\mathbb{E}}, PK^{pb}_{\mathbb{E}}) \leftarrow \Sigma_{pb}.Keygen()$;
    $(SK^{tee}_{\mathbb{E}}, PK^{tee}_{\mathbb{E}}) \leftarrow \Sigma_{tee}.Keygen()$;
    Output($PK^{tee}_{\mathbb{E}}$,$PK^{pb}_{\mathbb{E}}$);
function $Exec(txs[], \partial st^{old}, \pi^{inc}_{next}, LRoot_{tmp})$ public
    assert $\partial st^{old}.root = hdr_{last}.stRoot$;
    $\partial st^{new}, rcps, txs_{er} \leftarrow processTxs(txs, \partial st^{old}, \pi^{inc}_{next}, LRoot_{tmp})$;
    $\sigma \leftarrow \Sigma_{pb}.sign(SK^{pb}_{\mathbb{E}}, (LRoot_{pb}, LRoot_{cur}))$;
    Output($LRoot_{pb}$,$LRoot_{cur}$,$\partial st^{new}$,$hdr_{last}$,$rcps$, $txs_{er}$, $\sigma$);
function $Flush()$ public
    $LRoot_{pb} \leftarrow LRoot_{cur}$;                          ▷ Shift the version of L synchronized with PB.
function $processTxs(txs[], \partial st^{old}, \pi^{inc}_{next}, LRoot_{tmp})$ private
    $\partial st^{new}, rcps[], txs_{er} \leftarrow$ runVM($txs$, $\partial st^{old}$);   ▷ Run $txs$ in VM.
    $txs \leftarrow txs \setminus txs_{er}$;                      ▷ Filter out parsing errors/wrong signatures.
    $hdr \leftarrow$ Header($ID_{cur}$,$MkRoot(txs)$,$MkRoot(rcps)$,$\partial st^{new}.root$));
    $hdr_{last} \leftarrow hdr$;
    $ID_{cur} \leftarrow ID_{cur} + 1$;
    $LRoot_{cur} \leftarrow newLRoot(hdr, \pi^{inc}_{next}, LRoot_{tmp})$;
    return $\partial st^{new}$, $rcps$, $txs_{er}$;
function $newLRoot(hdr, \pi^{inc}_{next}, LRoot_{tmp})$ private
                    ▷ A modification of the incr. proof. template to contain hdr
    assert $\#(LRoot_{cur}) + 1 = \#(LRoot_{tmp})$;              ▷ 1 block $\Delta$.
    assert $\pi^{inc}_{next}.Verify(LRoot_{cur}, LRoot_{tmp})$;
    $\pi^{inc}_{next}$[-1] $\leftarrow h(hdr)$;
    return $deriveNewRoot(\pi^{inc}_{next})$;
```

Whenever $\mathbb{C}$ suspects that her transactions or read queries are censored, $\mathbb{C}$ might report censorship via $\mathbb{S}$. To do so, $\mathbb{C}$ encrypts her request with $PK^{pb}_{\mathbb{E}}$ and publishes it on the blockchain. $\mathbb{O}$ noticing a new request is obligated to pass the request to $\mathbb{E}$, which will process the request and reply with an encrypted response (by $PK^{pb}_{\mathbb{C}}$) that is processed by $\mathbb{S}$. If a pending request at $\mathbb{S}$ is not handled by $\mathbb{O}$, it is public evidence that $\mathbb{O}$ censors the request. We do not specify how can $\mathbb{C}$ use such a proof; it could be shown in a legal dispute or $\mathbb{S}$ itself could have an automated deposit-based punishments rules.

Design Consideration

We might design L as an append-only chain (as in blockchains), but such a design would bring a high overhead on clients who want to verify that a particular block belongs to L. During the verification, clients would have to download the headers of all blocks between

Algorithm 13: Censorship resolution in $\mathbb{E}$ (part of $prog^{\mathbb{E}}$).

```
function $Decrypt(edata)$ public
    $data \leftarrow \Sigma_{pb}.Decrypt(SK_{\mathbb{E}}^{pb}, edata)$;
    Output($data$);

function $SignTx(etx, \pi_{tx}^{mk}, hdr, \pi_{hdr}^{mem})$ public
    ▷ Resolution of a censored write tx.
    $tx \leftarrow \Sigma_{pb}.Decrypt(SK_{\mathbb{E}}^{pb}, etx)$;
    if $ERROR = parse(tx)$ then
        $status$ = PARSING_ERROR;
    else if $ERROR = \Sigma_{pb}.Verify((tx.\sigma, tx.PK_{\mathbb{C}}^{pb}), tx)$ then
        $status$ = SIGNATURE_ERROR;
    else
        ▷ Verify proofs binding TX to header and header to L.
        assert $\pi_{tx}^{mk}$.Verify($tx, hdr.txsRoot$);
        assert $\pi_{hdr}^{mem}$.Verify($hdr.ID, hdr, LRoot_{pb}$);
        $status \leftarrow$ INCLUDED;
    ▷ TX was processed, so $\mathbb{E}$ can issue a proof.
    $\sigma \leftarrow \Sigma_{pb}.sign(SK_{\mathbb{E}}^{pb}, (h(etx), status))$;
    Output($\sigma$, $status$);

function $SignQryTx(equery, blk, \pi_{hdr}^{mem})$ public
    ▷ Resolution of a censored read tx query.
    $\ldots, id_{tx}, id_{blk}, PK_{\mathbb{C}}^{pb} \leftarrow parse(Decrypt(equery))$;
    if $id_{blk} > \#(LRoot_{pb})$ then
        $status \leftarrow$ BLK_NOT_FOUND,  $edata \leftarrow \perp$;
    else
        assert $\pi_{hdr}^{mem}$.Verify($blk.hdr.ID, blk.hdr, LRoot_{pb}$);
        assert VerifyBlock(blk);                    ▷ Full check of block consistency.
        $tx \leftarrow$ findTx($id_{tx}, blk.txs$);
        if $\perp = tx$ then
            $status \leftarrow$ TX_NOT_FOUND,  $edata \leftarrow \perp$;
        else
            $status \leftarrow$ OK,  $edata \leftarrow \Sigma_{pb}.Encrypt(PK_{\mathbb{C}}^{pb}, tx)$;
    $\sigma \leftarrow \Sigma_{pb}.sign(SK_{\mathbb{E}}^{pb}, (h(equery), status, edata))$;
    Output($\sigma$, $status$, $edata$);

function $SignQryAS(equery, as, \pi_{as}^{mpt})$ public
    ▷ Resolution of a censored read account state query.
    $\ldots, id_{as}, PK_{\mathbb{C}}^{pb} \leftarrow parse(Decrypt(equery))$;
    if $\perp = as$ then
        assert $\pi_{as}^{mpt}.VerifyNeg(id_{as}, LRoot_{cur})$;
        $status \leftarrow$ NOT_FOUND,  $edata \leftarrow \perp$;
    else
        assert $\pi_{as}^{mpt}.Verify(id_{as}, LRoot_{cur})$;
        $status \leftarrow$ OK,  $edata \leftarrow \Sigma_{pb}.Encrypt(PK_{\mathbb{C}}^{pb}, as)$;
    $\sigma \leftarrow \Sigma_{pb}.sign(SK_{\mathbb{E}}^{pb}, (h(equery), status, h(edata)))$;
    Output($\sigma$, $status$, $edata$);
```

the head of L and the block in the query, resulting into linear space & time complexity. In contrast, when a history tree (see Section 2.8.2) is utilized for integrity preservation of L, the presence of any block in L can be verified with logarithmic space and time complexity.

Terminated and Failed Enclave

During the execution of $prog^{\mathbb{E}}$, $\mathbb{E}$ stores its secrets and state objects in a sealed file, which is updated and stored on the hard drive of $\mathbb{O}$ with each new block created. Hence, if $\mathbb{E}$ terminates due a temporary reason, such as a power outage or intentional command by $\mathbb{O}$, it can be initialized again by $\mathbb{O}$ who provides $\mathbb{E}$ with the sealed file; this file is used to recover its protected state objects.

However, if $\mathbb{E}$ experiences a permanent hardware failure of TEE, the sealed file cannot be decrypted on other TEE platforms. Therefore, we propose a simple mechanism that deals with this situation under the assumption that $\mathbb{O}$ is the only allowed entity that can replace the platform of $\mathbb{E}$. In detail, $\mathbb{O}$ first snapshots the header hdr_{sync} of the last block that was synchronized with $\mathbb{S}$ as well as all blocks $blks_{unsync}$ of L that were not synchronized with $\mathbb{S}$. Then, $\mathbb{O}$ restores L and her internal state objects into the version $\#(LRoot_{pb})$. After the restoration of L, $\mathbb{O}$ calls the function $ReInit()$ of $\mathbb{E}$ (see Algorithm 14) with hdr_{sync}, $blks_{unsync}$, and $LRoot_{pb}$ as the arguments. In this function, $\mathbb{E}$ first generates its public/private key-pair $SK_{\mathbb{E}}^{pb}$,$PK_{\mathbb{E}}^{pb}$, and then stores the passed header as hdr_{last} and copies the passed root hash into $LRoot_{cur}$ and $LRoot_{pb}$. Then, $\mathbb{E}$ iterates over all passed unprocessed blocks and their transactions txs, which are executed within VM of $\mathbb{E}$. Before the processing of txs of each passed block, $\mathbb{E}$ calls the unprotected code of $\mathbb{O}$ to obtain the current partial state ∂st^{old} of L and incremental proof template (see details in [HS20]) that serves for extending L within $\mathbb{E}$. However, these unprotected calls are always verified within $\mathbb{E}$ and malicious $\mathbb{O}$ cannot misuse them. In detail, $\mathbb{E}$ verifies ∂st^{old} obtained from $\mathbb{O}$ against the root hash of the state stored in the last header hdr_{last} of $\mathbb{E}$, while the incremental proof template is also verified against $LRoot_{cur}$ in the function $newLRoot()$ of $\mathbb{E}$.

Next, $\mathbb{E}$ processes txs of a block, extends L, and then it calls the unprotected code of $\mathbb{O}$ again, but this time to process txs of the current block by $\mathbb{O}$, and thus getting the same version and state of L in both $\mathbb{E}$ and $\mathbb{O}$. Note that any adversarial effect of this unprotected call is eliminated by the checks made after the former two unprotected calls. When all passed blocks are processed, $\mathbb{E}$ signs the version transition pair $\langle LRoot_{pb}, LRoot_{cur}\rangle$ and

Algorithm 14: Reinitialization of a failed $\mathbb{E}$ (part of $prog^{\mathbb{E}}$).

```
function $ReInit(LRoot_{old}, prevBlks[], hdr_{last})$ public
    $(SK_{\mathbb{E}}^{pb}, PK_{\mathbb{E}}^{pb}) \leftarrow \Sigma_{pb}.Keygen();$
    $hdr_{last} \leftarrow hdr_{last};$
    $LRoot_{cur} \leftarrow LRoot_{old}, LRoot_{pb} \leftarrow LRoot_{old};$
    for $\{b : prevBlks\}$ do
        $\pi_{next}^{inc}, LRoot_{tmp} \leftarrow prog^{\mathbb{O}}.nextIncProof();$
        $\partial st^{old} \leftarrow prog^{\mathbb{O}}.getPartialState(b.txs);$
        assert $\partial st^{old}.root = hdr_{last}.stRoot;$
        $\ldots \leftarrow processTxs(b.txs, \partial st^{old}, \pi_{next}^{inc}, LRoot_{tmp});$
        $LRoot_{ret} \leftarrow prog^{\mathbb{O}}.runVM(b.txs);$                ▷ Run VM at $\mathbb{O}$.
        assert $LRoot_{cur} = LRoot_{ret};$                ▷ $\mathbb{E}$ and $\mathbb{O}$ are at the same point.
    $\sigma \leftarrow \Sigma_{pb}.sign(SK_{\mathbb{E}}^{pb}, (LRoot_{pb}, LRoot_{cur}));$
    Output$(LRoot_{pb}, LRoot_{cur}, \sigma, PK_{pb}^{\mathbb{E}}, PK_{tee}^{\mathbb{E}});$
```

returns it to $\mathbb{O}$, together with the new public keys of $\mathbb{E}$. $\mathbb{O}$ creates a blockchain transaction that calls the function $ReplaceEnc()$ of $\mathbb{S}$ with data from $\mathbb{E}$ passed in the arguments. In $ReplaceEnc()$, $\mathbb{S}$ first verifies whether the signature of the transaction was made by $\mathbb{O}$ to avoid MiTM attacks on this functionality. Then, $\mathbb{S}$ calls its function $PostLRoot()$ with the signed version transition pair in the arguments. Upon the success, the current root hash of L is updated and $\mathbb{S}$ replaces the stored $\mathbb{E}$'s PKs by PKs passed in parameters. Finally, $\mathbb{E}$ informs $\mathbb{C}s$ by an event containing new PKs of $\mathbb{E}$, and $\mathbb{C}$s perform the remote attestation of $prog^{\mathbb{E}}$ using the new key $PK^{\mathbb{E}}_{tee}$ and the attestation service. We refer the reader to Appendix of [HS20] for the relevant pseudo-code of $\mathbb{O}$.

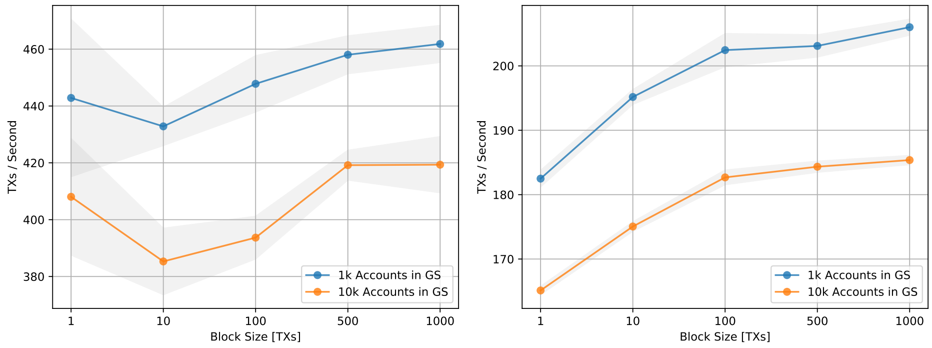
7.1.3 Implementation

We have made a proof-of-concept implementation of Aquareum, where we utilized Intel SGX and C++ for instantiation of $\mathbb{E}$, while $\mathbb{S}$ was built on top of Ethereum and Solidity. Although Aquareum can be integrated with various VMs running within $\mathbb{E}$, we selected EVM since it provides a Turing-complete execution environment and it is widely adopted in the community of decentralized applications. In detail, we utilized OpenEnclave SDK [Ope20] and a minimalistic EVM, called eEVM [Mic20]. However, eEVM is designed with the standard C++ map for storing the full state of L, which lacks efficient integrity-oriented operations. Moreover, eEVM assumes the unlimited size of $\mathbb{E}$ for storing the full state, while the size of $\mathbb{E}$ in SGX is constrained to $\sim$100 MB. This might work with enabled swapping but the performance of $\mathbb{E}$ would be significantly deteriorated with a large full state. Due to these limitations, we replaced eEVM's full state handling by Merkle-Patricia Trie from Aleth [Eth19], which we customized to support operations with the partial state. $\mathbb{O}$ and $\mathbb{C}$ were also implemented in C++.

Our implementation enables the creation and interaction of simple accounts as well as the deployment and execution of smart contracts written in Solidity. We verified the code of $\mathbb{S}$ by static/dynamic analysis tools Mythril [Con19b], Slither [cry18], and ContractGuard [Gua19]; none of them detected any vulnerabilities. The source code of our implementation will be made available upon publication.

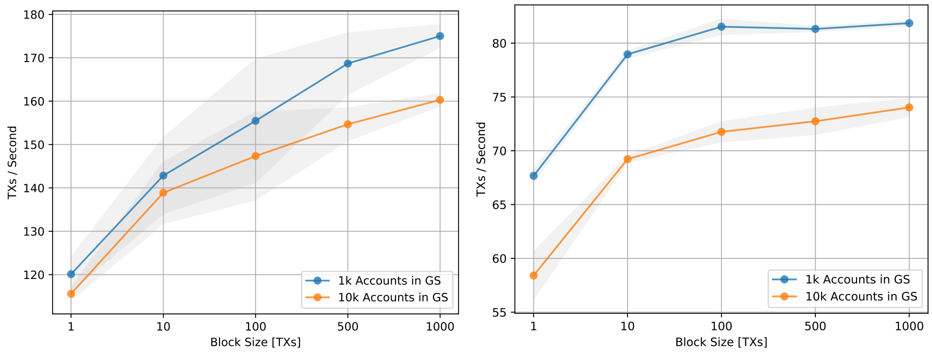
7.1.4 Performance Evaluation

All our experiments were performed on commodity laptop with Intel i7-10510U CPU supporting SGX v1, and they were aimed at reproducing realistic conditions – i.e., they included all operations and verifications described in [HS20], such as verification of recoverable ECDSA signatures, aggregation of transactions by Merkle tree, integrity verification of partial state, etc. We evaluated the performance of Aquareum in terms of transaction throughput per second, where we distinguished transactions with native payments (see Figure 7.3) and transactions with ERC20 smart contract calls (see Figure 7.4). All measurements were repeated 100 times, and we depict the mean and standard deviation in the graphs.



(a) Turbo Boost enabled (b) Turbo Boost disabled

Figure 7.3: Performance of Aquareum for native payments.

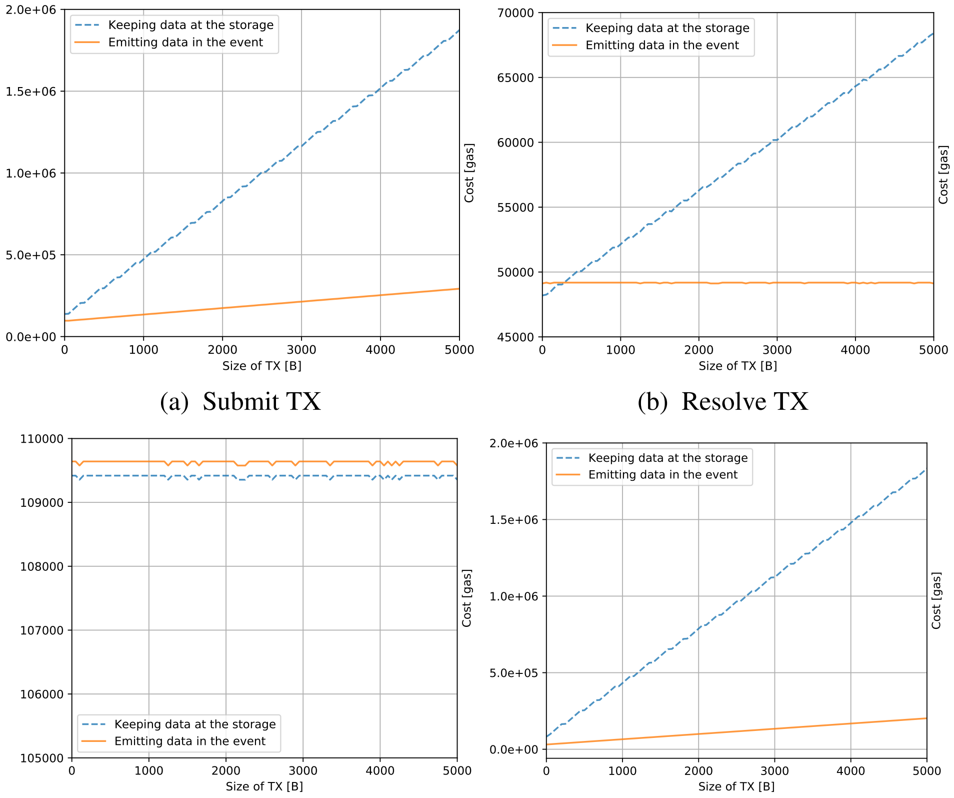


(a) Turbo Boost enabled (b) Turbo Boost disabled

Figure 7.4: Performance of Aquareum for ERC20 smart contract calls.

A Size of the Full State. The performance of Aquareum is dependent on a size of data that is copied from $\mathbb{O}$ to $\mathbb{E}$ upon call of $Exec()$. The most significant portion of the copied data is a partial state, which depends on the height of the MPT storing the full state. Therefore, we repeated our measurements with two different full states, one containing $1k$ accounts and another one containing $10k$ accounts. In the case of native payments, the full state with 10k accounts caused a decrease of throughput by 7.8%-12.1% (with enabled TB) in contrast to the full state with 1k accounts. In the case of smart contract calls, the performance deterioration was in the range 2.8%-8.4% (with enabled TB).

Block Size & Turbo Boost. In each experiment, we varied the block size in terms of the number of transactions in the block. Initially, we performed measurements with enabled Turbo Boost (see Figure 7.3a and Figure 7.4a), where we witnessed a high throughput and its high variability. For smart contract calls (see Figure 7.4a), the throughput increased with the size of the block modified from 1 to 1000 by 45.7% and 38.7% for a full state with 1k and 10k accounts, respectively. However, in the case of native payments the improvement was only 4.3% and 2.8%, while the throughput was not increased monotonically with the block size. Therefore, we experimentally disabled Turbo Boost (see Figure 7.3b) and observed the monotonic increase of throughput with increased block size, where the improvement achieved was 11.41% and 12.26% for a full state with 1k and 10k accounts, respectively. For completeness, we also disabled Trubo Boost in the case of smart con-



(a) Submit TX

(b) Resolve TX

(c) Submit Query (Get TX)

(d) Resolve Query (Get TX)

Figure 7.5: Costs for resolution of censored transactions and queries.

tract calls (see Figure 7.4b), where the performance improvement was 20.9% and 26.7% for both full states under consideration.

Analysis of Costs

Besides the operational cost resulting from running the centralized infrastructure, Aquareum imposes costs for interaction with the public blockchain with $\mathbb{S}$ deployed. The deployment cost of $\mathbb{S}$ is $1.51M$ of gas and the cost of most frequent operation – syncing L with $\mathbb{S}$ (i.e., $PostLRoot()$) – is $33k$ of gas, which is only 33% higher than the cost of a standard Ethereum transaction.[3] For example, if L is synced with $\mathbb{S}$ every 5 minutes, $\mathbb{O}$'s monthly expenses for this operation would be $285M$ of gas, while in the case of syncing every minute, monthly expenses would be $1{,}425M$ of gas.

Censorship Resolution. Our mechanism for censorship resolution imposes costs on $\mathbb{C}$s submitting requests as well as for $\mathbb{O}$ resolving these requests. The cost of submitting a censored request is mainly dependent on the size of the request/response and whether $\mathbb{S}$ keeps data of a request/response in the storage (i.e., an expensive option) or whether it just emits an asynchronous event with the data (i.e., a cheap option). We measured the

[3] This cost is low since we leverage the native signature scheme of the blockchain Σ_{pb}.

costs of both options and the results are depicted in Figure 7.5. Nevertheless, for practical usage, only the option with event emitting is feasible (see solid lines in Figure 7.5).

Figure 7.5a and Figure 7.5b depict the resolution of a censored transaction, which is more expensive for $\mathbb{C}$ than for $\mathbb{O}$, who resolves each censored transaction with constant cost 49k of gas (see Figure 7.5b). On the other hand, the resolution of censored queries is more expensive for $\mathbb{O}$ since she has to deliver a response with data to $\mathbb{S}$ (see Figure 7.5d), while $\mathbb{C}$ submits only a short query, e.g., get a transaction (see Figure 7.5c).

7.1.5 Security Analysis and Discussion

In this section, we demonstrate resilience of Aquareum against adversarial actions that the malicious operator $\mathcal{A}$ can perform to violate the desired properties (see Section 7.1.1).

Theorem 6. *(Correctness) $\mathcal{A}$ is unable to modify the full state of L in a way that does not respect the semantics of VM deployed in $\mathbb{E}$.*

Justification. The update of the L's state is performed exclusively in $\mathbb{E}$. Since $\mathbb{E}$ contains trusted code that is publicly known and remotely attested by $\mathbb{C}$s, $\mathcal{A}$ cannot tamper with this code. □

Theorem 7. *(Consistency) $\mathcal{A}$ is unable to extend L and modify the past records of L.*

Justification. All extensions of L are performed within trusted code of $\mathbb{E}$, while utilizing the history tree [CW09] as a tamper evident data structure, which enables us to make only such incremental extensions of L that are consistent with L's past. □

Theorem 8. *(Verifiability) $\mathcal{A}$ is unable to unnoticeably modify or delete a transaction tx that was previously inserted to L using Π_N, if sync with $\mathbb{S}$ was executed anytime afterward.*

Justification. Since tx was correctly executed (Theorem 6) as a part of the block b_i in a trusted code of $\mathbb{E}$, $\mathbb{E}$ produced a signed version transition pair $\{h(L_{i-1}), h(L_i)\}_{\mathbb{E}}$ of L from the version $i-1$ to the new version i that corresponds to L with b_i included. $\mathcal{A}$ could either sync L with $\mathbb{S}$ immediately after b_i was appended or she could do it n versions later. In the first case, $\mathcal{A}$ published $\{h(L_{i-1}), h(L_i)\}_{\mathbb{E}}$ to $\mathbb{S}$, which updated its current version of L to i by storing $h(L_i)$ into $LRoot_{pb}$. In the second case, n blocks were appended to L, obtaining its $(i+n)$th version. $\mathbb{E}$ executed all transactions from versions $(i+1), \ldots, (i+n)$ of L, while preserving correctness (Theorem 6) and consistency (Theorem 7). Then $\mathbb{E}$ generated a version transition pair $\{h(L_{i-1}), h(L_{i+n})\}_{\mathbb{E}}$ and $\mathcal{A}$ posted it to $\mathbb{S}$, where the current version of L was updated to $i+n$ by storing $h(L_{i+n})$ into $LRoot_{pb}$. When any $\mathbb{C}$ requests tx and its proofs from $\mathcal{A}$ with regard to publicly visible $LRoot_{pb}$, she might obtain a modified tx' with a valid membership proof $\pi^{mem}_{hdr_i}$ of the block b_i but an invalid Merkle proof $\pi^{mk}_{tx'}$, which cannot be forged. □

In the case of tx deletion, $\mathcal{A}$ provides $\mathbb{C}$ with the tampered full block b'_i (maliciously excluding tx) whose membership proof $\pi^{mem}_{hdr'_i}$ is invalid – it cannot be forged. □

Theorem 9. *(Non-Equivocation) Assuming L synced with $\mathbb{S}$: $\mathcal{A}$ is unable to provide two distinct $\mathbb{C}s$ with two distinct valid views on L.*

Justification. Since L is regularly synced with publicly visible $\mathbb{S}$, and $\mathbb{S}$ stores only a single current version of L (i.e., $LRoot_{pb}$), all $\mathbb{C}s$ share the same view on L. □

Theorem 10. *(Censorship Evidence) $\mathcal{A}$ is unable to censor any request (transaction or query) from $\mathbb{C}$ while staying unnoticeable.*

Justification. If $\mathbb{C}$'s request is censored, $\mathbb{C}$ asks for a resolution of the request through public $\mathbb{S}$. $\mathcal{A}$ observing the request might either ignore it and leave the proof of censoring at $\mathbb{S}$ or she might submit the request to $\mathbb{E}$ and obtain an enclave signed proof witnessing that a request was processed – this proof is submitted to $\mathbb{S}$, whereby publicly resolving the request. □

Other Properties and Implications

Privacy VS Performance. Aquareum provides privacy of data submitted to $\mathbb{S}$ during the censorship resolution since the requests and responses are encrypted. However, Aquareum does not provide privacy against $\mathbb{O}$ who has the read access to L. Although Aquareum could be designed with the support of full privacy, a disadvantage of such an approach would be the performance drop caused by the decryption of requested data from L upon every $\mathbb{C}$'s read query, requiring a call of $\mathbb{E}$. In contrast, with partial-privacy, $\mathbb{O}$ is able to respond queries of $\mathbb{C}$s without touching $\mathbb{E}$.

Access Control at $\mathbb{S}$. $\mathbb{C}$s interact with $\mathbb{S}$ only through functions for submission of censored requests. Nevertheless, access to these functions must be regulated through an access control mechanism in order to avoid exhaustion (i.e., DoS) of this functionality by external entities. This can be performed with a simple access control requiring $\mathbb{C}$s to provide access tickets when calling the functions of $\mathbb{S}$. An access ticket could be provisioned by $\mathbb{C}$ upon registration at $\mathbb{O}$, and it could contain $PK_{\mathbb{C}}^{pb}$ with a time expiration of the subscription, signed by $\mathbb{E}$. Whenever $\mathbb{C}$ initiates a censored request, verification of an access ticket would be made by $\mathbb{S}$, due to which DoS of this functionality would not be possible.

Security of TEE. Aquareum assumes that its TEE platform is secure. However, previous research showed that this might not be the case in practical implementations of TEE, such as SGX that was vulnerable to memory corruption attacks [BCD$^+$18] as well as side channel attacks [BCD$^+$17, VBMW$^+$18, LKO$^+$21, MOG$^+$20]. A number of software-based defense and mitigation techniques have been proposed [SLKP17, GLS$^+$17, CZRZ17, BCD$^+$17, SLK$^+$17] and some vulnerabilities were patched by Intel at the hardware level [Int18]. Nevertheless, we note that Aquareum is TEE-agnostic thus can be integrated with other TEEs such as ARM TrustZone or RISC-V architectures (using Keystone-enclave [Enc19] or Sanctum [CLD16]).

Another class of SGX vulnerabilities was presented by Cloosters et al. [CRD20] and involved incorrect application designs enabling arbitrary reads and writes of protected memory or work done by Borrello et al. which involves more serious microarchitectural flaws in chip design [BKS$^+$22]. Since the authors did not provide public with their tool (and moreover it does not support Open-enclave SDK), we did manual inspection of Aquareum code and did not find any of the concerned vulnerabilities.

Security vs. Performance. In SGX, performance is always traded for security, and our intention was to optimize the performance of Aquareum while making custom security checks whenever possible instead of using expensive buffer allocation and copying to/from $\mathbb{E}$ by trusted runtime of SDK (*trts*).

- **Output Parameters:** In detail, in the case of ECALL $Exec()$ function where $\mathbb{E}$ is provided with `[user_check]` output buffers pointing to host memory, the strict location-checking is always made in $\mathbb{E}$ while assuming maximal size of the output buffer passed from the host (i.e., $oe\_is\_outside\_enclave(buf,\ max)$). Moreover, the maximal size is always checked before any write to such output buffers. The concerned parameters of $Exec()$ are buffers for newly created and modified account state objects.
- **Input Parameters:** On the other hand, in the case of input parameters of $Exec()$, we utilize embedded buffering provided by trts of SDK since $\mathbb{E}$ has to check the integrity of input parameters before using them, otherwise Time-of-Check != Time-of-Use vulnerability [CRD20] might be possible. The concerned input parameters of $Exec()$ are transactions to process and their corresponding codes.

Time to Finality. Many blockchain platforms suffer from accidental forks, which temporarily create parallel inconsistent blockchain views. To mitigate this phenomenon, it is recommended to wait a certain number of block confirmations after a given block is created before considering it irreversible with overwhelming probability. This waiting time (a.k.a., time to finality) influences the non-equivocation property of Aquareum, and Aquareum inherits it from the underlying blockchain platform. Most blockchains have a long time to finality, e.g., $\sim$10mins in Bitcoin [Nak08], $\sim$3mins in Ethereum [Woo14a], $\sim$2mins in Cardano [KRDO17]. However, some blockchains have a short time to finality, e.g., HoneyBadgerBFT [MXC$^+$16], Algorand [GHM$^+$17], and StrongChain [SRHS19]. The selection of the underlying blockchain platform is dependent on the requirements of the particular use case that Aquareum is applied for.

7.2 CBDC-AquaSphere

For background related to blockchains, integrity-preserving data structures, atomic swap, and CBDC, we refer the reader to Chapter 2.

7.2.1 Problem Definition

Our goal is to propose a CBDC approach that respects the features proposed in DEA manifesto [Dig22] released in 2022, while on top of it, we assume other features that might bring more benefits and guarantees. First, we start with a specification of the desired features related to a single instance of CBDC that we assume is operated by a single entity (further a bank or its operator) that maintains its ledger. Later, we describe desired features related to multiple instances of CBDC that co-exist in the ecosystem of wholesale and/or retail CBDC.[4] In both cases, we assume that a central bank might not be a trusted entity. All features that respect this assumption are marked with asterisk * and are considered as requirements for such an attacker model.

Single Instance of CBDC

When assuming a basic building block of CBDC – a single bank's CBDC working in an isolated environment from the other banks – we specify the desired features of CBDC as follows:

Correctness of Operation Execution*: The clients who are involved in a monetary operation (such as a transfer) should be guaranteed with a correct execution of their operation.

Integrity*: The effect of all executed operations made over the client accounts should be irreversible, and no "quiet" tampering of the data by a bank should be possible. Also, no conflicting transactions can be (executed and) stored by the CBDC instance in its ledger.

Verifiability*: This feature extends integrity and enables the clients of CBDC to obtain easily verifiable evidence that the ledger they interact with is internally correct and consistent. In particular, it means that none of the previously inserted transactions was neither modified nor deleted.

Non-Equivocation*: From the perspective of the client's security, the bank should not be able to present at least two inconsistent views on its ledger to (at least) two distinct clients who would accept such views as valid.

Censorship Evidence*: The bank should not be able to censor a client's request without leaving any public audit trails proving the censorship occurrence.

Transparent Token Issuance*: Every CBDC-issued token should be publicly visible (and thus audit-able) to ensure that a bank is not secretly creating token value "out-

[4] Note that we will propose two deployment scenarios, one for the wholesale environment and the second one for the retail environment of multiple retail banks interacting with a single central bank.

of-nothing," and thus causing uncontrolled inflation. The transparency also holds for burning of existing tokens.

High Performance: A CBDC instance should be capable of processing a huge number of transactions per second since it is intended for daily usage by thousands to millions of people.

Privacy: All transfers between clients as well as information about the clients of CBDC should remain private for the public and all other clients that are not involved in particular transfers. However, a bank can access this kind of information and potentially provide it to legal bodies, if requested.

Multiple Instances of CBDC

In the case of multiple CBDC instances that can co-exist in a common environment, we extend the features described in the previous listing by features that are all requirements:

Interoperability*: As a necessary prerequisite for co-existence of multiple CBDC instances, we require them to be mutually interoperable, which means that tokens issued by one bank can be transferred to any other bank. For simplicity, we assume that all the CBDC instances are using the unit token of the same value within its ecosystem.[5] At the hearth of interoperability lies atomicity of supported operations. Atomic interoperability, however, requires means for accountable coping with censorship and recovery from stalling. We specify these features in the following.

- **Atomicity*:** Any operation (e.g., transfer) between two interoperable CBDC instances must be either executed completely or not executed at all. As a consequence, no new tokens can be created out-of-nothing and no tokens can be lost in an inter-bank operation. Note that even if this would be possible, the state of both involved instances of CBDC would remain internally consistent; therefore, consistency of particular instances (Section 7.2.1) is not a sufficient feature to ensure atomicity within multiple interoperable CBDC instances. This requirement is especially important due to trustless assumption about particular banks, who might act in their benefits even for the cost of imposing the extreme inflation to the whole system.[6]
- **Inter-CBDC Censorship Evidence*:** Having multiple instances of CBDC enables a different way of censorship, where one CBDC (and its clients) might be censored within some inter-CBDC operation with another CBDC instance, precluding them to finish the operation. Therefore, there should exist a means how to accountably detect this kind of censorship as well.
- **Inter-CBDC Censorship Recovery*:** If the permanent censorship happens and is indisputably proven, it must not impact other instances of CBDC, including the ones that the inter-CBDC operations are undergoing. Therefore, the interoper-

[5] On the other hand, conversions of disparate CBDC-backed tokens would be possible by following trusted oracles or oracle networks.

[6] For example, if atomicity is not enforced, one bank might send the tokens to another bank, while not decreasing its supply due to pretended operation abortion.

able CBDC environment should provide a means to recover from inter-CBDC censorship of unfinished operations.

Identity Management of CBDC Instances*: Since we assume that CBDC instances are trustless, in theory, there might emerge a fake CBDC instance, pretentding to act as a valid one. To avoid this kind of situation, it is important for the ecosystem of wholesale CBDC to manage identities of particular valid CBDC instances in a secure manner.

Adversary Model

The attacker can be represented by the operator of a bank or the client of a bank, and her intention is to break functionalities that are provided by the features described above. Next, we assume that the adversary cannot undermine the cryptographic primitives used, the blockchain platform, and the TEE platform deployed.

7.2.2 Proposed Approach

We propose a holistic approach for the ecosystem of wholesale and/or retail CBDC, which aims at meeting the features described in Section 7.2.1. To accomplish these features, we leverage interesting properties stemming from a combination of a public blockchain (with smart contract platform) and TEE. Such a combination was proposed for various purposes in related work, out of which the use case of generic centralized ledger Aquareum [HS20] is most convenient to build on. Therefore, we utilize Aquareum as a building block for a single instance of CBDC, and we make a few CBDC-specific modifications to it, enhancing its transparency and functionality. Our modifications are outlined in Figure 7.6 by red color, while the details of them (especially changes in programs of smart contract and enclave) will be described in this section. First, we start by a description of a single CBDC instance and then we extend it to a fully interoperable environment consisting of multiple CBDC instances.

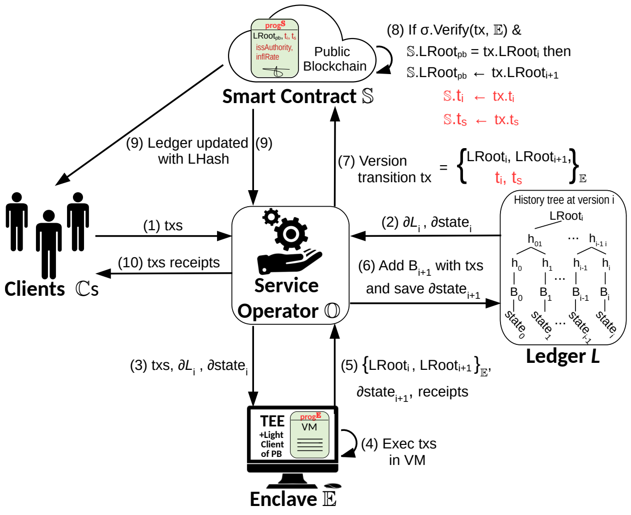

prog$^{\mathbb{S}}$
LRoot$_{pb}$, t_i, t_s
issAuthority, inflRate
Public Blockchain
Smart Contract $\mathbb{S}$
(8) If σ.Verify(tx, $\mathbb{E}$) &
$\mathbb{S}$.LRoot$_{pb}$ = tx.LRoot$_i$ then
$\mathbb{S}$.LRoot$_{pb}$ ← tx.LRoot$_{i+1}$
$\mathbb{S}.t_i$ ← tx.t_i
$\mathbb{S}.t_s$ ← tx.t_s
(9) Ledger updated with LHash
(9)
(7) Version transition tx = $\{$LRoot$_i$, LRoot$_{i+1}$, t_i, $t_s\}_{\mathbb{E}}$
Clients $\mathbb{C}$s
(1) txs
(10) txs receipts
Service Operator $\mathbb{O}$
(2) ∂L_i, ∂state$_i$
(6) Add B$_{i+1}$ with txs and save ∂state$_{i+1}$
History tree at version i
LRoot$_i$
h$_{01}$ … h$_{i-1\,i}$
h$_0$ h$_1$ … h$_{i-1}$ h$_i$
B$_0$ B$_1$ B$_{i-1}$ B$_i$
state$_0$ state$_1$ … state$_{i-1}$ state$_i$
Ledger L
(3) txs, ∂L_i, ∂state$_i$
(5) $\{$LRoot$_i$, LRoot$_{i+1}\}_{\mathbb{E}}$, ∂state$_{i+1}$, receipts
TEE +Light Client of PB
prog$^{\mathbb{E}}$
VM
(4) Exec txs in VM
Enclave $\mathbb{E}$

Figure 7.6: Architecture of Aquareum with our modifications in red.

Note that we focus solely on the transfer of tokens operation within the context of CBDC interoperability. However, our approach could be extended to different operations, involving inter-CBDC smart contract invocations. Also, note that to distinguish between smart contracts on a public blockchains and smart contracts running in TEE, we will denote latter as **micro contracts** (or μ-contracts). Similarly, we denote transactions sent to TEE as **micro transactions** (or μ-transactions) and blocks created in the ledger of CBDC instance as **micro blocks** (or μ-blocks).

A CBDC Instance

Alike in Aquareum, the primary entity of each CBDC instance is its operator $\mathbb{O}$ (i.e., a bank), who is responsible for (1) maintaining the ledger L, (2) running the TEE enclave $\mathbb{E}$, (3) synchronization of the L's snapshot to a public blockchain with smart contract $\mathbb{IPSC}$ ($\mathbb{I}$ntegrity $\mathbb{P}$reserving $\mathbb{S}$mart $\mathbb{C}$ontract), (4) resolving censorship requests, and (5) a communication with clients $\mathbb{C}$s.

Token Issuance. On top of Aquareum's $\mathbb{S}$, our $\mathbb{IPSC}$ contains snapshotting of the total issued tokens t_i by the current CBDC instance and the total supply t_s available at the instance for the purpose of transparency in token issuance (and potentially even burning). Therefore, we extend the $\mathbb{E}$-signed version transition pair periodically submitted to $\mathbb{IPSC}$ by these two fields that are relayed to $\mathbb{IPSC}$ upon snapshotting L (see red text in Figure 7.6). Notice that $t_i = t_s$ in the case of a single instance since the environment of the instance is isolated.

- **An Inflation Bound.** Although snapshotting the total tokens in circulation is useful for the transparency of token issuance, $\mathbb{O}$ might still hyper-inflate the CBDC instance. Therefore, we require $\mathbb{O}$ to guarantee a maximal inflation rate i_r per year, which can be enforced by $\mathbb{IPSC}$ as well as $\mathbb{E}$ since the code of both is publicly visible and attestable. The i_r should be adjusted to a constant value by $\mathbb{O}$ at the initialization of $\mathbb{IPSC}$ and verified every time the new version of L is posted to $\mathbb{IPSC}$; in the case of not meeting the constrain, the new version would not be accepted at $\mathbb{IPSC}$. However, another possible option is that the majority vote of $\mathbb{C}$s can change i_r even after initialization. Besides, $\mathbb{E}$ also enforces i_r on t_i and does not allow $\mathbb{O}$ to issue yearly more tokens than defined by i_r. Nevertheless, we put the inflation rate logic also into $\mathbb{IPSC}$ for the purpose of transparency.

Initialization. First, $\mathbb{E}$ with program $prog^{\mathbb{E}}$ (see Algorithm 12 of Appendix) generates and stores two key pairs, one under Σ_{pb} (i.e., $SK^{pb}_{\mathbb{E}}$, $PK^{pb}_{\mathbb{E}}$) and one under Σ_{tee} (i.e., $SK^{tee}_{\mathbb{E}}$, $PK^{tee}_{\mathbb{E}}$). Then, $\mathbb{O}$ generates one key pair under Σ_{pb} (i.e., $SK^{pb}_{\mathbb{O}}$, $PK^{pb}_{\mathbb{O}}$), which is then used as the sender of a transaction deploying $\mathbb{IPSC}$ with program $prog^{\mathbb{IPSC}}$ (see Algorithm 15 of Appendix) at public blockchain with parameters $PK^{pb}_{\mathbb{E}}$, $PK^{tee}_{\mathbb{E}}$, $PK^{pb}_{\mathbb{O}}$, t_i, and i_r. Then, $\mathbb{IPSC}$ stores the keys in parameters, sets the initial version of L by putting

$LRoot_{pb} \leftarrow \perp$, and sets the initial total issued tokens and the total supply, both to t_i.[7]

- **Client Registration.** A client $\mathbb{C}$ registers with $\mathbb{O}$, who performs know your customer (KYC) checks and submits her public key $PK^{\mathbb{C}}_{pb}$ to $\mathbb{E}$. Then, $\mathbb{E}$ outputs an execution receipt about the successful registration of $\mathbb{C}$ as well as her access ticket $t^{\mathbb{C}}$ that will serve for potential communication with $\mathbb{IPSC}$ and its purpose is to avoid spamming $\mathbb{IPSC}$ by invalid requests. In detail, $t^{\mathbb{C}}$ is the $\mathbb{E}$-signed tuple that contains $PK^{\mathbb{C}}_{pb}$ and optionally other fields such as the account expiration timestamp. Next, $\mathbb{C}$ verifies whether her registration (proved by the receipt) was already snapshotted by $\mathbb{O}$ at $\mathbb{IPSC}$.

Normal Operation. $\mathbb{C}$s send *μ-transactions* (writing to L) and *queries* (reading from L) to $\mathbb{O}$, who validates them and relays them to $\mathbb{E}$, which processes them within its virtual machine (Aquareum uses eEVM [Mic20]). Therefore, L and its state are modified in a trusted code of $\mathbb{E}$, creating a new version of L, which is represented by the root hash $LRoot$ of the history tree. Note that program $prog^{\mathbb{E}}$ is public and can be remotely attested by $\mathbb{C}$s (or anybody). $\mathbb{O}$ is responsible for a periodic synchronization of the most recent root hash $LRoot_{cur}$ (i.e., snapshotting the current version of L) to $\mathbb{IPSC}$, running on a public blockchain PB. Besides, $\mathbb{C}$s use this smart contract to resolve censored transactions and queries, while preserving the privacy of data.

Censorship Resolution. $\mathbb{O}$ might potentially censor some *write* transactions or *read* queries of $\mathbb{C}s$. However, these can be resolved by Aquareum's mechanism as follows. If $\mathbb{C}$'s μ-transaction μ-tx is censored by $\mathbb{O}$, $\mathbb{C}$ first creates $PK^{tee}_{\mathbb{E}}$-encrypted $\mu\text{-}etx$ (to ensure privacy in PB), and then she creates and signs a transaction containing $\mathbb{C}'s$ access ticket $t^{\mathbb{C}}$ and $\mu\text{-}etx$. $\mathbb{C}$ sends this transaction to $\mathbb{IPSC}$, which verifies $t^{\mathbb{C}}$ and stores $\mu\text{-}etx$, which is now visible to $\mathbb{O}$ and the public. Therefore, $\mathbb{O}$ might relay $\mu\text{-}etx$ to $\mathbb{E}$ for processing and then provide $\mathbb{E}$-signed execution receipt to $\mathbb{IPSC}$ that publicly resolves this censorship request. On the other hand, if $\mathbb{O}$ were not to do it, $\mathbb{IPSC}$ would contain an indisputable proof of censorship by $\mathbb{O}$ on a client $\mathbb{C}$.

Multiple CBDC Instances

The conceptual model of our interoperable CBDC architecture is depicted in Figure 7.7. It consists of multiple CBDC instances (i.e., at least two), whose $\mathbb{C}s$ communicate in three different ways: (1) directly with each other, (2) in the instance-to-instance fashion through the infrastructure of their $\mathbb{O}$ as well as counterpart's $\mathbb{O}$, (3) through PB with $\mathbb{IPSC}$ of both $\mathbb{O}$s and a global registry $\mathbb{IMSC}$ managing identities of instances.

For simplified description, in the following we assume the transfer operation where a local CBDC instance in Figure 7.7 is A (i.e., the sender of tokens) and the external one

[7] Among these parameters, a constructor of $\mathbb{IPSC}$ also accepts the indication whether an instance is allowed to issue tokens. This is, however, implicit for the single instance, while restrictions are reasonable in the case of multiple instances.

Algorithm 15: The program $prog^{\mathbb{IPSC}}$ of $\mathbb{IPSC}$ with our modifications in red (as opposed to Aquareum).

```
▷ DECLARATION OF TYPES AND CONSTANTS:
    CensInfo { $\mu$-$etx$,$\mu$-$equery$,$status$,$edata$ },
    $msg$: a current transaction that called $\mathbb{IPSC}$,
▷ DECLARATION OF FUNCTIONS:
function $Init(PK_{\mathbb{E}}^{pb}, PK_{\mathbb{E}}^{tee}, PK_{\mathbb{O}}, \_i_r, [ia \leftarrow T])$ public
    $PK_{\mathbb{E}}^{tee}[].add(PK_{\mathbb{E}}^{tee})$;                ▷ PK of enclave $\mathbb{E}$ under $\Sigma_{tee}$.
    $PK_{\mathbb{E}}^{pb}[].add(PK_{\mathbb{E}}^{pb})$;                  ▷ PK of enclave $\mathbb{E}$ under $\Sigma_{pb}$.
    $PK_{\mathbb{O}}^{pb} \leftarrow PK_{\mathbb{O}}$;                   ▷ PK of operator $\mathbb{O}$ under $\Sigma_{pb}$.
    $LRoot_{pb} \leftarrow \perp$;      ▷ The most recent root hash of $L$ synchronized with $\mathbb{IPSC}$.
    $censReqs \leftarrow []$;                  ▷ Request that $\mathbb{C}$s wants to resolve publicly.
    $t_s \leftarrow 0$;                                  ▷ The total supply of the instance.
    $t_i \leftarrow 0$;                          ▷ The total issued tokens by the instance.
    const $issueAuthority \leftarrow ia$;      ▷ Token issuance capability of the instance.
    const $i_r \leftarrow \_i_r$;                    ▷ Max. yearly inflation of the instance.
    const $createdAt \leftarrow timestamp()$;   ▷ The timestamp of creation a CBDC instance.
function $snapshotLedger(root_A, root_B, \_t_i, \_t_s, \sigma)$ public
                                ▷ Verify whether msg was signed by $\mathbb{E}$.
    assert $\Sigma_{pb}.verify((\sigma, PK_{\mathbb{E}}^{pb}$[-1]$), (root_A, root_B, \_t_i, \_t_s))$;
                                ▷ Snapshot issued tokens and total supply.
    if $issueAuthority$ then
        assert $\_meetsInflationRate(\_t_i)$;          ▷ The code is trivial, and we omit it.
        $t_i \leftarrow \_t_i$;
    else
        assert $t_i = \_t_i$;
                       ▷ Verify whether a version transition extends the last one.
    if $LRoot_{pb} = root_A$ then
        $LRoot_{pb} \leftarrow root_B$;                            ▷ Do a version transition of $L$.
function $SubmitCensTx(\mu\text{-}etx, \sigma_{msg})$ public
                 ▷ Called by $\mathbb{C}$ in the case her $\mu$-tx is censored. $\mathbb{C}$ encrypts it by $PK_{\mathbb{E}}^{tee}$.
    accessControl($\sigma_{msg}$,$msg.PK_{\mathbb{C}}^{pb}$);
    $censReqs$.add(CensInfo($\mu$-$etx$,$\perp$,$\perp$,$\perp$));
function $ResolveCensTx(idx_{req}, status, \sigma)$ public
                 ▷ Called by $\mathbb{O}$ to prove that $\mathbb{C}$'s $\mu$-tx was processed.
    assert $idx_{req} < |censReqs|$;
    $r \leftarrow censReqs[idx_{req}]$;
    assert $\Sigma_{pb}.verify((\sigma, PK_{\mathbb{E}}^{pb}$[-1]$), (h(r.\mu\text{-}etx), status))$;
    $r.status \leftarrow status$;
function $SubmitCensQry(\mu\text{-}equery, \sigma_{msg})$ public
                 ▷ Called by $\mathbb{C}$ in the case its read query is censored. $\mathbb{C}$ encrypts it by $PK_{\mathbb{E}}^{tee}$.
    accessControl($msg$, $\sigma_{msg}$,$msg.PK_{\mathbb{C}}^{pb}$);
    $censReqs$.add(CensInfo($\perp$,$\mu$-$equery$,$\perp$,$\perp$));
function $ResolveCensQry(idx_{req}, status, edata, \sigma)$ public
                 ▷ Called by $\mathbb{O}$ as a response to the $\mathbb{C}$'s censored read query.
    assert $idx_{req} < |censReqs|$;
    $r \leftarrow censReqs[idx_{req}]$;
    assert $\Sigma_{pb}.verify((\sigma, PK_{\mathbb{E}}^{pb}$[-1]$), (h(r.\mu\text{-}equery), status, h(edata)))$;
    $r.\{edata \leftarrow edata, status \leftarrow status\}$;
function $ReplaceEnc(PKN_{\mathbb{E}}^{pb}, PKN_{\mathbb{E}}^{tee}, r_A, r_B, \_t_i, \_t_s, \sigma, \sigma_{msg})$ public
                 ▷ Called by $\mathbb{O}$ in the case of enclave failure.
    assert $\Sigma_{pb}.verify((\sigma_{msg}, PK_{\mathbb{O}}^{pb}), msg)$;       ▷ Avoiding MiTM attack.
    $snapshotLedger(r_A, r_B, \_t_i, \_t_s, \sigma)$ ;                 ▷ Do a version transition.
    $PK_{\mathbb{E}}^{tee}.add(PKN_{\mathbb{E}}^{tee})$;          ▷ Upon change, $\mathbb{C}$s make remote attestation.
    $PK_{\mathbb{E}}^{pb}.add(PKN_{\mathbb{E}}^{pb})$;
```

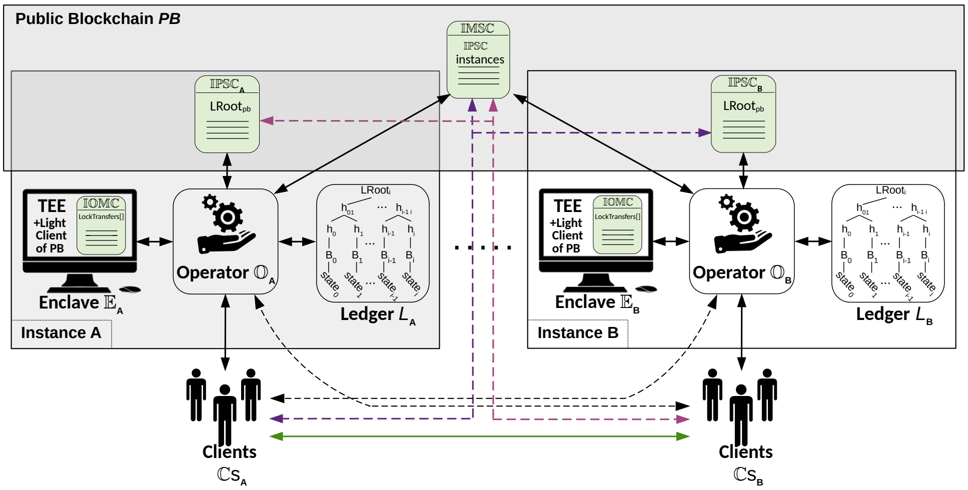


Figure 7.7: Overview of our CBDC architecture supporting interoperability among multiple CBDC instances (i.e., banks). The schema depicts two instances, where each of them has its own centralized ledger L modified in a secure way through TEE of $\mathbb{E}$, while its integrity is ensured by periodic integrity snapshots to the integrity preserving smart contract ($\mathbb{IPSC}$) in a public blockchain PB. Each CBDC instance is registered in the identity management smart contract $\mathbb{IMSC}$ of a public blockchain, serving as a global registry of bank instances. A client who makes an inter-bank transfer communicates with her bank and the counter-party bank utilizing interoperability micro contracts ($\mathbb{IOMC}$), running in the TEE. Any censored request of a client is resolved by $\mathbb{IPSC}$ of a particular bank and can be initiated by its client or a counter-party client.

is B (i.e., the receiver of tokens). To ensure interoperability, we require a communication channel of local clients $\mathbb{C}s_A$ to external clients $\mathbb{C}s_B$ (the green arrow), the local operator $\mathbb{O}_A$ (the black arrow), and the external operator $\mathbb{O}_B$ (the black dashed arrow). In our interoperability protocol Π^T (described later in Section 28), external $\mathbb{C}s_B$ use the channel with the local operator $\mathbb{O}_A$ only for obtaining incremental proofs of L_A's history tree to verify inclusion of some μ-transactions in L_A. However, there might arise a situation in which $\mathbb{O}_A$ might censor such queries, therefore, we need to address it by another communication channel – i.e., the public blockchain PB.

- **Censorship of External Clients.** We allow external clients $\mathbb{C}s_B$ to use the same means of censorship resolution as internal clients of a single CBDC instance (see Section 53). To request a resolution of a censored query, the external $\mathbb{C}_B$ uses the access ticket $t^{\mathbb{C}_B}$ at $\mathbb{IPSC}_A$, which is issued by $\mathbb{E}_A$ in the first phase of Π^T.
- **Identification of Client Accounts** To uniquely identify $\mathbb{C}$'s account at a particular CBDC instance, first it is necessary to specify the globally unique identifier of the CBDC instance. The best candidate is the blockchain address of the $\mathbb{IPSC}$ in PB since it is publicly visible and unique in PB (and we denote it by $\mathbb{IPSC}$). Then, the identification of $\mathbb{C}$'s relevant account is a pair $\mathbb{C}^{ID} = \{PK^{\mathbb{C}}_{pb} || \ \mathbb{IPSC}\}$. Note that $\mathbb{C}$ might use the same $PK^{\mathbb{C}}_{pb}$ for the registration at multiple CBDC instances (i.e., equivalent of having accounts in multiple banks); however, to preserve better privacy, making linkage of $\mathbb{C}$'s instances more difficult, we recommend $\mathbb{C}s$ to have dedicated key pair for each instance.

Identity Management of CBDC Instances. To manage identities of all CBDC instances in the system, we need a global registry of their identifiers ($\mathbb{IPSC}$ addresses). For this purpose, we use the $\mathbb{I}$dentity $\mathbb{M}$anagement $\mathbb{S}$mart $\mathbb{C}$ontract ($\mathbb{IMSC}$) deployed in PB (see program $prog^{\mathbb{IMSC}}$ in Algorithm 16). We propose $\mathbb{IMSC}$ to be managed in either decentralized or centralized fashion, depending on the **deployment scenario**:

- **Decentralized Scheme** In the decentralized scheme, the enrollment of a new CBDC instance must be approved by a majority vote of the already existing instances. This might be convenient for interconnecting central banks from various countries/regions. The enrollment requires creating a request entry at $\mathbb{IMSC}$ (i.e., $newJoinRequest()$) by a new instance specifying the address of its $\mathbb{IPSC}_{new}$ and $PK_{PB}^{\mathbb{O}_{new}}$. Then, the request has to be approved by voting of existing instances. Prior to voting (i.e., $approveJoinRequest()$), the existing instances should first verify a new instance by certain legal processes as well as by technical means: do the remote attestation of $prog^{\mathbb{E}}_{new}$, verify the inflation rate i_r and the initial value of total issued tokens t_i in $\mathbb{IPSC}$, etc. Removing of the existing instance also requires the majority of all instances, who should verify legal conditions prior to voting.
- **Centralized Scheme** So far, we were assuming that CBDC instances are equal, which might be convenient for interconnection of central banks from different countries. However, from the single-country point-of-view, there usually exist only one central bank, which might not be interested in decentralization of its competences (e.g., issuing tokens, setting inflation rate) among multiple commercial banks. We respect this and enable our approach to be utilized for such a use case, while the necessary changes are made to $\mathbb{IMSC}_c$ (see Algorithm 17), allowing to have only one CBDC authority that can add or delete instances of (commercial) banks, upon their verification (as outlined above). The new instances can be adjusted even with token issuance capability and constraints on inflation, which is enforced within the code of $\mathbb{E}$ as well as $\mathbb{IPSC}$.

Token Issuance. With multiple CBDC instances, $\mathbb{C}$s and the public can obtain the total value of issued tokens in the ecosystem of CBDC and compare it to the total value of token supply of all instances. Nevertheless, assuming only two instances A and B, the value of t_s snapshotted by $\mathbb{IPSC}_A$ might not reflect the recently executed transfers to instance B that might have already made the snapshot of its actual L_B version to $\mathbb{IPSC}_B$, accounting for the transfers. As a consequence, given a set of instances, the value of the aggregated t_s should always be greater or equal than the corresponding sum of t_i:

$$t_i^A + t_i^B \quad \leq \quad t_s^A + t_s^B. \tag{7.1}$$

We can generalize it for N instances known by $\mathbb{IMSC}$ as follows:

$$\sum_{\forall X \,\in\, \mathbb{IMSC}} t_i^X \quad \leq \quad \sum_{\forall X \,\in\, \mathbb{IMSC}} t_s^X. \tag{7.2}$$

Inflation Rate. In contrast to a single CBDC instance, multiple independent instances must provide certain guarantees about inflation not only to their clients, but also to each

Algorithm 16: $prog_d^{\mathbb{IMSC}}$ of decentralized $\mathbb{IMSC}$

```
▷ DECLARATION OF TYPES AND VARIABLES:
  $msg$: a current transaction that called $\mathbb{IMSC}$,
  struct InstanceInfo {
      $operator : PK_{\mathbb{O}}^{PB}$ of the instance's $\mathbb{O}$,
      $isApproved$: admission status of the instance,
      $approvals \leftarrow []$ : $\mathbb{O}$s who have approved the instance creation (or deletion),
  }
  $instances[]$: a mapping of $\mathbb{IPSC}$ to InstanceInfo,
▷ DECLARATION OF FUNCTIONS:
function $Init(\mathbb{IPSC}s[], \mathbb{O}s[])$ public        ▷ Initial instances are implicitly approved.
    assert $|\mathbb{IPSC}s| = |\mathbb{O}s|$ ;
    for $i \leftarrow 0;\ i \leq |\mathbb{O}s|;\ i \leftarrow i + 1$ do
        $instances[\mathbb{IPSC}s[i]] \leftarrow$ InstanceInfo($\mathbb{O}s[i], True, []$);

function $newJoinRequest(\mathbb{IPSC})$ public
    assert $instances[\mathbb{IPSC}] = \perp$;        ▷ The instance must not exist yet.
    $instances[\mathbb{IPSC}] \leftarrow$ InstanceInfo($msg.sender, False, []$);

function $approveJoinRequest(\mathbb{IPSC}_{my}, \mathbb{IPSC}_{new})$ public
    assert $instances[\mathbb{IPSC}_{my}].operator = msg.sender$;        ▷ Sender's check.
    assert $instances[\mathbb{IPSC}_{my}].isApproved$;        ▷ The sending $\mathbb{O}$ has valid instance.
    assert $!instances[\mathbb{IPSC}_{new}].isApproved$;        ▷ The new instance is not approved.
    $r \leftarrow instances[\mathbb{IPSC}_{new}]$;
    $r.approvals[msg.sender] \leftarrow True$;        ▷ The sender acknowledges the request.
    if $|r.approvals| > \lfloor |instances|/2 \rfloor$ then
        $r.isApproved \leftarrow True$;        ▷ Majority vote applies.
        $r.approvals \leftarrow []$;        ▷ Switch this field for a potential deletion.

function $approveDelete(\mathbb{IPSC}_{my}, \mathbb{IPSC}_{del})$ public
    assert $instances[\mathbb{IPSC}_{my}].operator = msg.sender$;        ▷ Sender's check.
    assert $instances[\mathbb{IPSC}_{my}].isApproved$;        ▷ The sending $\mathbb{O}$ has valid instance.
    assert $instances[\mathbb{IPSC}_{del}].isApproved$;        ▷ An instance to delete must be approved.
    $r \leftarrow instances[\mathbb{IPSC}_{del}]$;
    $r.approvals[msg.sender] \leftarrow True$;        ▷ The sender acknowledges the request.
    if $|r.approvals| > \lfloor |instances|/2 \rfloor$ then
        delete $r$;
```

other. For this purpose, the parameter inflation rate i_r is adjusted to a constant value in the initialization of $\mathbb{IPSC}$ and checked before the instance is approved at $\mathbb{IMSC}$.

If one would like to enable the update of i_r at CBDC instances, a majority vote at $\mathbb{IMSC}$ on a new value could be utilized (or just the vote of authority in the case of centralized scenario). Nevertheless, to support even fairer properties, $\mathbb{C}$s of a particular instance might vote on the value of i_r upon its acceptance by $\mathbb{IOMC}$ and before it is propagated to $\mathbb{IPSC}$ of an instance. Then, based on the new value of $\mathbb{IPSC}.i_r$, $\mathbb{E}.i_r$ can be adjusted as well (i.e., upon the validation by the light client of $\mathbb{E}$). However, the application of such a mechanism might depend on the use case, and we state it only as a possible option that can be enabled in our approach.

Interoperability. The interoperability logic itself is provided by our protocol Π^T that utilizes $\mathbb{I}$nter$\mathbb{O}$perability $\mathbb{M}$icro $\mathbb{C}$ontracts $\mathbb{IOMC}^S$ and $\mathbb{IOMC}^R$, which serve for sending and receiving tokens, respectively. Therefore, in the context of $\mathbb{E}$-isolated environment these μ-contracts allow to mint and burn tokens, reflecting the changes in t_s after sending

Algorithm 17: $prog_c^{\mathbb{IMSC}}$ of centralized $\mathbb{IMSC}$

```
▷ DECLARATION OF TYPES AND VARIABLES:
  $msg$: a current transaction that called $\mathbb{IMSC}$,
  $authority$: $\mathbb{IPSC}$ of the authority bank,
  $authority^{\mathbb{O}}$: $PK_{pb}^{\mathbb{O}}$ of $\mathbb{O}$ at authority bank,
  $instances[]$: a mapping of $\mathbb{IPSC}$ to $PK_{\mathbb{O}}^{PB}$,
▷ DECLARATION OF FUNCTIONS:
function $Init(\mathbb{IPSC})$ public                    ▷ Initial instances are implicitly approved.
    $authority^{\mathbb{O}} \leftarrow msg.sender$;
    $authority \leftarrow \mathbb{IPSC}$;
function $addInstance(\mathbb{IPSC}_{new}, \mathbb{O}_{new})$ public
    assert $msg.sender = authority^{\mathbb{O}}$ ;           ▷ Only the authority can add instances.
    assert $instances[\mathbb{IPSC}_{new}] = \perp$;           ▷ The instance must not exist yet.
    $instances[\mathbb{IPSC}_{new}] \leftarrow \mathbb{O}_{new}$;
function $delInstance(\mathbb{IPSC}_{del})$ public
    assert $msg.sender = authority^{\mathbb{O}}$ ;           ▷ Only the authority can delete instances.
    delete $instances[\mathbb{IPSC}_{del}]$;
```

or receiving tokens between CBDC instances. Both μ-contracts are deployed in $\mathbb{E}$ by each $\mathbb{O}$ as soon as the instance is created, while $\mathbb{E}$ records their addresses that can be obtained and attested by $\mathbb{C}$s. We briefly review these contracts in the following, while they detailed usage will be demonstrated in Section 28.

- **The Sending** $\mathbb{IOMC}^S$ The sending $\mathbb{IOMC}^S$ (see Algorithm 18) is based on Hash Time LoCks (HTLC), thus upon initialization of transfer by $hashlock$ provided by $\mathbb{C}_A$ (i.e., $hashlock \leftarrow h(secret)$) and calling $sendInit(hashlock, \ldots)$, $\mathbb{IOMC}^S$ locks transferred tokens for the timeout required to complete the transfer by $sendCommit(secret, \ldots)$. If tokens are not successfully transferred to the recipient of the external instance during the timeout, they can be recovered by the sender (i.e., $sendRevert()$).[8] If tokens were sent successfully from $\mathbb{C}_A$ to $\mathbb{C}_B$, then instance A burns them within $sendCommit()$ of $\mathbb{IOMC}^S$ and deducts them from t_s. Note that deducting t_s is a special operation that cannot be executed within standard μ-contracts, but $\mathbb{IOMC}$ contracts are exceptions and can access some variables of $\mathbb{E}$.
- **The Receiving** $\mathbb{IOMC}^R$ The receiving $\mathbb{IOMC}^R$ (see Algorithm 19) is based on Hashlocks (referred to as HLC) and works pairwise with sending $\mathbb{IOMC}^S$ to facilitate four phases of our interoperable transfer protocol Π_T (described below). After calling $\mathbb{IOMC}^{\mathbb{S}}.sendInit()$, incoming initiated transfer is recorded at $\mathbb{IOMC}^R$ by $receiveInit(hashlock, \ldots)$. Similarly, after executing token deduction at instance A (i.e., $\mathbb{IOMC}^{\mathbb{S}}.sendCommit(secret, \ldots)$), incoming transfer is executed at $\mathbb{IOMC}^R$ by $receiveCommit(secret, \ldots)$ that mints tokens to $\mathbb{C}_B$ and increases t_s. Similar to $\mathbb{IOMC}^S$, minting tokens and increasing t_s are special operations requiring access to $\mathbb{E}$, which is exceptional for $\mathbb{IOMC}$. The overview of Π_T is depicted in Figure 7.8.

[8] Note that setting a short timeout might prevent the completion of the protocol.

Algorithm 18: $prog^{\mathbb{IOMC}^S}$ of sending $\mathbb{IOMC}^S$

```
▷ DECLARATION OF TYPES AND VARIABLES:
  $\mathbb{E}$,                                    ▷ The reference to $\mathbb{E}_A$ of sending party.
  $msg$,                                           ▷ The current $\mu$-transaction that called $\mathbb{IOMC}^S$.
  struct LockedTransfer {
      $sender$,                                    ▷ Sending client $\mathbb{C}_A$.
      $receiver$,                                  ▷ Receiving client $\mathbb{C}_B$.
      $receiver\mathbb{IPSC}$,                     ▷ The $\mathbb{IPSC}$ contract address of the receiver's instance.
      $amount$,                                    ▷ Amount of tokens sent.
      $hashlock$,                                  ▷ Hash of the secret of the sending $\mathbb{C}_A$.
      $timelock$,                                  ▷ A timestamp defining the end of validity of the transfer.
      $isCompleted$,                               ▷ Indicates whether the transfer has been completed.
      $isReverted$,                                ▷ Indicates whether the transfer has been canceled.
  },
  $transfers \leftarrow []$,                       ▷ Initiated outgoing transfers (i.e., LockedTransfer).
  const $timeout^{HTLC} \leftarrow 24h$,           ▷ Set the time lock for e.g., 24 hours.
▷ DECLARATION OF FUNCTIONS:
function $sendInit(receiver, receiver\mathbb{IPSC}, hashlock)$ public payable
    assert $msg.value > 0$;                        ▷ Checks the amount of tokens.
    $timelock \leftarrow timestamp.now() + timeout^{HTLC}$;
    $t \leftarrow$ LockedTransfer($msg.sender, receiver, receiver\mathbb{IPSC}$,
        $msg.value$, $hashlock, timelock, False, False$);   ▷ A new receiving transfer.
    $transfers.append(t)$;
    Output ("$sendInitialized$", $transferID \leftarrow |transfers| - 1$));

function $sendCommit(transferID, secret, extTransferID)$ public
    assert $transfers[transferID] \neq \perp$;     ▷ Check the existence of locked transfer.
    $t \leftarrow transfers[transferID]$;
    assert $t.hashlock = h(secret)$;               ▷ Check the secret.
    assert $!t.isCompleted \land !t.isReverted$;   ▷ Test if the transfer is still pending.
    $t.isCompleted \leftarrow True$;
    burn t.amount;                                 ▷ Burn tokens.
    $\mathbb{E}.t_s \leftarrow \mathbb{E}.t_s - t.amount$;   ▷ Decrease the total supply of the instance.
    Output ("$sendCommitted$", $transferID$, $extTransferID$, $t.receiver$, $t.receiver\mathbb{IPSC}$, $t.amount$);

function $sendRevert(transferID)$ public
    assert $transfers[transferID] \neq \perp$;     ▷ Check the existence of locked transfer.
    $t \leftarrow transfers[transferID]$;
    assert $!t.isCompleted \land !t.isReverted$;   ▷ Test the transfer is still pending.
    assert $t.timelock \leq timestamp.now()$;      ▷ Check the HTLC expiration.
    $transfer(t.amount, t.sender)$;                ▷ Returning tokens back to the sender.
    $t.isReverted \leftarrow True$;
    Output("$sendReverted$", $transferID$);
```

Interoperable Transfer Protocol $\Pi^{\mathbf{T}}$

In this section we outline our instance-to-instance interoperable transfer protocol Π^T for inter-CBDC transfer operation, which is inspired by the atomic swap protocol (see Section 2.10), but in contrast to the exchange-oriented approach of atomic swap, Π^T focuses only on one-way atomic transfer between instances of the custodial environment of CBDC, where four parties are involved in each transfer – a sending $\mathbb{C}_A$ and $\mathbb{O}_A$ versus a receiving $\mathbb{C}_B$ and $\mathbb{O}_B$. The goal of Π^T is to eliminate any dishonest behavior by $\mathbb{C}s$ or $\mathbb{O}$s that would incur token duplication or the loss of tokens.

Algorithm 19: $prog^{\mathbb{IOMC}^R}$ of receiving $\mathbb{IOMC}^R$

```
▷ DECLARATION OF TYPES AND VARIABLES:
    $\mathbb{E}$,                                    ▷ The reference to $\mathbb{E}_B$ of receiving party.
    struct LockedTransfer {
        $sender$,                                    ▷ Sending client $\mathbb{C}_A$.
        $sender\mathbb{IPSC}$,                       ▷ The IPSC contract address of the sender's instance.
        $receiver$,                                  ▷ Receiving client $\mathbb{C}_B$.
        $amount$,                                    ▷ Amount of transferred tokens.
        $hashlock$,                                  ▷ Hash of the secret of the sending $\mathbb{C}_A$.
        $isCompleted$,                               ▷ Indicates whether the transfer has been completed.
    },
    $transfers \leftarrow []$,                       ▷ Initiated incoming transfers (i.e., LockedTransfer).
▷ DECLARATION OF FUNCTIONS:
function $receiveInit(sender, sender\mathbb{IPSC}, hashlock, amount)$ public
    assert $amount > 0$;
    $t \leftarrow$ LockedTransfer($sender, sender\mathbb{IPSC}, msg.sender, amount,$
            $hashlock, False$);                      ▷ Make a new receiving transfer entry.
    $transfers.append(t)$;
    Output("$receiveInitialized$", $transferID \leftarrow |transfers| - 1$);
function $receiveCommit(transferID, secret)$ public
    assert $transfers[transferID] \neq \perp$;       ▷ Check the existence of transfer entry.
    $t \leftarrow transfers[transferID]$;
    assert $t.hashlock = h(secret)$;                 ▷ Check the secret.
    assert $!t.isCompleted$;                         ▷ Check whether the transfer is pending.
    $\mathbb{E}.mint(this, t.amount)$;               ▷ Call $\mathbb{E}$ to mint tokens on $\mathbb{IOMC}_R$.
    $\mathbb{E}.t_s \leftarrow \mathbb{E}.t_s + t.amount$ ;   ▷ Increase the total supply of the instance.
    $transfer(t.amount, t.receiver)$;                ▷ Credit tokens to the recipient.
    $t.isCompleted \leftarrow True$;
    Output("$receiveCommited$", $transferID$);
```

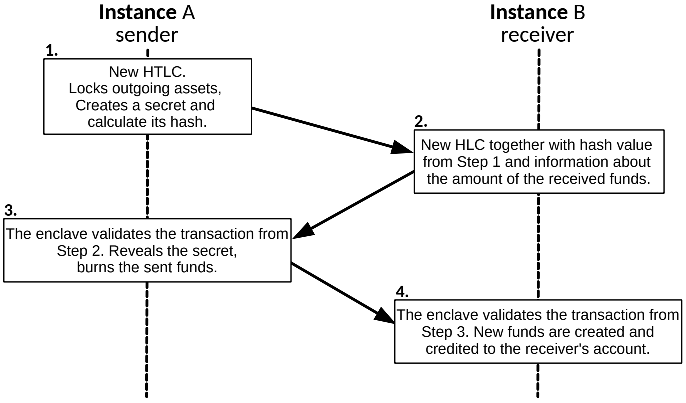
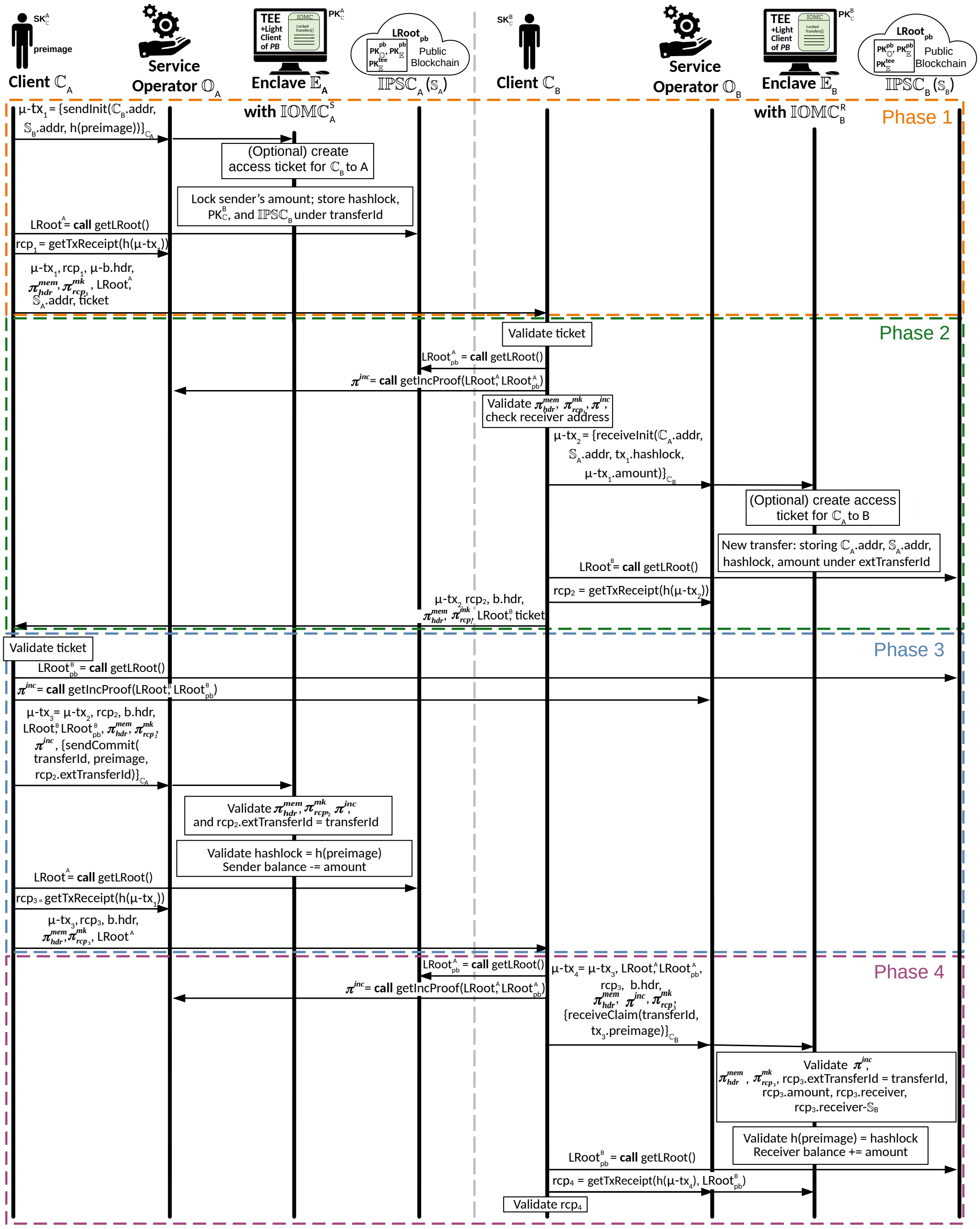

Instance A
sender
Instance B
receiver
1.
New HTLC.
Locks outgoing assets,
Creates a secret and
calculate its hash.
2.
New HLC together with hash value
from Step 1 and information about
the amount of the received funds.
3.
The enclave validates the transaction from
Step 2. Reveals the secret,
burns the sent funds.
4.
The enclave validates the transaction from
Step 3. New funds are created and
credited to the receiver's account.

Figure 7.8: Overview of the protocol Π^T, consisting of 4 phases.



$SK_{\mathbb{C}}^{A}$
preimage
Client $\mathbb{C}_A$
Service Operator $\mathbb{O}_A$
TEE +Light Client of PB
IOMC Locked Transfers[]
$PK_{\mathbb{C}}^{A}$
Enclave $\mathbb{E}_A$
with $\mathbb{IOMC}_A^S$
$LRoot_{pb}$
$PK_{\mathbb{O}}^{pb}$, $PK_{\mathbb{E}}^{pb}$ $PK_{\mathbb{E}}^{tee}$
Public Blockchain
$\mathbb{IPSC}_A$ ($\mathbb{S}_A$)
$SK_{\mathbb{C}}^{B}$
Client $\mathbb{C}_B$
Service Operator $\mathbb{O}_B$
TEE +Light Client of PB
IOMC Locked Transfers[]
$PK_{\mathbb{C}}^{B}$
Enclave $\mathbb{E}_B$
with $\mathbb{IOMC}_B^R$
$LRoot_{pb}$
$PK_{\mathbb{O}}^{pb}$, $PK_{\mathbb{E}}^{pb}$ $PK_{\mathbb{E}}^{tee}$
Public Blockchain
$\mathbb{IPSC}_B$ ($\mathbb{S}_B$)
Phase 1
μ-tx_1 = {sendInit($\mathbb{C}_B$.addr, $\mathbb{S}_B$.addr, h(preimage))}$_{\mathbb{C}_A}$
(Optional) create access ticket for $\mathbb{C}_B$ to A
Lock sender's amount; store hashlock, $PK_{\mathbb{C}}^{B}$, and $\mathbb{IPSC}_B$ under transferId
$LRoot^A$= call getLRoot()
rcp_1 = getTxReceipt(h(μ-tx_1))
μ-tx_1, rcp_1, μ-b.hdr, π_{hdr}^{mem}, $\pi_{rcp_1}^{mk}$, $LRoot^A$, $\mathbb{S}_A$.addr, ticket
Phase 2
Validate ticket
$LRoot_{pb}^A$ = call getLRoot()
π^{inc}= call getIncProof($LRoot^A$, $LRoot_{pb}^A$)
Validate π_{hdr}^{mem}, $\pi_{rcp_1}^{mk}$, π^{inc}, check receiver address
μ-tx_2 = {receiveInit($\mathbb{C}_A$.addr, $\mathbb{S}_A$.addr, tx_1.hashlock, μ-tx_1.amount)}$_{\mathbb{C}_B}$
(Optional) create access ticket for $\mathbb{C}_A$ to B
New transfer: storing $\mathbb{C}_A$.addr, $\mathbb{S}_A$.addr, hashlock, amount under extTransferId
$LRoot^B$= call getLRoot()
rcp_2 = getTxReceipt(h(μ-tx_2))
μ-tx_2, rcp_2, b.hdr, π_{hdr}^{mem}, $\pi_{rcp_2}^{mk}$, $LRoot^B$, ticket
Phase 3
Validate ticket
$LRoot_{pb}^B$ = call getLRoot()
π^{inc}= call getIncProof($LRoot^B$, $LRoot_{pb}^B$)
μ-tx_3= μ-tx_2, rcp_2, b.hdr, $LRoot^B$, $LRoot_{pb}^B$, π_{hdr}^{mem}, $\pi_{rcp_2}^{mk}$, π^{inc}, {sendCommit(transferId, preimage, rcp_2.extTransferId)}$_{\mathbb{C}_A}$
Validate π_{hdr}^{mem}, $\pi_{rcp_2}^{mk}$, π^{inc}, and rcp_2.extTransferId = transferId
Validate hashlock = h(preimage) Sender balance -= amount
$LRoot^A$= call getLRoot()
rcp_3 = getTxReceipt(h(μ-tx_1))
μ-tx_3, rcp_3, b.hdr, π_{hdr}^{mem}, $\pi_{rcp_3}^{mk}$, $LRoot^A$
Phase 4
$LRoot_{pb}^A$ = call getLRoot()
π^{inc}= call getIncProof($LRoot^A$, $LRoot_{pb}^A$)
μ-tx_4= μ-tx_3, $LRoot^A$, $LRoot_{pb}^A$, rcp_3, b.hdr, π_{hdr}^{mem}, π^{inc}, $\pi_{rcp_3}^{mk}$, {receiveClaim(transferId, tx_3.preimage)}$_{\mathbb{C}_B}$
Validate π^{inc}, π_{hdr}^{mem}, $\pi_{rcp_3}^{mk}$, rcp_3.extTransferId = transferId, rcp_3.amount, rcp_3.receiver, rcp_3.receiver-$\mathbb{S}_B$
Validate h(preimage) = hashlock Receiver balance += amount
$LRoot_{pb}^B$ = call getLRoot()
rcp_4 = getTxReceipt(h(μ-tx_4), $LRoot_{pb}^B$)
Validate rcp_4

Figure 7.9: The details of the proposed interoperability protocol Π^T.

To execute Π^T it is necessary to inter-connect $\mathbb{E}$s of two instances involved in a transfer. However, $\mathbb{E}$ does not allow direct communication with the outside world, and therefore it is necessary to use an intermediary. One solution is to involve $\mathbb{O}$s but they might be overwhelmed with other activities, updating the ledger by executing μ-transactions, and moreover, they might not have direct incentives to execute inter-CBDC transfers. Therefore, we argue that in contrast to the above option, involving $\mathbb{C}s$ as intermediaries has two advantages: (1) elimination of the synchronous communication overhead on $\mathbb{O}$s and (2) enabling $\mathbb{C}$s to have a transparent view about the status of the transfer and take action if required. The details of the protocol are depicted in Figure 7.9.

Phase 1 – Client $\mathbb{C}_A$ Initiates the Protocol. The client $\mathbb{C}_A$ creates a μ-tx$_1$ with the amount being sent, which invokes the `sendInit()` of $\mathbb{IOMC}_A$ with arguments containing the address of the external client $\mathbb{C}_B$, the address of $\mathbb{IPSC}_B$ (denoted as $\mathbb{S}_B$ in Figure 7.9 for brevity), and the hash of the secret that is created by $\mathbb{C}_A$. $\mathbb{C}_A$ sends signed μ-tx$_1$ to $\mathbb{O}_A$ who forwards it to the $\mathbb{E}_A$. Before executing the μ-tx$_1$, $\mathbb{E}_A$ ensures that the external recipient (i.e., $\mathbb{C}_B$) has the access ticket already issued and valid, enabling her to post censorship resolution requests to $\mathbb{IPSC}_A$ (if needed). The access ticket should be valid for at least the entire period defined by the HTLC of $\mathbb{IOMC}_A$. In the next step, a μ-tx$_1$ is executed by $\mathbb{E}_A$, creating a new transfer record with $transferId$ in $\mathbb{IOMC}_A$. During the execution, $\mathbb{C}_A$'s tokens are transferred (and thus locked) to the $\mathbb{IOMC}_A$'s address. $\mathbb{C}_A$ waits until the new version of L_A is snapshotted to $\mathbb{IPSC}_A$, and then obtains $LRoot^A$ from it. Then $\mathbb{C}_A$ asks $\mathbb{O}_A$ for the execution receipt rcp_1 of μ-tx$_1$ that also contains a set of proofs (π^{mem}_{hdr}, $\pi^{mk}_{rcp_1}$) and the header of the μ-block that includes μ-tx$_1$. In detail, π^{mem}_{hdr} is the inclusion proof of the μ-block b in the current version of L_A; $\pi^{mk}_{rcp_1}$ is the Merkle proof proving that rcp_1 is included in *b* (while rcp_1 proves that μ-tx$_1$ was executed correctly). The mentioned proofs and the receipt are provided to $\mathbb{C}_B$, who verifies that μ-tx$_1$ was executed and included in the L_A's version that is already snapshotted to $\mathbb{IPSC}_A$, thus irreversible (see below).

Phase 2 – $\mathbb{C}_B$ Initiates Receive. First, $\mathbb{C}_B$ validates an access ticket to $\mathbb{IPSC}_A$ using the enclave $\mathbb{E}_A$'s public key accessible in that smart contract. Next, $\mathbb{C}_B$ obtains the root hash $LRoot^A_{pb}$ of L_A to ensure that $\mathbb{C}_B$'s received state has been already published in $\mathbb{IPSC}_A$, and thus contains μ-tx$_1$. After obtaining $LRoot^A_{pb}$, $\mathbb{C}_B$ forwards it along with the root $LRoot^A$ obtained from $\mathbb{C}_A$ to $\mathbb{O}_A$, who creates an incremental proof π^{inc} of $\langle LRoot^A, LRoot^A_{pb}\rangle$. Once the proof π^{inc} has been obtained and validated, the protocol can proceed to validate the remaining proofs sent by the client $\mathbb{C}_A$ along with verifying that the receiving address belongs to $\mathbb{C}_B$. Next, $\mathbb{C}_B$ creates μ-tx_2, invoking the method `receiveInit()` with the arguments: the address of $\mathbb{C}_A$ obtained from μ-tx_1,[9] the address $\mathbb{IPSC}_A.addr$ of $\mathbb{C}_A$'s instance, the hash value of the secret, and the amount of crypto-tokens being sent. $\mathbb{C}_B$ sends μ-tx_2 to $\mathbb{O}_B$, who forwards it to $\mathbb{E}_B$ for processing. During processing of μ-tx_2, $\mathbb{E}_B$ determines whether the external client (from its point of

[9] Note that we assume that the address is extractable from the signature.

view – i.e., $\mathbb{C}_A$) has an access ticket issued with a sufficiently long validity period; if not, one is created. Subsequently, $\mathbb{E}_B$ creates a new record in $\mathbb{IOMC}_B$ with $extTransferId$. Afterward, $\mathbb{C}_B$ retrieves the $LRoot^B$ from L_B and requests the execution receipt rcp_2 from $\mathbb{O}_B$, acknowledging that the μ-tx_2 has been executed. Finally, $\mathbb{C}_B$ sends a message $\mathbb{C}_A$ with μ-tx_2 and cryptographic proofs π^{mem}_{hdr}, $\pi^{mk}_{rcp_2}$, the execution receipt of μ-tx_2, the block header b in which the μ-tx_2 was included, $LRoot^B$ (i.e., the root value of L_B after μ-tx_2 was executed), and the valid client access ticket for $\mathbb{C}_A$.

Phase 3 – Confirmation of Transfer by $\mathbb{C}_A$. First, $\mathbb{C}_A$ validates the received access ticket to $\mathbb{IPSC}_B$. Next, $\mathbb{C}_A$ obtains the snapshotted root hash $LRoot^B_{pb}$ of L_B from $\mathbb{IPSC}_B$. As in the previous phases, it is necessary to verify that the version of L_B that includes μ-tx_2 is represented by $LRoot^B_{pb}$ (thus is irreversible). Next, both root hashes ($LRoot^B$ and $LRoot^B_{pb}$) are sent to the external operator $\mathbb{O}_B$, which produces the incremental proof π^{inc} from them. Next, $\mathbb{C}_A$ creates μ-tx_3 that consists of invoking the `sendCommit()` method at $\mathbb{E}_A$ with the arguments containing the published secret (i.e., $preimage$) and the record identifier of the transfer at local instance (i.e., $transferId$) as well as the external one (i.e., $extTransferId$). Along with the invocation of `sendCommit()`, μ-tx_3 also wraps π^{inc} with its versions ($LRoot^B_{pb}$ and $LRoot^B$), μ-tx_2, its execution receipt rcp_2 with its Merkle proof $\pi^{mk}_{rcp_2}$, $b.hdr$ – the header of the block that included μ-tx_2, and its membership proof π^{mem}_{hdr} of L_B. Next, $\mathbb{C}_A$ sends μ-tx_3 to $\mathbb{E}_A$ through $\mathbb{O}_A$. During the execution of μ-tx_3, $\mathbb{E}_A$ validates the provided proofs and the equality of transfer IDs from both sides of the protocol. Note that to verify π^{mem}_{hdr}, $\mathbb{E}_A$ uses its light client to L_B. $\mathbb{E}_A$ then validates whether $\mathbb{C}_A$'s provided secret corresponds to the hashlock recorded in the 1st phase of the protocol, and if so, it burns the sent balance of the transfer.

Next, $\mathbb{C}_A$ waits until the new version of L_A is snapshotted to $\mathbb{IPSC}_A$, and then obtains $LRoot^A$ from it. Then $\mathbb{C}_A$ asks $\mathbb{O}_A$ for the execution receipt rcp_3 of μ-tx$_3$ that also contains a set of proofs (π^{mem}_{hdr}, $\pi^{mk}_{rcp_3}$) and the header of the μ-block that includes μ-tx$_3$. The proofs have the same interpretation as in the end of the 1st phase. The mentioned proofs and the receipt are provided to $\mathbb{C}_B$, who verifies that μ-tx$_1$ was executed and included in the L_A's version that is already snapshotted to $\mathbb{IPSC}_A$, thus irreversible.

Phase 4 – Acceptance of Tokens by $\mathbb{C}_B$. After receiving a message from client $\mathbb{C}_A$, the client $\mathbb{C}_B$ obtains $LRoot^A_{pb}$ from $\mathbb{IPSC}_A$ and then requests the incremental proof between versions $\langle LRoot^A, LRoot^A_{pb} \rangle$ from $\mathbb{O}_A$. Then, $\mathbb{C}_B$ creates μ-tx_4 invoking the `receiveClaim()` function at $\mathbb{E}_B$ with $transferId$ and the disclosed secret by $\mathbb{C}_A$ as the arguments. Moreover, μ-tx_4 contains remaining items received from $\mathbb{C}_A$. Then, μ-tx_4 is sent to $\mathbb{O}_B$, who forwards it to $\mathbb{E}_B$. During the execution of μ-tx_4, $\mathbb{E}_B$ verifies the provided proofs, the equality of transfer IDs from both sides of the protocol, the amount being sent, and the receiver of the transfer (i.e., $\mathbb{C}_B$ —— $\mathbb{IPSC}_B$). Note that to verify π^{mem}_{hdr}, $\mathbb{E}_B$ uses its light client to L_A. $\mathbb{E}_A$ then validates whether $\mathbb{C}_A$'s provided secret corresponds to the hashlock recorded in the 2nd phase of the protocol, and if so, it mints the sent balance of the transfer on the receiver's account $\mathbb{C}_B$. Finally, $\mathbb{C}_B$ verifies that

Function	constructor	sendInitialize	sendCommit	sendRevert
Deployment	901 509	160 698	64 629	60 923
Execution	653 689	134 498	42 717	39 523

Table 7.1: The cost of deployment and invocation of functions in the sending $\mathbb{IOMC}^S$ μ-contract in gas units (CBDC private ledger).

Function	constructor	receiveInit	receiveClaim	fund
Deployment	716 330	139 218	61 245	23 168
Execution	509 366	112 762	39 653	1 896

Table 7.2: The cost of deployment and invocation of functions in the receiving $\mathbb{IOMC}^R$ μ-contract in units of gas (CBDC private ledger).

μ-tx_4 was snapshotted at $\mathbb{IPSC}_B$, thus is irreversible. In detail, first $\mathbb{C}_B$ obtains $LRoot^B_{pb}$ from $\mathbb{IPSC}_B$ and then asks $\mathbb{O}_B$ to provide her with the execution receipt rcp_4 of μ-tx_4 in the version of L_B that is equal or newer than $LRoot^B_{pb}$. Then, $\mathbb{C}_B$ verifies rcp_4, which completes the protocol.

7.2.3 Evaluation

We used *Ganache*[10] and *Truffle*,[11] to develop $\mathbb{IOMC}$, $\mathbb{IPSC}$, and $\mathbb{IMSC}$ contracts. In addition, using the *Pexpect*[12] tool, we tested the intercommunication of the implemented components and validated the correctness of the implemented interoperability protocol. The tool enabled the parallel execution and control of numerous programs (in this case, multiple Aquareum instances and client programs) to check the correctness of the expected output.

The computational cost of executing the operations defined in $\mathbb{IOMC}$ and $\mathbb{IMSC}^X$ contracts is presented in Table 7.1, Table 7.2, and Table 7.3.[13] We optimized our implementation to minimize the storage requirements of smart contract platform. On the other hand, it is important to highlight that $\mathbb{IOMC}^X$ μ-contracts are executed on a private ledger corresponding to the instance of CBDC, where the cost of gas is minimal or negligible as compared to a public blockchain. Other experiments are the subject of our future work.

[10] `https://github.com/trufflesuite/ganache-cli`

[11] `https://github.com/trufflesuite/truffle`

[12] `https://github.com/pexpect/pexpect`

[13] Note that we do not provide the gas measurements for $\mathbb{IPSC}$ since these are almost the same as in Aquareum [HS20].

Function	constructor	newJoinRequest	approveRequest	isApproved
Deployment	830 074	48 629	69 642	0
Execution	567 838	25 949	45 554	0

Table 7.3: The invocation cost of functions in $\mathbb{IMSC}$ smart contract in units of gas (Ethereum public blockchain).

7.2.4 Security Analysis

In this section, we analyze our approach in terms of security-oriented features and requirements specified in Section 7.2.1. In particular, we focus on resilience analysis of our approach against adversarial actions that the malicious CBDC instance (i.e., its operator $\mathcal{O}$) or malicious client (i.e., $\mathcal{C}$) can perform to violate the security requirements.

Single Instance of CBDC

For properties common with Aquareum (i.e., correctness of operation execution, integrity, verifiability, non-equivocation, censorship evidence, and privacy) as well as for security analysis of TEE and time to finality, we refer the reader to Section 7.1.5.

Theorem 11. *(Transparent Token Issuance) $\mathcal{O}$ is unable to issue or burn any tokens without leaving a publicly visible evidence.*

Justification. All issued tokens of CBDC are publicly visible at $\mathbb{IPSC}$ since each transaction posting a new version transition pair also contains $\mathbb{E}$-signed information about the current total issued tokens t_i and total supply of the instance t_s,[14] while t_i was updated within the trusted code of $\mathbb{E}$. The information about t_i is updated at $\mathbb{IPSC}$ along with the new version of L. Note that the history of changes in total issued tokens t_i can be parsed from all transactions updating version of L published by $\mathcal{O}$ to PB. □

Multiple Instances of CBDC

In the following, we assume two CBDC instances A and B.

Theorem 12. *(Atomic Interoperability I) Neither $\mathcal{O}_A$ (operating A) nor $\mathcal{O}_B$ (operating B) is unable to steal any tokens during the inter-bank CBDC transfer.*

Justification. Atomic interoperability is ensured in our approach by adaptation of atomic swap protocol for all inter-bank transfers, which enables us to preserve the wholesale environment of CBDC in a consistent state (respecting Equation 7.2). In detail, the transferred tokens from CBDC instance A to instance B are not credited to B until A does

[14] Note that in the case of single CBDC instance $t_i = t_s$

not provide the indisputable proof that tokens were deducted from a relevant A's account. This proof confirms irreversible inclusion of tx_3 (i.e., $\mathbb{E}_A.sendCommit()$ that deducts account of A's client) in A's ledger and it is verified in 4th stage of our protocol by the trusted code of $\mathbb{E}_B$.

In the case that $\mathcal{O}_A$ would like to present B with integrity snapshot of L_A that was not synced to $\mathbb{IPSC}_A$ yet, B will not accept it since the 4th phase of our protocol requires $\mathcal{O}_B$ to fetch the recent $\mathbb{IPSC}_A.LRoot_{pb}$ and verify its consistency with A-provided $LRoot$ as well as inclusion proof in PB; all executed/verified within trusted code of $\mathbb{E}_B$. □

Theorem 13. *(Atomic Interoperability II) Colluding clients $\mathcal{C}_A$ and $\mathcal{C}_B$ of two CBDC instances cannot steal any tokens form the system during the transfer operation of our protocol.*

Justification. If the first two phases of our protocol have been executed, $\mathcal{C}_A$ might potentially reveal the $preimage$ to $\mathcal{C}_B$ without running the 3rd phase with the intention to credit the tokens at B while deduction at A had not been executed yet. However, this is prevented since the trusted code of $\mathbb{E}_B$ verifies that the deduction was performed at A before crediting the tokens to $\mathcal{C}_B$ – as described in Theorem 12. □

Theorem 14. *(Inter-CBDC Censorship Evidence) $\mathcal{O}_A$ is unable to unnoticeably censor any request (transaction or query) from $\mathbb{C}_B$.*

Justification. If $\mathbb{C}_B$'s request is censored by $\mathcal{O}_A$, $\mathbb{C}_B$ can ask for a resolution of the request through public $\mathbb{IPSC}_A$ since $\mathbb{C}_B$ already has the access ticket to instance A. The access ticket is signed by $\mathbb{E}_A$ and thus can be verified at $\mathbb{IPSC}_A$. Hence, the censorship resolution/evidence is the same as in Theorem 10 of a single CBDC instance. □

Theorem 15. *(Inter-CBDC Censorship Recovery) A permanent inter-CBDC censorship by $\mathcal{O}_A$ does not cause an inconsistent state or permanently frozen funds of undergoing transfer operations at any other CBDC instance – all initiated and not finished transfer operations can be recovered from.*

Justification. If $\mathcal{O}_A$ were to censor $\mathbb{C}_B$ in the 2nd phase of our protocol, no changes at ledger L_B would be made. If $\mathcal{O}_A$ were to censor $\mathbb{C}_B$ in the 4th phase of our protocol, L_B would contain an initiated transfer entry, which has not any impact on the consistency of the ledger since it does not contain any locked tokens. □

If $\mathcal{O}_B$ were to censor $\mathbb{C}_A$ in the 3rd phase of our protocol, A would contain some frozen funds of the initiated transfer. However, these funds can be recovered back to $\mathbb{C}_A$ upon a recovery call of $\mathbb{E}_A$ after a recovery timeout has passed. Note that after tokens of $\mathbb{C}_A$ have been recovered and synced to $\mathbb{IPSC}_A$ in PB, it is not possible to finish the 4th stage of our protocol since it requires providing the proof that tokens were deducted at A and such a proof cannot be constructed anymore. The same holds in the situation where the sync to $\mathbb{IPSC}_A$ at PB has not been made yet – after recovery of tokens, $\mathbb{E}_A$ does not allow to deduct the same tokens due to its correct execution (see Theorem 6). □

Theorem 16. *(Identity Management of CBDC Instances I) A new (potentially fake) CBDC instance cannot enter the ecosystem of wholesale CBDC upon its decision.*

Justification. To extend the list of valid CBDC instances (stored in IMSC contract), the majority vote of all existing CBDC instances must be achieved through public voting on IMSC. □

Theorem 17. *(Identity Management of CBDC Instances II) Any CBDC instance (that e.g., does not respect certain rules for issuance of tokens) might be removed from the ecosystem of CBDC by majority vote.*

Justification. A publicly visible voting about removal of a CBDC instance from the ecosystem is realized by IMSC contract that resides in PB, while each existing instance has a single vote. □

7.3 Contributing Papers

The papers that contributed to this research direction are enumerated in the following, while highlighted papers are attached to this thesis in their original form. Note that these papers were not yet published nor accepted as of writing.

[HS20] **Ivan Homoliak and Pawel Szalachowski. Aquareum: A centralized ledger enhanced with blockchain and trusted computing. *arXiv preprint arXiv:2005.13339*, 2020.**

[HPH+23] **Ivan Homoliak, Martin Perešíni, Patrik Holop, Jakub Handzuš, and Fran Casino. CBDC-Aquasphere: Interoperable central bank digital currency built on trusted computing and blockchain. *arXiv preprint arXiv:2305.16893*, 2023.**

Chapter 8

Conclusion

In this work, we presented the summary of our research and its contributions to the standardization of vulnerability/threat analysis and modeling in blockchains as well as particular areas in blockchains' consensus protocols, cryptocurrency wallets, electronic voting, and secure logging with the focus on security and/or privacy aspects.

In detail, first, we introduced the security reference architecture for blockchains that adopts a stacked model, describing the nature and hierarchy of various security and privacy aspects. Then, we proposed a blockchain-specific version of the threat-risk assessment standard ISO/IEC 15408 by embedding the stacked model into this standard. Next, we investigated a few attacks on Proof-of-Work consensus protocols such as selfish mining attacks, greedy transaction selection attacks, and undercutting attacks – for all of them we proposed mitigation techniques. Then, we dealt with cryptocurrency wallets, where we described our proposed classification of authentication schemes and proposed Smart-OTPs, two-factor authentication for smart contract wallets based on One-Time Passwords. Next, we focused on electronic voting using blockchains as an instance of a public bulletin board, and we described our proposals BBB-Voting and SBvote as well as the Always-on-Voting framework for repetitive voting. Finally, we dealt with secure logging, where we presented Aquareum, a centralized ledger based on blockchain and trusted computing. At a follow-up stage in secure logging direction, we built on top of Aquareum and proposed CBDC-AquaSphere, an interoperability protocol for central bank digital currencies.

In our future work, we plan to focus on several directions, such as secure and efficient Proof-of-Stake protocols based on Direct Acyclic Graphs, simulations of selfish mining and similar incentive attacks (with various numbers of attackers) on consensus protocols designed to mitigate selfish mining, optimized on-chain verification in e-voting protocols (including relatively new concepts of partial-tally-hiding), interoperable execution of smart contracts across CBDC instances, optimization of Merkle-Patricia tries to enable secure and consistent parallel processing and thus improvement in processing throughput.

Bibliography

[In18] Infinity Blockchain Labs Europe. Infinito wallet, 2018.

[Adi08] Ben Adida. Helios: Web-based open-audit voting. In *USENIX Security*, pages 335–348. USENIX Assoc., 2008.

[ADMG$^+$16] Matthew Amy, Olivia Di Matteo, Vlad Gheorghiu, Michele Mosca, Alex Parent, and John Schanck. Estimating the cost of generic quantum preimage attacks on sha-2 and sha-3. In *International Conference on Selected Areas in Cryptography*, pages 317–337. Springer, 2016.

[AGJS13] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative technology for cpu based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, volume 13. ACM New York, NY, USA, 2013.

[AGKK19] Myrto Arapinis, Andriana Gkaniatsou, Dimitris Karakostas, and Aggelos Kiayias. A formal treatment of hardware wallets. In *Financial Cryptography*. Springer, 2019.

[AGLS18] Emmanuelle Anceaume, Antoine Guellier, Romaric Ludinard, and Bruno Sericola. Sycomore: A Permissionless Distributed Ledger that Self-Adapts to Transactions Demand. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–8. IEEE, 2018.

[AMQ13] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. RBFT: Redundant byzantine fault tolerance. In *ICDCS*. IEEE, 2013.

[Ant16] Anton Churyumov. Byteball: A Decentralized System for Storage and Transfer of Value. `https://byteball.org/Byteball.pdf`, 2016.

[AOK$^+$19] Yusuke Aoki, Kai Otsuki, Takeshi Kaneko, Ryohei Banno, and Kazuyuki Shudo. SimBlock: A Blockchain Network Simulator. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 325–329. IEEE, 2019.

[Arm16] Armory Technologies, Inc. Bitcoin Armory, 2016.

[AvM19] Maher Alharby and Aad van Moorsel. Blocksim: A simulation framework for blockchain systems. *SIGMETRICS Perform. Eval. Rev.*, 46(3):135–138, January 2019.

[AZEH09] Fadi Aloul, Syed Zahidi, and Wassim El-Hajj. Two factor authentication using mobile phones. In *Computer Systems and Applications, 2009. IEEE/ACS International Conference on*, pages 641–644. IEEE, 2009.

[B+79] George Robert Blakley et al. Safeguarding cryptographic keys. In *Proceedings of the national computer conference*, volume 48, pages 313–317, 1979.

[Bar18] James Barry. Blockchain Technology Needs Standardization, 2018.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. Cryptology ePrint Archive, Report 2018/601, 2018. `https://eprint.iacr.org/2018/601`.

[BCD+17] Ferdinand Brasser, Srdjan Capkun, Alexandra Dmitrienko, Tommaso Frassetto, Kari Kostiainen, Urs Müller, and Ahmad-Reza Sadeghi. Dr. sgx: hardening sgx enclaves against cache attacks with data location randomization. *arXiv preprint arXiv:1709.09917*, 2017.

[BCD+18] Andrea Biondo, Mauro Conti, Lucas Davi, Tommaso Frassetto, and Ahmad-Reza Sadeghi. The guard's dilemma: Efficient code-reuse attacks against intel {SGX}. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1213–1227, 2018.

[BCG15] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On Bitcoin as a public randomness source. *IACR Cryptol. ePrint Arch.*, 2015:1015, 2015.

[BCGH16] Richard Gendal Brown, James Carlyle, Ian Grigg, and Mike Hearn. Corda: an introduction. *R3 CEV, August*, 1(15):14, 2016.

[BDL+12] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012.

[Ber09] Daniel J Bernstein. Introduction to post-quantum cryptography. In *Post-quantum cryptography*, pages 1–14. Springer, 2009.

[BFP+01] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing*, PODC '01, pages 274–283, New York, NY, USA, 2001. ACM.

[BGG$^+$19] Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé, and Paul Zimmermann. Discrete Logarithms in GF(p) – 795 bits. [Online], 2019.

[Bin19] Binance. Binance Security Breach Update, 2019.

[Bin20] Binance.com. Binance, 2020.

[Bit18a] Bitcoin Project. Bitcoin Core, 2018.

[Bit18b] Bitcoin Wallet developers. Bitcoin Wallet, 2018.

[Bit18c] Bitcoin Wiki. Atomic Swap, 2018.

[Bit18d] BitLox. BitLox wallet, 2018.

[Bit19] Bitmex_Reserach. The mystery of the bitcoin nonce pattern. https://blog.bitmex.com/the-mystery-of-the-bitcoin-nonce-pattern/, 2019.

[BKM18] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on bft consensus, 2018.

[BKS$^+$22] Pietro Borrello, Andreas Kogler, Martin Schwarzl, Moritz Lipp, Daniel Gruss, and Michael Schwarz. ÆPIC Leak: Architecturally leaking uninitialized data from the microarchitecture. In *31st USENIX Security Symposium (USENIX Security 22)*, 2022.

[BKT$^+$19] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In *ACM SIGSAC CCS '19*, pages 585–602. ACM New York, NY, USA, 2019.

[Blo18] Blockchain Luxembourg S.A. Blockchain Wallet, 2018.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *ASIACRYPT*. Springer, 2001.

[BMC$^+$15] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A Kroll, and Edward W Felten. Sok: Research perspectives and challenges for Bitcoin and cryptocurrencies. In *IEEE S&P*. IEEE, 2015.

[BRA$^+$18] Thanh Bui, Siddharth Prakash Rao, Markku Antikainen, Viswanathan Manihatty Bojan, and Tuomas Aura. Man-in-the-machine: exploiting ill-secured communication inside the computer. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1511–1525. USENIX Association, 2018.

[Bre00] Eric A Brewer. Towards robust distributed systems. In *PODC*, volume 7, pages 343477–343502. Portland, OR, 2000.

[BRR$^+$15] Josh Benaloh, Ronald Rivest, Peter YA Ryan, Philip Stark, Vanessa Teague, and Poorvi Vora. End-to-end verifiability. *arXiv preprint arXiv:1504.03778*, 2015.

[BSCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *USENIX Security*, 2014.

[BSH23] Rastislav Budinský, Ivana Stančíková, and Ivan Homoliak. Fee-Redistribution Smart Contracts for Transaction-Fee-Based Regime of Blockchains with the Longest Chain Rule. In *IEEE International Conference on Blockchain, Blockchain 2023, Hainan, China, December 17-21, 2023*. IEEE, 2023.

[BT94] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, STOC '94, pages 544–553, New York, NY, USA, 1994. ACM.

[BTC18] BTC.com. BTC wallet: Powerful Bitcoin and Bitcoin Cash wallet, 2018.

[Bun16] Jean-Pierre Buntinx. Brain wallets are not secure and 'no one should use them,' says study, 2016.

[BWG$^+$19] Dan Boneh, Riad S. Wahby, Sergey Gorbunov, Hoeteck Wee, and Zhenfei Zhang. RFC Internet-Draft: BLS signature. 2019.

[Car18] CarbonWallet.com. Multi Signature Online Cryptocurrency Wallet, 2018.

[CCF$^+$20] Manuel MT Chakravarty, Sandro Coretti, Matthias Fitzi, Peter Gazi, Philipp Kant, Aggelos Kiayias, and Alexander Russell. Hydra: Fast isomorphic state channels. *Cryptology ePrint Archive*, 2020.

[CDGM19] Melissa Chase, Apoorvaa Deshpande, Esha Ghosh, and Harjasleen Malvai. Seemless: Secure end-to-end encrypted messaging with less trust. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1639–1656, 2019.

[CDP18] Fran Casino, Thomas K Dasaklis, and Constantinos Patsakis. A systematic literature review of blockchain-based applications: current status, classification and open issues. *Telematics and Informatics*, 2018.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 103–118. Springer, 1997.

[CHH$^+$18] Vincent Chia, Pieter Hartel, Qingze Hum, Sebastian Ma, Georgios Piliouras, Daniel Reijsbergen, Mark van Staalduinen, and Pawel Szalachowski. Rethinking blockchain security: Position paper. In *Blockchain*, 2018.

[Cir18] Circle Internet Financial Limited. Circle Pay, 2018.

[Cit18] Citowise Developments. Citowise wallet, 2018.

[CKWN16a] Miles Carlsten, Harry A. Kalodner, S. Matthew Weinberg, and Arvind Narayanan. On the instability of Bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[CKWN16b] Miles Carlsten, Harry A. Kalodner, S. Matthew Weinberg, and Arvind Narayanan. On the instability of Bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[CL$^+$99] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, 1999.

[CLD16] Victor Costan, Ilia Lebedev, and Srinivas Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 857–874, 2016.

[Coi18a] Coinbase. Coinbase wallet, 2018.

[Coi18b] Coinomi Ltd. Coinomi Wallet, 2018.

[Com17] Common Criteria. Common criteria for information technologysecurity evaluation. part 1: Introduction and general model, 2017.

[Con19a] ConsenSys. Gnosis Wallet, 2019.

[Con19b] ConsenSys. Mythril, 2019.

[Coo18] CoolBitX. The CoolWallet S, 2018.

[Cop19] Copay. The Secure, Shared Bitcoin Wallet, 2019.

[CP02] C. Cachin and J. A. Poritz. Secure intrusion-tolerant replication on the internet. In *Proceedings International Conference on Dependable Systems and Networks*, pages 167–176. IEEE, 2002.

[CRD20] Tobias Cloosters, Michael Rodler, and Lucas Davi. Teerex: Discovery and exploitation of memory corruption vulnerabilities in {SGX} enclaves. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pages 841–858, 2020.

[cry18] crytic. Slither, the Solidity source analyzer, 2018.

[CSC16] Nicolas Courtois, Guangyan Song, and Ryan Castellucci. Speed optimizations in Bitcoin key recovery attacks. *Tatra Mountains Mathematical Publications*, 67(1), 2016.

[CW09] Scott A Crosby and Dan S Wallach. Efficient data structures for tamper-evident logging. In *USENIX Security Symposium*, pages 317–334, 2009.

[CXS$^+$17] Lin Chen, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi. On security analysis of proof-of-elapsed-time (poet). In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*. Springer, 2017.

[CZK$^+$19a] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 185–200. IEEE, 2019.

[CZK$^+$19b] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *EuroS&P*, pages 185–200. IEEE, 2019.

[CZRZ17] Sanchuan Chen, Xiaokuan Zhang, Michael K Reiter, and Yinqian Zhang. Detecting privileged side-channel attacks in shielded execution with déjá vu. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 7–18, 2017.

[Dae18] Daedalus Team. Daedalus Wallet, 2018.

[Del15] Dell SecureWorks. Cryptocurrency-stealing malware landscape, 2015.

[DF18] Bharatee Bhusana Dash and J. Stephen Ferris. Economic Performance and Electoral Volatility: Testing the Economic Voting Hypothesis on Indian States, 1957–2013. Carleton Economic Papers 18-07, Carleton University, Department of Economics, June 2018.

[DFKO18] Vivek Dhakal, Anna Maria Feit, Per Ola Kristensson, and Antti Oulasvirta. Observations on typing from 136 million keystrokes. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 646–658. ACM, 2018.

[DGK$^+$20] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 566–583, 2020.

[DHV$^+$23] Jozef Drga, Ivan Homoliak, Juraj Vanco, Athanasios Vasilakos, Martin Perešíni, and Petr Hanacek. Detecting and preventing credential misuse in otp-based two and half factor authentication toward centralized services utilizing blockchain-based identity management. In *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–4. IEEE, 2023.

[Dig22] Digital Euro Association. The CBDC Manifesto, 2022.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2), 1988.

[DMMM18] Gaby Dagher, Praneeth Babu Marella, Matea Milojkovic, and Jordan Mohler. BroncoVote: Secure Voting System using Ethereum's Blockchain. In *ICISSP'18*, pages 96–107. SCITEPRESS, 2018.

[DMPZ14] Sisi Duan, Hein Meling, Sean Peisert, and Haibin Zhang. Bchain: Byzantine replication with high throughput and embedded reconfiguration. In *OPODIS*. Springer, 2014.

[Don19] Donjon Team. Extracting seed from Ellipal wallet, 2019.

[DPS19] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *International Conference on Financial Cryptography and Data Security*, pages 23–41. Springer, 2019.

[Dra18] Justin Drake. Minimal vdf randomness beacon. https://ethresear.ch/t/minimal-vdf-randomness-beacon/3566, 2018.

[DSS05] Chris Dods, Nigel P Smart, and Martijn Stam. Hash based digital signature schemes. In *IMA International Conference on Cryptography and Coding*, pages 96–115. Springer, 2005.

[ECBS18] Shayan Eskandari, Jeremy Clark, David Barrera, and Elizabeth Stobert. A first look at the usability of Bitcoin key management, 2018.

[EKR17] Thomas Espel, Laurent Katz, and Guillaume Robin. Proposal for protocol on a quorum blockchain with zero knowledge., 2017.

[Ele18] Electrum Technologies GmbH. Electrum Bitcoin wallet, 2018.

[ELL19] ELLIPAL. ELLIPAL Hardware Wallet 2.0, 2019.

[Enc19] Keystone Enclave. Keystone: An open framework for architecting trusted execution environments. `https://keystone-enclave.github.io/`, 2019.

[ent17] entethalliance.org. Enterprise Ethereum Alliance. [Online], 2017.

[Ent19] Enterprise Ethereum Alliance. Enterprise Ethereum Alliance, 2019.

[ES14] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*. Springer, 2014.

[ES18] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM*, 61(7), 2018.

[Eth19] Ethereum. Aleth — Ethereum C++ client, tools and libraries, 2019.

[Eth22] Ethereum Foundation. PLASMA CHAINS. [Online], 2022.

[Eya15] Ittay Eyal. The miner's dilemma. In *2015 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2015.

[FDP$^+$14] Sascha Fahl, Sergej Dechand, Henning Perl, Felix Fischer, Jaromir Smrcek, and Matthew Smith. Hey, nsa: Stay away from my market! future proofing app markets against powerful attackers. In *proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1143–1155, 2014.

[FHZ$^+$19] Qi Feng, Debiao He, Sherali Zeadally, Muhammad Khurram Khan, and Neeraj Kumar. A survey on privacy protection in blockchain system. *Journal of Network and Computer Applications*, 126, 2019.

[Gam85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[Gav15] Gavin Andresen. Bitcoin Miningsim. `https://github.com/gavinandresen/bitcoin_miningsim`, 2015.

[GG16] Vasilis Glykantzis and Arthur Gervais. Bitcoin-Simulator, capable of simulating any re-parametrization of Bitcoin. online, april 2016.

[GGK$^+$15] Steven Goldfeder, Rosario Gennaro, Harry Kalodner, Joseph Bonneau, Joshua A Kroll, Edward W Felten, and Arvind Narayanan. Securing Bitcoin wallets via a new DSA/ECDSA threshold signature scheme, 2015.

[GHM$^+$17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP*. ACM, 2017.

[GJKR07] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, 2007.

[GLS$^+$17] Daniel Gruss, Julian Lettner, Felix Schuster, Olya Ohrimenko, Istvan Haller, and Manuel Costa. Strong and efficient cache side-channel protection using hardware transactional memory. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 217–233, 2017.

[GMSK22] Tiantian Gong, Mohsen Minaei, Wenhai Sun, and Aniket Kate. Towards overcoming the undercutting problem. In *Financial Cryptography '22*, pages 444–463. Springer, 2022.

[Goa19] Laurence Goasduff. Gartner Says Blockchain Deployments Across Financial Services Ecosystems Are At Least Three Years Away, 2019.

[Gro96] Lov K Grover. A fast quantum mechanical algorithm for database search. In *STOC*, pages 212–219. ACM, 1996.

[Gro04] Jens Groth. Efficient maximal privacy in boardroom voting and anonymous broadcast. In Ari Juels, editor, *Financial Cryptography, 8th International Conference, FC 2004, Key West, FL, USA, February 9-12, 2004. Revised Papers*, volume 3110 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2004.

[Gua19] GuardStrike. ContractGuard, 2019.

[HBH$^+$20a] Ivan Homoliak, Dominik Breitenbacher, Ondrej Hujnak, Pieter Hartel, Alexander Binder, and Pawel Szalachowski. SmartOTPs: An air-gapped 2-factor authentication for smart-contract wallets. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies, AFT 2020, New York, NY, USA, October 21-23, 2020*. ACM, 2020.

[HBH$^+$20b] Ivan Homoliak, Dominik Breitenbacher, Ondrej Hujnak, Pieter Hartel, Alexander Binder, and Pawel Szalachowski. SmartOTPs: An Air-Gapped 2-Factor Authentication for Smart-Contract Wallets (Extended Version). *arXiv preprint arXiv:1812.03598*, 2020.

[Hel80] Martin Hellman. A cryptanalytic time-memory trade-off. *IEEE transactions on Information Theory*, 26(4):401–406, 1980.

[HHMH19] Lukáš Hellebrandt, Ivan Homoliak, Kamil Malinka, and Petr Hanáček. Increasing trust in Tor node list using blockchain. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 29–32. IEEE, 2019.

[HHR19] Pieter Hartel, Ivan Homoliak, and Daniël Reijsbergen. An empirical study into the success of listed smart contracts in ethereum. *IEEE Access*, 7:177539–177555, 2019.

[HJP05] Yih-Chun Hu, Markus Jakobsson, and Adrian Perrig. Efficient constructions for one-way hash chains. In *International Conference on Applied Cryptography and Network Security*, pages 423–441. Springer, 2005.

[HL14] Andrew Healy and Gabriel S. Lenz. Substituting the end for the whole: Why voters respond primarily to the election-year economy. *American Journal of Political Science*, 58(1):31–47, 2014.

[HLP$^+$13] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. Using innovative instructions to create trustworthy software solutions. *HASP@ ISCA*, 11, 2013.

[HLS23] Ivan Homoliak, Zengpeng Li, and Pawel Szalachowski. BBB-Voting: 1-out-of-k blockchain-based boardroom voting. In *IEEE International Conference on Blockchain, Blockchain 2023, Hainan, China, December 17-21, 2023*. IEEE, 2023.

[HMV05] Darrel Hankerson, Alfred J Menezes, and Scott Vanstone. Guide to elliptic curve cryptography. *Computing Reviews*, 46(1):13, 2005.

[HN01] Johan Håstad and Mats Näslund. Practical construction and analysis of pseudo-randomness primitives. In *ASIACRYPT*, pages 442–459. Springer, 2001.

[Hoe22] Jochen Hoenicke. Johoe's Bitcoin Mempool Statistic. `https://jochen-hoenicke.de/queue/`, 2022.

[HPH$^+$23] Ivan Homoliak, Martin Perešíni, Patrik Holop, Jakub Handzuš, and Fran Casino. Cbdc-aquasphere: Interoperable central bank digital currency built on trusted computing and blockchain. *arXiv preprint arXiv:2305.16893*, 2023.

[HRZ10] Feng Hao, Peter Y. A. Ryan, and Piotr Zielinski. Anonymous voting by two-round public discussion. *IET Information Security*, 4(2):62–67, 2010.

[HS00] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *Proceedings of the 19th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'00, pages 539–556, Berlin, Heidelberg, 2000. Springer-Verlag.

[HS20] Ivan Homoliak and Pawel Szalachowski. Aquareum: A centralized ledger enhanced with blockchain and trusted computing. *arXiv preprint arXiv:2005.13339*, 2020.

[HTT$^+$20] Qingze Hum, Wei Jin Tan, Shi Ying Tey, Latasha Lenus, Ivan Homoliak, Yun Lin, and Jun Sun. Coinwatch: A clone-based approach for detecting vulnerabilities in cryptocurrencies. In *2020 IEEE International Conference on Blockchain (Blockchain)*, pages 17–25. IEEE, 2020.

[HVHS19] Ivan Homoliak, Sarad Venugopalan, Qingze Hum, and Pawel Szalachowski. A security reference architecture for blockchains. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 390–397. IEEE, 2019.

[HVR$^+$20] Ivan Homoliak, Sarad Venugopalan, Daniël Reijsbergen, Qingze Hum, Richard Schumi, and Pawel Szalachowski. The security reference architecture for blockchains: Toward a standardized model for studying vulnerabilities, threats, and defenses. *IEEE Communications Surveys & Tutorials*, 23(1):341–390, 2020.

[Hyp17] Hyperledger team. Hyperledger architecture, volume 1: Consensus, 2017.

[Hyp22] Hyperledger Foundation. Hyperledger. `https://github.com/hyperledger`, 2022.

[Int18] Intel. Resources and Response to Side Channel L1 Terminal Fault, 2018.

[ISO19a] ISO. ISO/CD 23257: Blockchain and distributed ledger technologies – Reference architecture vulnerabilities, 2019.

[ISO19b] ISO. ISO/DTR 23245: Blockchain and distributed ledger technologies – Security risks, threats and vulnerabilities, 2019.

[JMP13] Hugo Jonker, Sjouke Mauw, and Jun Pang. Privacy and verifiability in voting systems: Methods, developments and trends. *Comput. Sci. Rev.*, 10:1–30, 2013.

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.

[JX22] Si Yuan Jin and Yong Xia. Cev framework: A central bank digital currency evaluation and verification framework with a focus on consensus algorithms and operating architectures. *IEEE Access*, 10:63698–63714, 2022.

[KAC12] Ghassan O Karame, Elli Androulaki, and Srdjan Capkun. Double-spending fast payments in Bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012.

[KAD$^+$20] Mr John Kiff, Jihad Alwazir, Sonja Davidovic, Aquiles Farias, Mr Ashraf Khan, Mr Tanai Khiaonarong, Majid Malaika, Mr Hunter K Monroe, Nobu Sugimoto, Hervé Tourpe, et al. A survey of research on retail central bank digital currency. 2020.

[Kal15] Harry Kalodner. Bitcoin mining simulator. online, 2015.

[KBRK19] Kristián Košt'ál, Rastislav Bencel, Michal Ries, and Ivan Kotuliak. Blockchain e-voting done right: Privacy and transparency with public blockchain. In *ICSESS*, pages 592–595. IEEE, 2019.

[Kee18] KeepKey. The Simple Cryptocurrency Hardware Wallet, 2018.

[KJG$^+$18] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE S&P 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, 2018.

[KMB17] Dmitry Kogan, Nathan Manohar, and Dan Boneh. T/key: second-factor authentication from secure hash chains. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 983–999. ACM, 2017.

[KMS$^+$16] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *IEEE S&P*. IEEE, 2016.

[Kob94] Neal Koblitz. *A course in number theory and cryptography, 2nd Edition*, volume 114 of *Graduate texts in mathematics*. Springer, 1994.

[KR18] Aggelos Kiayias and Alexander Russell. Ouroboros-BFT: A Simple Byzantine Fault Tolerant Consensus Protocol, 2018.

[Kra19] Kraken. Inside Kraken Security Labs: Flaw Found in Keepkey Crypto Hardware Wallet, 2019.

[Kra20] Kraken. Kraken Identifies Critical Flaw in Trezor Hardware Wallets, 2020.

[KRDO17] Aggelos Kiayias, Al'exander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO'17*. Springer, 2017.

[KRS$^+$20] C. Killer, B. Rodrigues, E. J. Scheid, M. Franco, M. Eck, N. Zaugg, A. Scheitlin, and B. Stiller. Provotum: A blockchain-based and end-to-end verifiable remote electronic voting system. In *IEEE LCN*, pages 172–183, 2020.

[KSRH12] Dalia Khader, Ben Smyth, Peter Y. A. Ryan, and Feng Hao. A fair and robust voting system by broadcast. In *5th International Conference on Electronic Voting 2012, (EVOTE 2012), Co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC, July 11-14, 2012, Castle Hofen, Bregenz, Austria*, pages 285–299, 2012.

[KY02] Aggelos Kiayias and Moti Yung. Self-tallying elections and perfect ballot secrecy. In David Naccache and Pascal Paillier, editors, *Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings*, volume 2274 of *Lecture Notes in Computer Science*, pages 141–158. Springer, 2002.

[L$^+$98] Leslie Lamport et al. The part-time parliament. *ACM Transactions on Computer systems*, 16(2), 1998.

[Lam79] Leslie Lamport. Constructing digital signatures from a one-way function. Technical report, Technical Report CSL-98, SRI International Palo Alto, 1979.

[Lam81] Leslie Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.

[Led18a] Ledger. Ledger Nano, 2018.

[Led18b] Ledger. Ledger Nano S, 2018.

[LeM18] Colin LeMahieu. Nano : A feeless distributed cryptocurrency network. `https://www.exodus.com/assets/docs/nano-whitepaper.pdf`, 2018.

[LKO$^+$21] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021.

[LLK13] Ben Laurie, Adam Langley, and Emilia Kasper. Rfc 6962 – certificate transparency. RFC 6962, RFC Editor, June 2013.

[LLW$^+$19] Ziyao Liu, Nguyen Cong Luong, Wenbo Wang, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. A Survey on Blockchain: A Game Theoretical Perspective. `doi.org/10.1109/ACCESS.2019.2909924`, 2019.

[LM95] Frank T Leighton and Silvio Micali. Large provably fast and secure digital signature schemes based on secure hash functions, 1995. US Patent 5,432,852.

[LNE$^+$19] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Emin Gün Sirer, and Peter Pietzuch. Teechain: a secure payment network with asynchronous blockchain access. In *ACM SOPS*, pages 63–79, 2019.

[LRP20] Stefanos Leonardos, Daniël Reijsbergen, and Georgios Piliouras. Presto: A systematic framework for blockchain consensus protocols. *IEEE Transactions on Engineering Management*, 2020.

[LSY$^+$20] Y. Li, W. Susilo, G. Yang, Y. Yu, D. Liu, X. Du, and M. Guizani. A blockchain-based self-tallying voting protocol in decentralized iot. *IEEE TDSC*, pages 1–1, 2020.

[LSZ15] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols. In *Financial Crypto*. Springer, 2015.

[LTM$^+$11] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, and Dawn Leaf. NIST cloud computing reference architecture. *NIST special publication*, 500(2011):1–28, 2011.

[Lun18] Luno. Luno wallet, 2018.

[MAB$^+$13] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative instructions and software model for isolated execution. *Hasp@isca*, 10(1), 2013.

[Max11] Gregory Maxwell. Deterministic wallets, 2011.

[MB15] Malte Möser and Rainer Böhme. Trends, tips, tolls: A longitudinal study of bitcoin transaction fees. In *Financial Cryptography Workshops*, 2015.

[MBB$^+$15] Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. {CONIKS}: Bringing key transparency to end users. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 383–398, 2015.

[MBH$^+$05] D M'raihi, M Bellare, F Hoornaert, D Naccache, and O Ranen. HOTP: An HMAC-based one-time password algorithm. Technical report, 2005.

[MC13] Tyler Moore and Nicolas Christin. Beware the middleman: Empirical analysis of Bitcoin-exchange risk. In *Financial Crypto*. Springer, 2013.

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Proceedings of Advances in Cryptology*, 1988.

[Mer89] Ralph C Merkle. A certified digital signature. In *Conference on the Theory and Application of Cryptology*, pages 218–238. Springer, 1989.

[Met19] MetaMask team. MetaMask, 2019.

[Mic03] Silvio Micali. Simple and fast optimistic protocols for fair electronic exchange. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 12–19. ACM, 2003.

[Mic20] Microsoft. Enclave EVM, 2020.

[Mil03] J.D. Miller. *Game Theory at Work: How to Use Game Theory to Outthink and Outmaneuvar Your Competition*. McGraw Hill LLC, 2003.

[MKV23] Robert McLaughlin, Christopher Kruegel, and Giovanni Vigna. A large scale study of the ethereum arbitrage ecosystem. In *USENIX Security '23*, pages 3295–3312, 2023.

[MMC$^+$19] Jelena Mišić, Vojislav B Mišić, Xiaolin Chang, Saeideh Gholamrezazadeh Motlagh, and M Zulfiker Ali. Modeling of Bitcoin's blockchain delivery network. *IEEE TNSE*, 7(3):1368–1381, 2019.

[MMPR11] David M'Raihi, Salah Machani, Mingliang Pei, and Johan Rydell. Totp: Time-based one-time password algorithm. Technical report, 2011.

[MOG$^+$20] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based fault injection attacks against intel sgx. In *Proceedings of the 41st IEEE Symposium on Security and Privacy (S&P'20)*, 2020.

[MR01] Philip MacKenzie and Michael Reiter. Two-party Generation of DSA Signatures. In *Annual International Cryptology Conference*, pages 137–154. Springer, 2001.

[MRSS19] Raphael Matile, Bruno Rodrigues, Eder J. Scheid, and Burkhard Stiller. CaIV: Cast-as-Intended Verifiability in Blockchain-based Voting. In *ICBC 2019, Seoul, Korea (South), May 14-17, 2019*, pages 24–28, Washington, DC, 2019. IEEE.

[MRV99] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.

[MS06] George J. Mailath and Larry Samuelson. *Repeated Games and Reputations: Long-Run Relationships*. Oxford University Press, November 2006.

[MSH17] Patrick McCorry, Siamak F. Shahandashti, and Feng Hao. A smart contract for boardroom voting with maximum voter privacy. In *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers*, pages 357–375, 2017.

[MXC$^+$16] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *ACM CCS*. ACM, 2016.

[Myc18] Mycelium Holding LTD. Mycelium wallet, 2018.

[Myc19] Mycelium Holding LTD. Mycelium Entropy, 2019.

[MyE18] MyEtherWallet, Inc. MyEtherWallet, 2018.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[Nak09] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2009.

[Nat22] Nathan Reiff. What Is Polygon (MATIC)? Definition, Strengths, and Weaknesses. [Online], 2022.

[Nod18] NodeMcu Team. NodeMCU, 2018.

[OL88] Brian M Oki and Barbara H Liskov. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *ACM Symposium on Principles of Distributed Computing*. ACM, 1988.

[OO14] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *USENIX ATC*, 2014.

[Ope20] Open Enclave SDK Contributors. Open Enclave SDK, 2020.

[OR94] Martin J. Osborne and Ariel Rubinstein. *A course in game theory*. The MIT Press, Cambridge, USA, 1994. electronic edition.

[Par17a] Parity Technologies. A Postmortem on the Parity Multi-Sig Library Self-Destruct, 2017.

[Par17b] Parity Technologies. The Multi-sig Hack: A Postmortem, 2017.

[Pax20] Paxful, Inc. Paxful, 2020.

[Pay20] Payward, Inc. Kraken, 2020.

[PBH$^+$23] Martin Perešíni, Federico Matteo Benčić, Martin Hrubý, Kamil Malinka, and Ivan Homoliak. Incentive Attacks on DAG-Based Blockchains with Random Transaction Selection. In *IEEE International Conference on Blockchain, Blockchain 2023, Hainan, China, December 17-21, 2023*. IEEE, 2023.

[PBMH21] Martin Perešíni, Federico Matteo Benčić, Kamil Malinka, and Ivan Homoliak. DAG-Oriented Protocols PHANTOM and GHOSTDAG under Incentive Attack via Transaction Selection Strategy. *arXiv:2109.01102*, 2021.

[Pey17] Antony Peyton. Cyren sounds siren over Bitcoin siphon scam, 2017.

[PGF21] Remigijus Paulavičius, Saulius Grigaitis, and Ernestas Filatovas. A Systematic Review and Empirical Analysis of Blockchain Simulators. `doi.org/10.1109/ACCESS.2021.3063324`, 2021.

[PHB$^+$23] Martin Perešíni, Ivan Homoliak, Federico Matteo Benčić, Martin Hrubý, and Kamil Malinka. Incentive Attacks on DAG-Based Blockchains with Random Transaction Selection. arXiv:2305.16757, 2023.

[PHMH23] Martin Perešíni, Tomáš Hladký, Kamil Malinka, and Ivan Homoliak. DAG-Sword: A Simulator of Large-Scale Network Topologies for DAG-Oriented Proof-of-Work Blockchains. *arXiv:2311.04638*, 2023.

[PHMH24] Martin Perešíni, Tomáš Hladký, Kamil Malinka, and Ivan Homoliak. DAG-Sword: A Simulator of Large-Scale Network Topologies for DAG-Oriented Proof-of-Work Blockchains. In *Hawaii International Conference on System Sciences (HICSS), Hawaii, USA, January 3-6, 2024*. IEEE, 2024.

[Pie19] Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 60:1–60:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[Pol20] Polo Digital Assets, Ltd. Poloniex, 2020.

[Pop16] Serguei Popov. The Tangle. `http://tanglereport.com/wp-content/uploads/2018/01/IOTA_Whitepaper.pdf`, 2016.

[PR18] Somnath Panja and Bimal Kumar Roy. A secure end-to-end verifiable e-voting system using zero knowledge based blockchain. *IACR Cryptology ePrint Archive*, 2018:466, 2018.

[PRVB13] Marek Palatinus, Pavol Rusnak, Aaron Voisine, and Sean Bowe. BIP-39, 2013.

[PS17] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In *PODC*. ACM, 2017.

[PSNR21] Sunoo Park, Michael Specter, Neha Narula, and Ronald L Rivest. Going from bad to worse: from internet voting to blockchain voting. *Journal of Cybersecurity*, 7(1), 2021.

[QRS11] QRStuff.com. What Size Should A Printed QR Code Be?, 2011.

[Rac14] Rachel Abrams and Nathaniel Popper. Trading Site Failure Stirs Ire and Hope for Bitcoin, 2014.

[Ray19] James Ray. Modified merkle patricia trie specification. `https://github.com/ethereum/wiki/wiki/Patricia-Tree`, 2019.

[Reu16] Reuters. Bitcoin Worth $72M was Stolen in Bitfinex Exchange Hack in Hong Kong, 2016.

[Riz16] Peter R Rizun. Subchains: A technique to scale Bitcoin and improve the user experience. *Ledger*, 1, 2016.

[RST01] Ronald L Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2001.

[Sch90] Fred B Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM CSUR*, 22(4), 1990.

[Sch05] Bruce Schneier. Two-factor authentication: too little, too late. *Communications of the ACM*, 48(4):136, 2005.

[SDS$^+$20] Rajani Singh, Ashutosh Dhar Dwivedi, Gautam Srivastava, Agnieszka Wiszniewska-Matyszkiel, and Xiaochun Cheng. A game theoretic analysis of resource mining in blockchain. *Cluster Computing*, 23(3):2035–2046, 2020.

[SEC00] SECG SEC. 2: Recommended elliptic curve domain parameters. *Standards for Efficient Cryptography Group, Certicom Corp*, 2000.

[SF20] D. SAMUEL and D FRANCK. Cpu-z oc world records. highest overclocking of all times. https://valid.x86.fr/records.php, 2020.

[SGY20] Mohamed Seifelnasr, Hisham S. Galal, and Amr M. Youssef. Scalable open-vote network on ethereum. In *Financial Cryptography and Data Security*, volume 12063 of *LNCS*, pages 436–450. Springer, 2020.

[SH23a] Ivana Stančíková and Ivan Homoliak. Sbvote: Scalable self-tallying blockchain-based voting. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, pages 203–211, 2023.

[SH23b] Ivana Stančíková and Ivan Homoliak. SBvote: Scalable Self-Tallying Blockchain-Based Voting. In *ACM/SIGAPP SAC*, 2023.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[SHI18] SHIFT Cryptosecurity. BitBox hardware wallet, 2018.

[SI21] Vijak Sethaput and Supachate Innet. Blockchain application for central bank digital currencies (cbdc). In *2021 Third International Conference on Blockchain Computing and Applications (BCCA)*, pages 3–10. IEEE, 2021.

[SJK$^+$17] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 444–460. Ieee, 2017.

[SLK$^+$17] Jaebaek Seo, Byoungyoung Lee, Seong Min Kim, Ming-Wei Shih, Insik Shin, Dongsu Han, and Taesoo Kim. Sgx-shield: Enabling address space layout randomization for sgx programs. In *NDSS*, 2017.

[SLKP17] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. T-sgx: Eradicating controlled-channel attacks against enclave programs. In *NDSS*, 2017.

[SLZ16] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. Spectre: A fast and scalable cryptocurrency protocol. *IACR Cryptol. ePrint Arch.*, 2016:1159, 2016.

[SM20] Wellington Fernandes Silvano and Roderval Marcelino. IOTA Tangle: A cryptocurrency to communicate Internet-of-Things data. *Future Generation Computer Systems*, 112:307–319, 2020.

[Smi82] John Maynard Smith. *Evolution and the Theory of Games*. Cambridge University Press, 1982.

[Som22] Yonatan Sompolinsky. Kaspa. online, 2022. `https://kaspa.org/`.

[SRHS19a] Pawel Szalachowski, Daniël Reijsbergen, Ivan Homoliak, and Siwei Sun. Strongchain: Transparent and collaborative proof-of-work consensus. In *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019.*, pages 819–836, 2019.

[SRHS19b] Pawel Szalachowski, Daniël Reijsbergen, Ivan Homoliak, and Siwei Sun. Strongchain: Transparent and collaborative proof-of-work consensus. In *USENIX Security*, 2019.

[SSZ16] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in Bitcoin. In *FC*. Springer, 2016.

[SWZ21] Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. PHANTOM GHOSTDAG: A Scalable Generalization of Nakamoto Consensus. In *ACM AFT '21*, pages 57–70, New York, NY, USA, 2021. ACM.

[SZ13] Yonatan Sompolinsky and Aviv Zohar. Accelerating bitcoin's transaction processing. fast money grows on trees, not chains. *IACR Cryptology ePrint Archive*, 2013(881), 2013.

[Tan17] Wei Tang. Ecip-1029: Include uncles in total difficulty calculation, 2017.

[TE18a] Itay Tsabary and Ittay Eyal. The gap game. In *ACM SIGSAC CCS '18*, pages 713–728, 2018.

[TE18b] Itay Tsabary and Ittay Eyal. The gap game. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018.

[Tea18] Team Rocket. Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies, 2018.

[Tec18] Parity Technologies. Parity Wallet, 2018.

[TH18] Dominic Williams Timo Hanke, Mahnush Movahedi. DFINITY technology overview series, consensus system. *CoRR*, abs/1805.04548, 2018.

[Tre18] Trezor. Trezor, 2018.

[Tru19] TrustedCoin, LLC. TrustedCoin cosigning service, 2019.

[Ull17] Peter Ullrich. The risk to breach vote privacy by unanimous voting. *Journal of Information Security and Applications*, 35:168–174, 2017.

[Unc18] Unchained Capital. TrezorMultisig2of3: Ethereum Multisignature smart contract, 2018.

[VBMW$^+$18] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 991–1008, 2018.

[VHLS20] Sarad Venugopalan, Ivan Homoliak, Zengpeng Li, and Pawel Szalachowski. Bbb-voting: 1-out-of-k blockchain-based boardroom voting, 2020.

[VHLS21] Sarad Venugopalan, Ivan Homoliak, Zengpeng Li, and Pawel Szalachowski. BBB-Voting: 1-out-of-k Blockchain-Based Boardroom Voting, 2021.

[VM15] Marie Vasek and Tyler Moore. There's no free lunch, even using bitcoin: Tracking the popularity and profits of virtual currency scams. In *Financial Cryptography*, pages 44–61. Springer, 2015.

[VSH23] Sarad Venugopalan, Ivana Stančíková, and Ivan Homoliak. Always on voting: A framework for repetitive voting on the blockchain. *IEEE Transactions on Emerging Topics in Computing*, 2023.

[W3C19a] W3C community. Decentralized Identifiers (DIDs), 2019.

[W3C19b] W3C Working Group. Verifiable credentials use cases. [Online], 2019.

[WBW$^+$21] Taotao Wang, Xiaoqian Bai, Hao Wang, Soung Chang Liew, and Shengli Zhang. Game-Theoretical Analysis of Mining Strategy for Bitcoin-NG Blockchain Protocol. *IEEE Systems Journal*, 15(2):2708–2719, 2021.

[Wes20] Benjamin Wesolowski. Efficient verifiable delay functions. *J. Cryptol.*, 33(4):2113–2147, 2020.

[WG18] Karl Wüst and Arthur Gervais. Do you need a blockchain? In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 45–54. IEEE, 2018.

[WHH$^+$19] Wenbo Wang, Dinh Thai Hoang, Peizhao Hu, Zehui Xiong, Dusit Niyato, Ping Wang, Yonggang Wen, and Dong In Kim. A survey on consensus mechanisms and mining strategy management in blockchain networks. *IEEE Access*, 7:22328–22370, 2019.

[Wik22] Bitcoin Wiki. Miner Fees, 2022.

[Wit19] Witnet Team. elliptic-curve-solidity. [Online], 2019.

[Wle15] Christopher Wlezien. The myopic voter? the economy and us presidential elections. *Electoral Studies*, 39:195–204, 2015.

[Wol18] Wolfie Zhao. Bithumb $31 Million Crypto Exchange Hack: What We Know (And Don't), 2018.

[Woo14a] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151, 2014.

[Woo14b] Gavin Wood. Ethereum: A secure decentralized generalized transaction ledger. [Online], 2014.

[XX13] Zhifeng Xiao and Yang Xiao. Security and privacy in cloud computing. *IEEE Communications Surveys & Tutorials*, 15(2), 2013.

[YLS$^+$18] Bin Yu, Joseph K Liu, Amin Sakzad, Surya Nepal, Ron Steinfeld, Paul Rimba, and Man Ho Au. Platform-independent secure blockchain-based voting system. In *International Conference on Information Security*, pages 369–386. Springer, 2018.

[ZCC$^+$13] Filip Zagórski, Richard T Carback, David Chaum, Jeremy Clark, Aleksander Essex, and Poorvi L Vora. Remotegrity: Design and use of an end-to-end verifiable remote voting system. In *ACNS*, pages 441–457. Springer, 2013.

[Zer17] Zerocoin Team. Big number library for solidity. [Online], 2017.

[ZH21] Tao Zhang and Zhigang Huang. Blockchain and central bank digital currency. *ICT Express*, 2021.

[ZIL17] ZILLIQA Team. The ZILLIQA Technical Whitepaper, 2017.

[Zim80] Hubert Zimmermann. OSI reference model-the ISO model of architecture for open systems interconnection. *IEEE Transactions on communications*, 28(4), 1980.

[ZMR18] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *ACM CCS*, 2018.

[ZSS$^+$18] Alexei Zamyatin, Nicholas Stifter, Philipp Schindler, Edgar Weippl, and William J Knottenbelt. Flux: Revisiting Near Blocks for Proof-of-Work Blockchains, 2018. `https://eprint.iacr.org/2018/415/20180529:172206`.

[ZXD$^+$18] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*, 14(4):352–375, 2018.

Part II

SELECTED PAPERS

Acknowledgment of the publishers. Due to copyright reasons, we list below the bibliographic information of all attached papers, acknowledging the original source:

[HVR+20] Ivan Homoliak, Sarad Venugopalan, Daniël Reijsbergen, Qingze Hum, Richard Schumi, and Pawel Szalachowski. The security reference architecture for blockchains: Toward a standardized model for studying vulnerabilities, threats, and defenses. *IEEE Communications Surveys & Tutorials*, 23(1):341–390, 2020.

[SRHS19] Pawel Szalachowski, Daniël Reijsbergen, Ivan Homoliak, and Siwei Sun. Strongchain: Transparent and collaborative proof-of-work consensus. In *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019.*, pages 819–836, 2019.

[PBH+] Martin Perešíni, Federico Matteo Benčić, Martin Hrubý, Kamil Malinka, and Ivan Homoliak. Incentive attacks on DAG-based blockchains with random transaction selection. In *IEEE International Conference on Blockchain, Blockchain 2023, Hainan, China, December 17-21, 2023*. IEEE.

[BHS23] Rastislav Budinský, and Ivana Stančíková, and Ivan Homoliak. Fee-redistribution smart contracts for transaction-fee-based regime of blockchains with the longest chain rule. In *IEEE International Conference on Blockchain, Blockchain 2023, Hainan, China, December 17-21, 2023*. IEEE, 2023.

[HLS23] Ivan Homoliak, Zengpeng Li, and Pawel Szalachowski. BBB-Voting: 1-out-of-k blockchain-based boardroom voting. In *IEEE International Conference on Blockchain, Blockchain 2023, Hainan, China, December 17-21, 2023*. IEEE, 2023.

[SH23] Ivana Stančíková and Ivan Homoliak. SBvote: Scalable self-tallying blockchain-based voting. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, pages 203–211, 2023.

[VSH23] Sarad Venugopalan, Ivana Stančíková, and Ivan Homoliak. Always on Voting: A framework for repetitive voting on the blockchain. *IEEE Transactions on Emerging Topics in Computing*, 2023.

[HBH+20a] Ivan Homoliak, Dominik Breitenbacher, Ondrej Hujnak, Pieter Hartel, Alexander Binder, and Pawel Szalachowski. SmartOTPs: An air-gapped 2-factor authentication for smart-contract wallets. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies, AFT 2020, New York, NY, USA, October 21-23, 2020*. ACM, 2020.

[HS20] Ivan Homoliak and Pawel Szalachowski. Aquareum: A centralized ledger enhanced with blockchain and trusted computing. *arXiv preprint arXiv:2005.13339*, 2020.

[HPH+23] Ivan Homoliak, Martin Perešíni, Patrik Holop, Jakub Handzuš, and Fran Casino. CBDC-Aquasphere: Interoperable central bank digital currency built on trusted computing and blockchain. *arXiv preprint arXiv:2305.16893*, 2023.

The Security Reference Architecture for Blockchains: Toward a Standardized Model for Studying Vulnerabilities, Threats, and Defenses

Ivan Homoliak, *Member, IEEE*, Sarad Venugopalan, Daniël Reijsbergen, Qingze Hum, Richard Schumi, and Pawel Szalachowski

***Abstract*—Blockchains are distributed systems, in which security is a critical factor for their success. However, despite their increasing popularity and adoption, there is a lack of standardized models that study blockchain-related security threats. To fill this gap, the main focus of our work is to systematize and extend the knowledge about the security and privacy aspects of blockchains and contribute to the standardization of this domain.We propose the security reference architecture (SRA) for blockchains, which adopts a stacked model (similar to the ISO/OSI) describing the nature and hierarchy of various security and privacy aspects. The SRA contains four layers: (1) the network layer, (2) the consensus layer, (3) the replicated state machine layer, and (4) the application layer. At each of these layers, we identify known security threats, their origin, and countermeasures, while we also analyze several cross-layer dependencies. Next, to enable better reasoning about security aspects of blockchains by the practitioners, we propose a blockchain-specific version of the threat-risk assessment standard ISO/IEC 15408 by embedding the stacked model into this standard. Finally, we provide designers of blockchain platforms and applications with a design methodology following the model of SRA and its hierarchy.**

***Index Terms*—Reference architecture, blockchains, distributed ledgers, security, privacy, vulnerabilities, threats, ISO/IEC 15408.**

I. Introduction

THE POPULARITY of blockchain systems has rapidly increased in recent years, mainly due to the decentralization of control that they aim to provide. Blockchains are full-stack distributed systems in which multiple layers, (sub)systems, and dynamics interact together. Hence, they should leverage a secure and resilient networking architecture, a robust consensus protocol, and a safe environment for building higher-level applications. Although most of the deployed blockchains need better scalability and well-aligned incentives to unleash their full potential, their security is undoubtedly a critical factor for their success. As these systems are actively being developed and deployed, it is often challenging to understand how secure they are, or what security implications are introduced by some specific components they consist of. Moreover, due to their complexity and novelty (e.g., built-in protocol incentives), their security assessment and analysis requires a more holistic view than in the case of traditional distributed systems.

Although some standardization efforts have already been undertaken, they are either specific to a particular platform [1] or still under development [2], [3]. Hence, there is a lack of platform-agnostic standards in blockchain implementation, interoperability, services, and applications, as well as the analysis of its security threats [4], [5]. All of these areas are challenging, and it might take years until they are standardized and agreed upon across a diverse spectrum of stakeholders. In this work, we aim to contribute to the standardization of security threat analysis. We believe that it is critical to provide blockchain stakeholders (developers, users, standardization bodies, regulators, etc.) with a comprehensive systematization of knowledge about the security and privacy aspects of today's blockchain systems.

In this work, we aim to achieve this goal, with a particular focus on system design and architectural aspects. We do not limit our work to an enumeration of security issues, but additionally, discuss the origins of those issues while listing possible countermeasures and mitigation techniques together with their potential implications. As our main contribution, we propose the security reference architecture (SRA) for blockchains, which is based on models that demonstrate the stacked hierarchy of different threat categories (similar to the ISO/OSI hierarchy [6]) and is inspired by security modeling performed in the cloud computing [7], [8]. As our next contribution, we enrich the threat-risk assessment standard ISO/IEC 15408 [9] to fit the blockchain infrastructure. We achieve this by embedding the stacked model into this standard.

This article is based on our previous work outlining the security reference architecture [10]. We substantially modify and extend it by the following:

- We provide a theoretical background related to the security reference architecture and the environment of blockchains, their types, failure models, consensus protocols, design goals, and means to achieve these goals.

Manuscript received October 22, 2019; revised August 4, 2020; accepted September 20, 2020. Date of publication October 26, 2020; date of current version February 24, 2021. This work was supported in part by the NRF, Prime Minister's Office, Singapore, through its National Cybersecurity Research and Development Programme under Award NRF2016NCR-NCR002-028, and administered by the National Cybersecurity Research and Development Directorate, in part by the Czech Science Foundation under Project 20-07487S, in part by the H2020 ECSEL Project VALU3S under Grant 876852, and in part by the Internal Project of Brno University of Technology under Grant FIT-S-20-6427. *(Corresponding author: Ivan Homoliak.)*

Ivan Homoliak is with the Faculty of Information Technology, Brno University of Technology, 601 90 Brno, Czech Republic (e-mail: ihomoliak@fit.vutbr.cz).

Sarad Venugopalan, Daniël Reijsbergen, Qingze Hum, Richard Schumi, and Pawel Szalachowski are with the iTrust, Center for Research in Cyber Security, Singapore University of Technology and Design, Singapore.

Digital Object Identifier 10.1109/COMST.2020.3033665

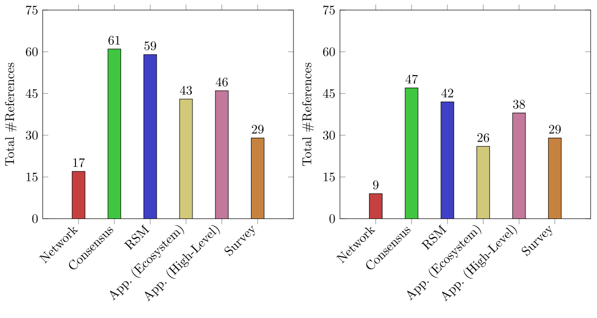
- For each layer, we model particular attacks or their categories through vulnerability/threat/defense graphs, while we provide reasoning about several important relations and causalities in these graphs.
- We modify and significantly extend the application layer, where we propose a novel functionality-oriented categorization, as opposed to the application-domain-oriented categorizations presented in related work [11], [12].
- We extend and revise the consensus layer by mining-pool-specific attacks, time-spoofing attacks, and we provide a more fine-grained categorization.
- For each layer, we present an incident table that lists and briefly describes examples of attacks that have occurred worldwide: either caused by malicious parties or by researchers who demonstrated their practical feasibility.
- In the lessons learned, we describe the hierarchy of security dependencies among particular categories, based on which, we propose a methodology for designers of blockchain platforms and applications.

The rest of the paper is organized as follows: We describe the scope and methodology of our research as well as quantitative analysis of the included literature in Section II. In Section III, we summarize the background on blockchain systems. Next, in Section IV we introduce the security reference architecture whose layers are discussed in the follow-up sections. In detail, Section V deals with the security and privacy of the network layer, Section VI focuses on the consensus layer, Section VII overviews the replicated state machine layer, and Section VIII with Section IX describe applications built on top of the blockchain. In Section X, we elaborate on lessons learned and propose a methodology for designers of blockchain-based solutions. We discuss the limitations of our work in Section XI and we compare our research to related work in Section XII. Finally, we conclude the paper in Section XIII.

II. Methodology & Scope

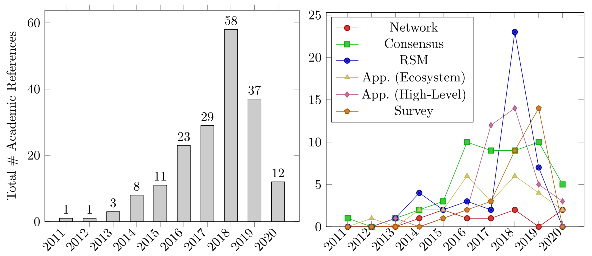
In contrast to conventional survey approaches, such as grounded theory for rigorous literature review [13], we do not sample included research from existing databases queried with specific terms. Instead, we study and analyze security-oriented literature for vulnerabilities and threats related to the blockchain infrastructure. The literature that we select as the input mainly covers existing blockchain-oriented surveys as well as various conferences and journals in security and distributed computing. Moreover, we include other materials published in preprints, whitepapers, products' documentation, and forums, which are also related.

We aim to consolidate the literature, categorize found vulnerabilities and threats according to their origin, and as a result, we create four main categories (also referred to as layers). At the level of particular main categories, we apply sub-categorization that is based on the existing knowledge and operation principles specific to such subcategories, especially concerning the security implications. If some subcategories impose equivalent security implications, we merge them into a single subcategory. See the road-map of all the categories in Figure 4. Our next aim is to indicate and explain the co-occurrences or relations of multiple threats, either at the same main category or across more categories.



(a) All references (b) Academic references

Fig. 1. Blockchain-specific references per category.



(a) #References per year (b) #References per year/category

Fig. 2. Blockchain-specific academic references over time.

The scope of our work mainly covers aspects related to the blockchain core elements, while we mention common operational security issues (e.g., key management in the blockchain ecosystem) and countermeasures only tangentially if required. Similarly, we do not pursue threats that emerge outside of the blockchain infrastructure and outside of the extra infrastructure required for certain blockchain-based applications. Out-of-scope examples are remote exploitation of arbitrary devices (e.g., covert mining/crypto-jacking [14]) and issues related to using blockchain explorers (e.g., spoofing attacks, availability issues). Examples of vulnerabilities that we assume only tangentially are semantic bugs and programming errors in the infrastructure of the blockchains – we assume that core blockchain infrastructure is implemented correctly, uses secure cryptographic primitives, and provides expected functionality.

A. Quantitative Analysis

For a quantitative summary of the literature, we considered only the references from the main sections that correspond to particular layers of the proposed stacked model (i.e., Section V to Section IX) and related work (Section XII). We excluded references from other sections and the references on the examples of incidents (Appendix C). Each assumed reference was labeled with three flags to indicate:

- The category to which it belongs. Note that some papers belong to multiple categories.
- Whether it is specifically related to blockchains.
- Whether it was written by academics (i.e., such that their affiliation was listed in the document). Note that the

positive value of this flag covers also the papers that have not been peer-reviewed but which appeared on public repositories (e.g., arXiv and Cryptology ePrint Archive). In sum, we collected 276 references, of which 247 were blockchain-specific, 211 were written by academics, and 180 met both criteria. In Figure 1, we show the number of references in each of the main categories for (a) all papers and (b) the academic papers. A total of 11 papers were found to belong to two categories. We observe that most of the references are presented at the application layer, which we conjecture that occurred because each application running on the blockchain might introduce new attack vectors.

The evolution of the number of references over time is depicted in Figure 2, where we observe that the number of academic papers about blockchains and security has increased steadily, although 2018 was the year with the highest number of references selected for the current work.[1] Of the four layers discussed in this article, the network layer has the lowest number of blockchain-focused papers. A possible reason for it could be that among the remaining layers, the consensus and replicated state machine (RSM) layers are more specific to blockchains, while the application layer is popular among practitioners building on top of other layers. Finally, we observe the number of surveys is steadily growing (culminating in 2019), which indicates increasing interest in this domain.

III. Blockchains at a Glance

The blockchain is a data structure representing an append-only distributed ledger that consists of entries (a.k.a., transactions) aggregated within ordered blocks. The order of the blocks is agreed upon by mutually untrusting participants running a consensus protocol – these participants are also referred to as nodes. The blockchain is resistant against modifications by design since blocks are linked using a cryptographic hash function, and each new block has to be agreed upon by nodes running a consensus protocol.

A transaction is an elementary data entry that may contain arbitrary data, e.g., an order to transfer native cryptocurrency (i.e., crypto-tokens), a piece of application code (i.e., smart contract), the execution orders of such application code, etc. Transactions sent to a blockchain are validated by all nodes that maintain a replicated state of the blockchain.

Blockchains were initially introduced as a means of coping with the centralization of monetary assets management, resulting in their most popular application – a decentralized cryptocurrency with a native crypto-token. Nevertheless, other blockchain applications have emerged, benefiting from features other than decentralization, e.g., privacy, energy efficiency, throughput, etc. For the review of the inherent and non-inherent features of the blockchains, we refer the reader to Appendix A.

A. Involved Parties

Blockchains usually involve three native types of parties that can be organized into a hierarchy, according to the actions that they perform (see Figure 3):

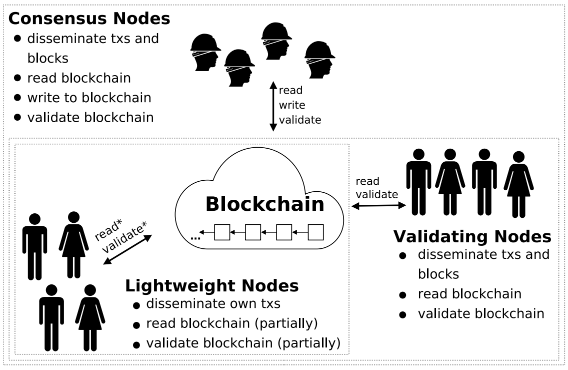


Fig. 3. Involved parties with their interactions and hierarchy.

(1) Consensus nodes (a.k.a., *miners* in Proof-of-Resource protocols) actively participate in the underlying consensus protocol. These nodes can read the blockchain and write to it by appending new transactions. Additionally, they can validate the blockchain and thus check whether writes of other consensus nodes are correct. Consensus nodes can prevent malicious behaviors (e.g., by not appending invalid transactions, or ignoring an incorrect chain).

(2) Validating nodes read the entire blockchain, validate it, and disseminate transactions. Unlike consensus nodes, validating nodes cannot write to the blockchain, and thus they cannot prevent malicious behaviors. On the other hand, they can detect malicious behavior since they possess copies of the entire blockchain.

(3) Lightweight nodes (a.k.a., clients or Simplified Payment Verification (SPV) clients) benefit from most of the blockchain functionalities, but they are equipped only with limited information about the blockchain. These nodes can read only fragments of the blockchain (usually block headers) and validate only a small number of transactions that concern them, while they rely on consensus and validating nodes. Therefore, they can detect only a limited set of attacks, pertaining to their own transactions.

Additional Involved Parties: Note that besides native types of involved parties, many applications using or running on the blockchain introduce their own (centralized) components.

B. Types of Blockchains

Based on how a new node enters a consensus protocol, we distinguish the following blockchain types:

Permissionless blockchains allow anyone to join the consensus protocol without permission. To prevent Sybil attacks, this type of blockchains usually requires consensus nodes to establish their identities by running a Proof-of-Resource protocol, where the consensus power of a node is proportional to its resources allocated.

Permissioned blockchains require a consensus node to obtain permission to join the consensus protocol from a centralized or federated authority(ies), while nodes usually have equal consensus power (i.e., one vote per node).

Semi-Permissionless blockchains require a consensus node to obtain some form of permission (i.e., stake) before joining the protocol; however, such permission can be given by

[1] Note that we started to work on this survey in early 2019.

any consensus node. The consensus power of a node is proportional to the stake that it has.

C. Design Goals of Consensus Protocols

1) Standard Design Goals–Liveness and Safety: **Liveness** ensures that all valid transactions are eventually processed – i.e., if a transaction is received by a single honest node, it will eventually be delivered to all honest nodes. **Safety** ensures that if an honest node accepts (or rejects) a transaction, then all other honest nodes make the same decision. Usually, consensus protocols satisfy safety and liveness only under certain assumptions: the minimal fraction of honest consensus power or the maximal fraction of adversarial consensus power. With regard to safety, literature often uses the term *finality* and *time to finality*. **Finality** represents the sequence of the blocks from the genesis block up to the block B, where it can be assumed that this sequence of blocks is infeasible to overturn. To reach finality up to the block B, several successive blocks need to be appended after B – the number of such blocks is referred to as *the number of confirmations*.

2) Specific Design Goals: As a result of this study, we learned that standard design goals of the consensus protocol should be amended by specific design goals related to the type of the blockchain. In permissionless type, *elimination of Sybil entities*, *a fresh and fair leader/committee election*, and *non-interactive verification of the consensus result* is required to meet. In contrast, the (semi)-permissionless types do not require the elimination of Sybil entities (see details in Section X-C).

3) Means to Achieve Design Goals:

a) Simulation of the verifiable random function (VRF): To ensure a fresh and fair leader/committee election, all consensus nodes should contribute to the pseudo-randomness generation that determines the fresh result of the election. This can be captured by the concept of the VRF [15], which ensures the unpredictability and fairness of the election process. Therefore, the leader/committee election process can be viewed as a simulation of VRF [16]. Due to the properties of VRF, the correctness of the election result can be verified non-interactively after the election took place.

b) Incentive and rewarding schemes: An important aspect for protocol designers is to include a rewarding/incentive scheme that motivates consensus nodes to participate honestly in the protocol. In the context of public (permissionless) blockchains that introduce their native crypto-tokens, this is achieved by block creation rewards as well as transaction fees, and optionally penalties for misbehavior. Transaction fees and block creation rewards are attributed to the consensus node(s) that create a valid block (e.g., [17]), although alternative incentive schemes rewarding more consensus nodes at the same time are also possible (e.g., [18]). While transaction fees are included in a particular transaction, the block reward is usually part of the first transaction in the block (a.k.a., *coinbase* transaction).

D. Basis of Consensus Protocols

Lottery and *voting* are two marginal techniques that deal with the establishment of a consensus [19]. However, in addition to them, their combinations have become popular.

Lottery-Based Protocols: These protocols provide consensus by running a lottery that elects a leader/committee, who produces the block. The advantages of lottery-based approaches are a small network traffic overheads and high scalability since the process is usually non-interactive (e.g., [17], [20], [21]). However, a disadvantage of this approach is the possibility of multiple "winners" being elected, who propose conflicting blocks, which naturally leads to inconsistencies called *forks*. Forks are resolved by fork-choice rules, which compute the difficulty of each branch and select the one. For the *longest chain rule*, the chain with the largest number of blocks is selected in the case of a conflict, while for the *strongest chain rule*, the selection criteria involve the quality of each block in the chain (e.g., [18], [22], [23], [24], [25]). Note that the possibility of forks in this category of protocols causes an increase of the time to finality, which in turn might enable some attacks such as double-spending.

Voting-Based Protocols: In this group of protocols, the agreement on transactions is reached through the votes of all participants. Examples include Byzantine Fault Tolerant (BFT) protocols – which require the consensus of a majority quorum (usually (2/3) of all consensus nodes (e.g., [26], [27], [28], [29], [30]). The advantage of this category is a low-latency finality due to a negligible likelihood of forks. The protocols from this group suffer from low scalability, and thus their throughput forms a trade-off with scalability (i.e., the higher the number of nodes, the lower the throughput).

Combinations: To improve the scalability of voting-based protocols, it is desirable to shrink the number of consensus nodes participating in the voting by a lottery, so that only nodes of such a committee vote for a block (e.g., [29], [31], [32], [33], [34]). Another option to reduce active voting nodes is to split them into several groups (a.k.a., *shards*) that run a consensus protocol in parallel (e.g., [35], [36]). Such a setting further increases the throughput in contrast to the single-group option, but on the other hand, it requires a mechanism that accomplishes inter-shard transactions.

E. Failure Models in Distributed Consensus Protocols

The relevant literature mentions two main failure models for consensus protocols [37]:

Fail-Stop Failures: A node either stops its operation or continues to operate, while obviously exposing its faulty behavior to other nodes. Hence, all other nodes are aware of the faulty state of that node (e.g., tolerated in Paxos [38], Raft [39], Viewstamped Replication [40]).

Byzantine Failures: In this model, the failed nodes (a.k.a., Byzantine nodes) may perform arbitrary actions, including malicious behavior targeting the consensus protocol and collusions with other Byzantine nodes. Hence, the Byzantine failure model is of particular interest to security-critical applications, such as blockchains (e.g., Nakamoto's consensus [17], pure BFT protocols [26], [27], [28], [41], and hybrid protocols [31], [35], [36]).

IV. The Security Reference Architecture

We present two models of the security reference architecture, which facilitate systematic studying of vulnerabilities and threats related to the blockchains and applications running on top of them. First, we introduce the stacked model, which we then project into the threat-risk assessment model.

A. *Stacked Model*

To classify the security aspects of blockchains, we utilize a stacked model consisting of four layers (see Figure 4). A similar stacked model was already proposed in the literature [16], but in contrast to it, we preserve only such a granularity level that enables us to isolate security threats and their nature, which is the key focus of our work. In the following, we briefly describe each layer.

(1) The network layer consists of the data representation and network services planes. The data representation plane deals with the storage, encoding, and protection of data, while the network service plane contains the discovery and communication with protocol peers, addressing, routing, and naming services.

(2) The consensus layer deals with the ordering of transactions, and we divide it into three main categories according to the protocol type: Byzantine Fault Tolerant, Proof-of-Resource, and Proof-of-Stake protocols.

(3) The replicated state machine (RSM) layer deals with the interpretation of transactions, according to which the state of the blockchain is updated. In this layer, transactions are categorized into two parts, where the first part deals with the privacy of data in transactions as well as the privacy of the users who created them, and the second part – smart contracts – deals with the security and safety aspects of decentralized code execution in this environment.

(4) The application layer contains the most common end-user functionalities and services. We divide this layer into two groups. The first group represents the applications that provide common functionalities for most of the higher-level blockchain applications, and it contains the following categories: wallets, exchanges, oracles, filesystems, identity management, and secure timestamping. We refer to this group as applications of the blockchain ecosystem. The next group of application types resides at a higher level and focuses on providing certain end-user functionality. This group contains categories such as e-voting, notaries, identity management, auctions, escrows, etc. We found out that these higher-level applications inherit security aspects from particular categories in the underlying ecosystem group (see Figure 15).

Throughout the paper, we summarize components and categories of particular layers of the reference architecture with their respective security threats, their origin, and corresponding countermeasures and/or mitigation techniques.

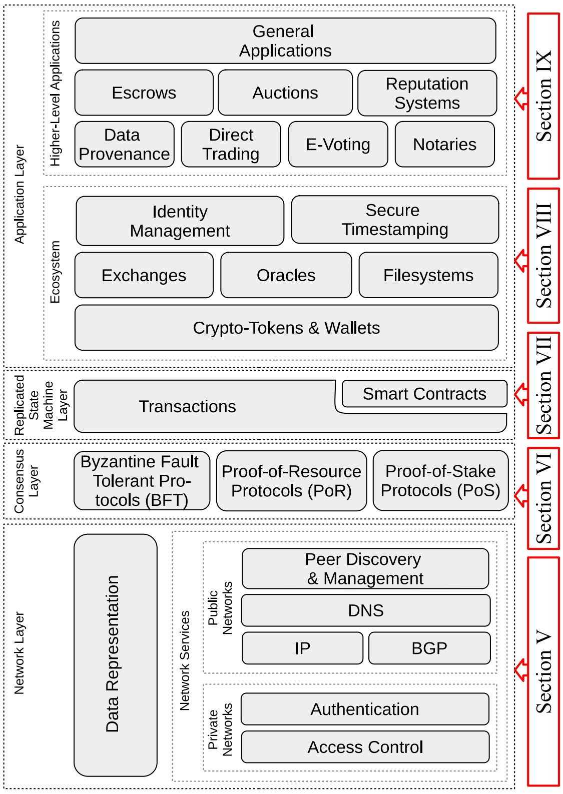


Fig. 4. Stacked model of the security reference architecture.

B. *Threat-Risk Assessment Model*

To better capture the security-related aspects of blockchain systems, we introduce a threat-risk model (see Figure 5) that is based on the template of ISO/IEC 15408 [9] and projection of our stacked model (see Figure 4). This model includes the following components and actors:

Owners are blockchain users who run any type of node and they exist at the application layer and the consensus layer. Owners possess crypto-tokens, and they might use or provide blockchain-based applications and services. Additionally, owners involve consensus nodes that earn crypto-tokens from running the consensus protocol.

Assets are present at the application layer, and they consist of monetary value (i.e., crypto-tokens or other tokens) as well as the availability of application-layer services and functionalities built on top of blockchains (e.g., notaries, escrows, data provenance, auctions). The authenticity of users, the privacy of users, and the privacy of data might also be considered as application-specific assets. Furthermore, we include here the reputation of service providers using the blockchain services.

Threat agents are spread across all the layers of the stacked model, and they mostly involve malicious users whose intention is to steal assets, break functionalities, or disrupt services. However, threat agents might also be inadvertent entities, such as developers of smart contracts who unintentionally create bugs and designers of blockchain applications who make mistakes in the design or ignore some issues.

Threats facilitate various attacks on assets, and they exist at all layers of the stacked model. Threats arise from vulnerabilities in the network, smart contracts, applications, from consensus protocol deviations, violations of consensus protocol assumptions.

Countermeasures protect owners from threats by minimizing the risk of compromising/losing the assets. Alike the threats and threat agents, countermeasures can be applied at each of the layers of our stacked model, and they involve various security/privacy/safety solutions, incentive schemes, reputation techniques, best practices, etc. Nevertheless, we emphasize that their utilization usually imposes some limitations such as higher complexity and additional performance overheads (e.g., resulting in decreased throughput).

Risks are related to the application layer, and they are caused by threats and their agents. Risks may lead to a loss of monetary assets, a loss of privacy, a loss of reputation, service malfunctions, and disruptions of services and applications (i.e., availability issues).

The owners wish to minimize the risk caused by threats that arise from threat agents. Within our stacked model, different threat agents appear at each layer. At the **network layer**, there are service providers including parties managing IP addresses and DNS names. The threats at this layer arise from man-in-the-middle (MITM) attacks, network partitioning, de-anonymization, and availability attacks. Countermeasures contain protection of availability, naming, routing, anonymity, and data. At the **consensus layer**, consensus nodes may be malicious and wish to alter the outcome of the consensus protocol by deviating from it. Moreover, if they are powerful enough, malicious nodes might violate assumptions of consensus protocols to take over the execution of the protocol or cause its disruption. The countermeasures include well-designed economic incentives, strong consistency, decentralization, and fast finality solutions. At the **RSM layer**, the threat agents may stand for developers who (un)intentionally introduce semantic bugs in smart contracts (intentional bugs represent backdoors) as well as users and external adversaries running lightweight nodes who pose threats due to the exploitation of such bugs. Countermeasures include safe languages, static/dynamic analysis, formal verification, audits, best practices, and design patterns. Other threats of the RSM layer are related to compromising the privacy of data and user identities with mitigation techniques involving mixers, privacy-preserving cryptography constructs (e.g., non-interactive zero-knowledge proofs (NIZKs), ring signatures, blinding signatures, homomorphic encryption) as well as usage of trusted hardware (respecting its assumptions and attacker models declared). At the **application layer**, threat agents are broad and involve arbitrary internal or external adversaries such as users, service providers, malware, designers of applications and services, manufactures of trusted execution environments (TEE) for concerned applications (e.g., oracles, auctions), authorities in the case of applications that require them for arbitration (e.g., escrows, auctions) or filtering of users (e.g., e-voting, auctions), token issuers. The threats on this layer might arise from false data feeds, censorship by application-specific authorities (e.g., auctions, e-voting), front running attacks, disruption of the availability of centralized components, compromising application-level privacy, misbehaving of the token issuer, misbehaving of manufacturer of TEE or permanent hardware (HW) faults in TEE. Examples of mitigation techniques are multi-factor authentication, HW wallets with displays for signing transactions, redundancy/distributions of some centralized components, reputation systems, and privacy preserving-constructs as part of the applications themselves. We elaborate closer on vulnerabilities, threats, and countermeasures (or mitigation techniques) related to each layer of the stacked model in the following sections.

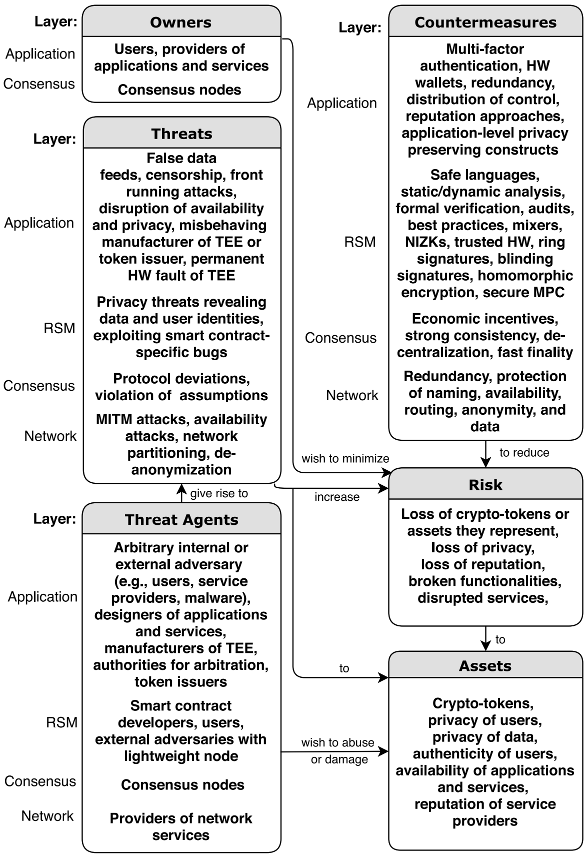

Layer:
Owners
Application
Users, providers of applications and services
Consensus
Consensus nodes
Layer:
Countermeasures
Application
Multi-factor authentication, HW wallets, redundancy, distribution of control, reputation approaches, application-level privacy preserving constructs
RSM
Safe languages, static/dynamic analysis, formal verification, audits, best practices, mixers, NIZKs, trusted HW, ring signatures, blinding signatures, homomorphic encryption, secure MPC
Consensus
Economic incentives, strong consistency, decentralization, fast finality
Network
Redundancy, protection of naming, availability, routing, anonymity, and data
Layer:
Threats
Application
False data feeds, censorship, front running attacks, disruption of availability and privacy, misbehaving manufacturer of TEE or token issuer, permanent HW fault of TEE
RSM
Privacy threats revealing data and user identities, exploiting smart contract-specific bugs
Consensus
Protocol deviations, violation of assumptions
Network
MITM attacks, availability attacks, network partitioning, de-anonymization
wish to minimize
to reduce
give rise to
increase
Risk
Loss of crypto-tokens or assets they represent, loss of privacy, loss of reputation, broken functionalities, disrupted services,
Layer:
Threat Agents
Application
Arbitrary internal or external adversary (e.g., users, service providers, malware), designers of applications and services, manufacturers of TEE, authorities for arbitration, token issuers
RSM
Smart contract developers, users, external adversaries with lightweight node
Consensus
Consensus nodes
Network
Providers of network services
to
to
Assets
wish to abuse or damage
Crypto-tokens, privacy of users, privacy of data, authenticity of users, availability of applications and services, reputation of service providers

Fig. 5. Threat-risk assessment model of the security reference architecture.

Involved Parties & Blockchain's Life-Cycle: In Section III, we presented several types of involved parties in the blockchain infrastructure (see Figure 3). We emphasize that these parties are involved in the operational stage of the blockchain's life-cycle. However, in the design and development stages of the blockchain's

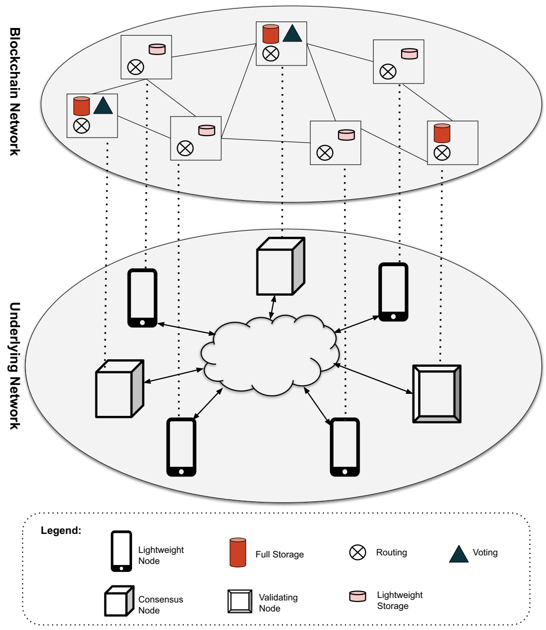


Fig. 6. Overlaying of blockchains over a private/public network.

life-cycle, programmers and designers should also be considered as potential threat agents who influence the security aspects of the whole blockchain infrastructure (regardless of whether their intention is malicious or not). This is of great concern especially for applications built on top of blockchains (i.e., at the application layer) since these applications are usually not thoroughly reviewed by the community or public, as it is typical for other (lower) layers.

V. Network Layer

Blockchains usually introduce peer-to-peer overlay networks built on top of other networks (see Figure 6). Hence, blockchains inherit security and privacy issues from their underlying networks. In our model (see Figure 4), we divide the network layer into *data representation* and *network services* sub-planes. The data representation plane is protected by cryptographic primitives that ensure data integrity, user authentication, and optionally confidentiality, privacy, anonymity, non-repudiation, and accountability. The main services provided by the network layer are peer management and discovery, which rely on the internals of the underlying network, such as domain name resolution (i.e., DNS) or network routing protocols. Based on permission to join the blockchain system, the networks are either private or public. In the following, we discuss the pros and cons of private and public networks as well as their security threats and mitigation techniques.

A. Private Networks

A private network ensures low latency, a centralized administration, privacy, and meeting regulatory obligations (e.g., HIPAA[2] for healthcare data). The organization owning the network provides access to local participants as well as to external ones when required; hence, systems deploying private networks belong to the group of permissioned private blockchains. Private networks inherently provide authentication and access control.

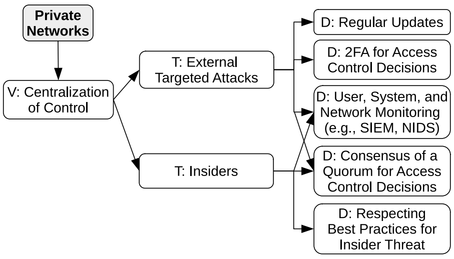

Private Networks
V: Centralization of Control
T: External Targeted Attacks
T: Insiders
D: Regular Updates
D: 2FA for Access Control Decisions
D: User, System, and Network Monitoring (e.g., SIEM, NIDS)
D: Consensus of a Quorum for Access Control Decisions
D: Respecting Best Practices for Insider Threat

Fig. 7. Vulnerabilities, threats, and defenses in private networks (network layer).

1) Pros: *Access control* is achieved by centralized authentication of users. A private network has full control over routing paths and physical resources used, which enables suitable regulation of the network topology with regard to the given requirements. *Data privacy* is ensured by permissioned settings. *User identities* might only be revealed within a private group of nodes. *Fine-grained authorization controls* are applied by the operators of the network resources to implement the security principle of minimal exposure and thus mitigate insider threat attacks on a local network. *Resource availability* is easier to manage and foresee, as all network participants and the deployment scenario are known ahead of time.

2) Cons: Virtual Private Network (VPN) connectivity is required to communicate between private networks spread over different geographical locations. While VPNs are in general secure, they inherit the disadvantages of running a service over the Internet. Private networks are suitable only for permissioned blockchains.

3) Security Threats and Countermeasures: We present a taxonomy of vulnerabilities, threats, and defenses against them in Figure 7. We identified *insiders* and *external targeted attacks* as the specific security threats for the (permissioned) blockchains running over the private networks. These threats are possible due to a centralization of access control that might occur in private networks, and thus permissioned blockchains. An external attacker might exploit a network or system vulnerability and obtain access to an element responsible for access control to the blockchain. In the case of the insider threat, she may already have the necessary privileges or obtain them by exploiting the system, network, or organization vulnerabilities. As a result, the insider might add malicious consensus nodes into the network or remove legitimate ones, and thereby increase the adversarial consensus power that is manifested at the consensus layer. In turn, this may lead to plenty of attacks occurring at the consensus layer

[2] Health insurance portability and accountability act, https://hipaa.com/.

(see Section VI) such as attacks on violation of protocol assumptions.

Countermeasures include regular software updates, monitoring of users, network, and systems (e.g., by SIEM, anti-virus software, intrusion detection systems), prevention techniques that minimize trust and maximize trustworthiness such as two-factor authentication for access control decisions (effective for the external attacker), as well as respecting best practices for mitigation of insider threat [42]. Another option of coping with the centralization of access control is to embed decentralized access control into the consensus layer and thus mandating a requirement on reaching a consensus of a quorum of nodes for each access control decision. In contrast to the other mentioned countermeasures, the embedding of access control into the consensus layer is more effective since it eliminates a single-point-of-failure of centralized access control service, and thus it makes an increase of adversarial consensus power more difficult for the attacker – the attacker has to compromise a high number of independent consensus nodes.

4) Side Effects of Countermeasures: Most of the mentioned countermeasures do not cause negative effects on the features of the blockchains under normal circumstances. The exception might be embedding of access control into the consensus layer under the assumption that the set of consensus nodes in the network is extremely dynamic, and thus nodes are entering and leaving the blockchain very often. In such a case, the throughput of the blockchain might be decreased.

B. Public Networks/The Internet

Public networks provide high decentralization, openness, and low entry barrier, while network latency, privacy, and network control are put aside. These networks are naturally required by all public (permissionless) blockchain systems.

1) Pros: High availability is attractive to multi-homed nodes since they have alternate routes to send and receive messages. Multi-homed nodes may benefit from disseminating blocks across multiple channels, thereby increasing the chance of blocks being appended to the blockchain. *High decentralization* is achieved through geographical dispersion of nodes. Public peer-to-peer (p2p) networks are harder to shut down. *Openness and low entry barriers* on the Internet are achieved through wide adoption, technology interoperability (e.g., using TCP/IP), economic (e.g., low cost of broadband connection), and societal (e.g., resistance to regulations) factors.

2) Cons: Single-point-of-failure – DNS with its hierarchy, IP addresses, and autonomous systems (ASes) are managed by centralized parties – Internet Corporation for Assigned Names and Numbers (ICANN); in particular, Internet Assigned Numbers Authority (IANA). *External adversaries* pose a threat to public networks. These adversaries can be classified based on their capabilities to which the blockchain network may be exposed to [43]: (1) resources under attacker control (e.g., botnets, DNS and BGP servers), (2) stolen or masqueraded identities (e.g., IP addresses participating in an eclipse attack or route manipulation), (3) MITM attacker (i.e., eavesdropping and spoofing), (4) the exploitation of common network vulnerabilities, (5) revealing secrets (e.g., de-anonymizing peers). *Efficiency* – although an average Internet bandwidth has been improved in recent years, distribution of powerful infrastructure is not uniform, which results in a different latency among peers, and thus the overall latency of the network is increased; this might result in the loss of created blocks and thus wasting consensus power.

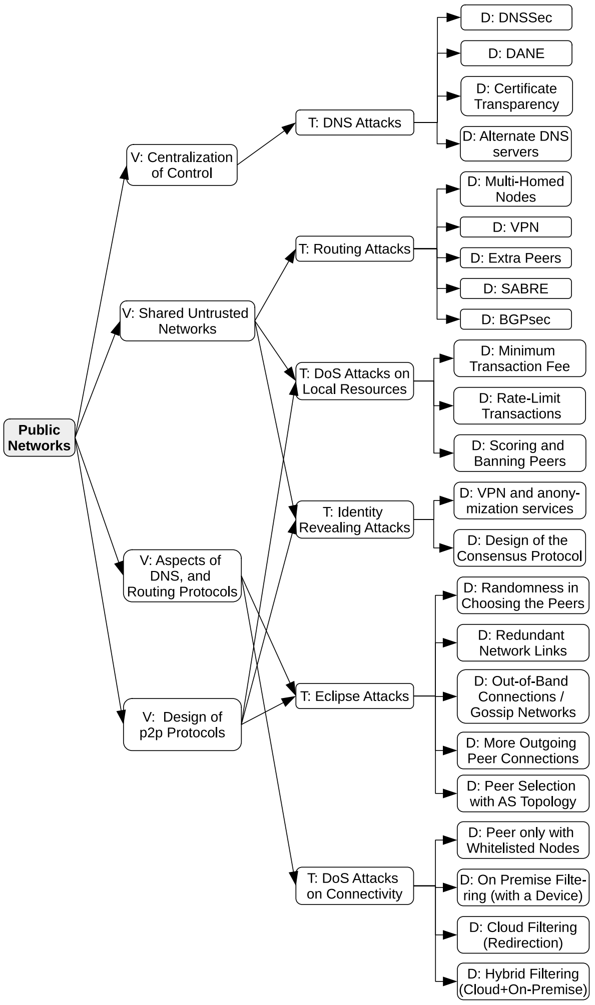

Public Networks
V: Centralization of Control
V: Shared Untrusted Networks
V: Aspects of DNS, and Routing Protocols
V: Design of p2p Protocols
T: DNS Attacks
T: Routing Attacks
T: DoS Attacks on Local Resources
T: Identity Revealing Attacks
T: Eclipse Attacks
T: DoS Attacks on Connectivity
D: DNSSec
D: DANE
D: Certificate Transparency
D: Alternate DNS servers
D: Multi-Homed Nodes
D: VPN
D: Extra Peers
D: SABRE
D: BGPsec
D: Minimum Transaction Fee
D: Rate-Limit Transactions
D: Scoring and Banning Peers
D: VPN and anonymization services
D: Design of the Consensus Protocol
D: Randomness in Choosing the Peers
D: Redundant Network Links
D: Out-of-Band Connections / Gossip Networks
D: More Outgoing Peer Connections
D: Peer Selection with AS Topology
D: Peer only with Whitelisted Nodes
D: On Premise Filtering (with a Device)
D: Cloud Filtering (Redirection)
D: Hybrid Filtering (Cloud+On-Premise)

Fig. 8. Vulnerabilities, threats, and defenses in public networks (network layer).

3) Security Threats and Countermeasures: We present a taxonomy of vulnerabilities, threats, and defenses related to public networks in Figure 8, while in Table VI of Appendix, we list several incidents that occurred in practice. In the following, we describe these threats as well as possible defense techniques.

DNS attacks arise from cache poisoning that mainly affects blockchains employing centralized DNS bootstrapping to retrieve online peers from a hard-coded list of DNS seeders. One *countermeasure* is a security extension of DNS, called DNSSEC, which provides authentication and data integrity. In addition to standard DNS, name resolution can also be made using alternate DNS servers [44].

Routing attacks are traffic route diversions, hijacking, or DoS attacks. Besides simple data eavesdropping or modification, these attacks may lead to network partitioning, which in turn raises the risks of 51% attacks or selfish mining attacks presented at the consensus layer (see Section VI). Apostolaki *et al.* [45] demonstrated that the Bitcoin protocol is vulnerable to BGP routing attacks where the attacker controlling a transit autonomous system (AS) can modify inter-domain routes for a few Bitcoin nodes and cause network partitioning. However, perpetration of this attack reveals the identity of the malicious AS, which might have immediate reputation consequences.

Countermeasures are multi-homed nodes (or using VPN) for route diversity, choosing extra peers whose connections do not pass through the same ASes, preference of peers hosted on the same AS within the same /24 prefix (to reduce risk of partitions), and fetching the same block from multiple peers [45]. Another mitigation is SABRE [46], a secure relay network that runs alongside with the Bitcoin network. Further, BGPsec is a security extension for BGP used between neighboring ASes, and it assures route origin and propagation by cryptographic verification.

Eclipse attacks aim to hijack all connections of a node to its peers in a blockchain network. Consequently, all traffic received and sent by the node is under the full control of the attacker. Eclipse attacks arise from threats on DNS and routing in the network, and they may be a result of vulnerabilities in p2p protocols [47], [48], [49]. Eclipse attacks increase chances of selfish mining and double-spending attacks (see Section VI) – the eclipsed victims may unknowingly vote for an attacker's chain, and thus cause a network partitioning. Erebus [50] is a stealthier attack causing network partitioning as compared to Apostolaki *et al.* [45]. However, Erebus is not a routing attack since it does not involve BGP prefix hijacking (which is easy to detect) and has a very small network traffic footprint. In Erebus, the attacker controls a large number of shadow IP addresses and influences the victim's peer selection mechanism to pick all outgoing connections with shadow IP addresses. This is achieved through slowly flooding the victim's peer tables by incoming connections from the attacker-controlled shadow IP addresses.

Countermeasures: Improving randomness in choosing peers was proposed in the work of Heilman *et al.* [47] by several rules that manage the peer table. Another mitigation strategy against eclipse attacks is to use redundant network links or out-of-band connections to verify transactions (e.g., by a blockchain explorer). Eclipse attacks can also be detected by employing out-of-band gossip networks (e.g., Web-servers) [51] that communicate with lightweight clients to exchange their views on the blockchain (i.e., block headers) along with native Web traffic. Erebus attacks can be made much harder by decreasing the size of the peer tables, increasing the number of peers, preferring the peers that provide fresh data, and incorporation the topology of ASes into the peer selection process. Also, note that countermeasures for DNS and routing attacks are applicable here as well.

DoS attacks on connectivity of consensus nodes may result in a loss of consensus power, thus preventing consensus nodes from being rewarded [52]. For validating nodes, this attack leads to a disruption of some blockchain-dependent services [53]. *Countermeasures:* One mitigation is to peer only with white-listed nodes. Methods to prevent volumetric DDoS include on-premise filtering (i.e., with an extra network device), cloud filtering (i.e., redirection of traffic through a cloud when DDoS is detected or through a cloud DDoS mitigation service), or hybrid filtering [54].

DoS attacks on resources such as memory and storage, may reduce the peering and consensus capabilities [55] of nodes. An example attack is flooding the network with low fee transactions (a.k.a., penny-flooding), which may cause memory pool depletion, resulting in a system crash. A possible mitigation is raising the minimum transaction fee and the rate-limit to the number of transactions. Several mitigating techniques were applied to Bitcoin [56] nodes including scoring DoS attacks and banning misbehaving peers. DoS attacks on connectivity may also target (additional) centralized elements of blockchain infrastructure, such as servers communicating with hosted wallets (see Section VIII-A), which in turn might lead to application layer attacks targeting clients of the wallets.

Identity revealing attacks are conducted by linking the IP address of a node with an identity propagated in transactions [57], [58]. Traffic analysis using Sybil listeners can reveal the linkage of node IP addresses and their transactions [59]. *Countermeasures* include using VPNs or anonymization services, such as Tor. See Section VII-A1 for further identity and privacy-protecting mechanisms at the RSM layer.

4) Side Effects of Countermeasures: The anonymization services cause deterioration of connectivity for consensus nodes, and these nodes might not distribute the created block on time and thus lose their reward. On the other hand, the slow connectivity of anonymization services can be acceptable for validating nodes and clients transmitting messages related to the creation and validation of their transactions. The trade-off for connectivity and anonymity can be provided by VPN services, which are fast but they are usually operated by a centralized party, and thus the risk of de-anonymization is higher than in the case of anonymization services. Increasing the number of outgoing peer connections as protection against eclipse attacks [50] can increase the volume of data that needs to be transferred, which might negatively impact the throughput of blockchains. Furthermore, fetching the same blocks from multiple peers [45] may contribute to the network congestion under certain configurations of blockchains focusing on a high throughput – this might lead to a slow down in keeping touch with the tip of the blockchain and thereby impact a response time of an application running on top of it. The response time of the application might be also impacted by cloud and hybrid filtering that relay some incoming traffic through 3rd party services.

VI. Consensus Layer

The consensus layer of the stacked model deals with the ordering of transactions, while the interpretation of them is left for the RSM layer (see Section VII). The consensus layer includes three main categories of consensus protocols concerning different principles of operation and thus their security aspects. First, we focus on the security aspects that are generic to all categories, and then we detail each category.

A. Generic Attacks

We present a taxonomy of the generic threats to all types of consensus protocols, their origins, and defenses against them in Figure 9. These threats originate mainly from *violation of protocol assumptions* but also due to *a long time to the finality* of some consensus protocols. In the following, we describe these threats as well as possible defense techniques.

1) Security Threats and Mitigations:

Adversarial Centralization of Consensus Power: In these attacks, a design assumption about the decentralized distribution of consensus power is violated. Examples of this category are *51% attacks* for PoR and PoS protocols as well as $\frac{1}{3}$ of *Byzantine nodes* for BFT protocols (and their combinations). In a 51% attack, the majority of the consensus power is held by the adversary, thus also the result of the protocol is under her control. In *Byzantine attacks*, a quorum of $\frac{1}{3}$ adversarial consensus nodes might cause the protocol to be disrupted or even halted. As a design-oriented countermeasure, it is important to promote decentralization by incentive schemes that reward honest participation and discourage [60] or punish [32], [61] protocol violations. Another mitigation that makes these attacks more expensive is a statistical analysis of sudden anomalies in the history of the consensus power distribution among nodes, which can be embedded in the fork-choice rule of the consensus protocol [62].

Breaking Network Assumptions: Protocols assuming synchronous or partially synchronous network delivery would inevitably fail when this assumption does not hold. For instance, this assumption can be violated in BFT protocols by *an unpredictable network scheduler*, as demonstrated on PBFT protocol [63]. This fact motivates asynchronous BFT protocols that can be based on threshold-based cryptography, which enables reliable and consistent broadcast [41], [63].

Time De-Synchronization Attacks: Usually, besides system time, nodes in PoW and PoS maintain network time that is computed as the median value of the time obtained from the peers. Such a time is often put into the block header, while nodes, upon receiving a block, validate whether it fits freshness constraints. An attacker can exploit this approach by connecting a significant number of nodes and propagate inaccurate timestamps, which can slow down or speed up the victim node's network time [64]. When such a desynchronized node creates a block, this block can be discarded by a network due to freshness constraints. To avoid de-synchronization attacks, a node can build a reputation list of trusted peers or employ a timestamping authority [65]. Another option to improve the accuracy of block timestamps is to compute them collaboratively by consensus nodes [18].

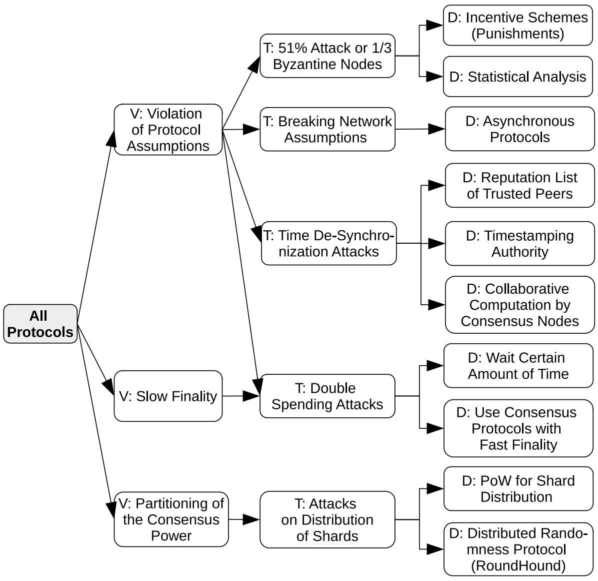

All Protocols
V: Violation of Protocol Assumptions
V: Slow Finality
V: Partitioning of the Consensus Power
T: 51% Attack or 1/3 Byzantine Nodes
T: Breaking Network Assumptions
T: Time De-Synchro-nization Attacks
T: Double Spending Attacks
T: Attacks on Distribution of Shards
D: Incentive Schemes (Punishments)
D: Statistical Analysis
D: Asynchronous Protocols
D: Reputation List of Trusted Peers
D: Timestamping Authority
D: Collaborative Computation by Consensus Nodes
D: Wait Certain Amount of Time
D: Use Consensus Protocols with Fast Finality
D: PoW for Shard Distribution
D: Distributed Rando-mness Protocol (RoundHound)

Fig. 9. Generic threats and defenses of the consensus layer.

Double-Spending Attack: This attack is possible due to the creation of two or more conflicting blocks with the same height, resulting in inconsistencies called *forks*. Thus, some crypto-tokens might be temporarily spent in both conflicting blocks, while only a single block is later included in the honest chain. A double-spending attack mainly affects consensus protocols with slow finality. This attack usually occurs as a consequence of 51% attacks.[3] To prevent this attack, it is recommended to wait a certain amount of time (i.e., time to the finality) until a block "is settled" or utilize consensus protocols with fast time to the finality, such as BFT protocols and their combinations.

Attacks on Shards: Sharding means that consensus nodes are distributed among subgroups (i.e., shards) such that each node only validates the transactions in its group. Shards operate in parallel and can achieve higher scalability and throughput since each shard has a throughput similar to an entire non-sharded blockchain. On the other hand,

[3] Note that in the case of PoR protocols, this attack may also occur as a consequence of the selfish mining attack.

sharding has the potential to harm security because each shard has a lower number of participating nodes than the entire blockchain, which means that it may be easier for the attacker to compromise a single shard than the entire blockchain [66], [67]. The main mitigation technique is to achieve a truly random distribution of nodes among the shards, and thus minimize the potential for adversaries to bias the randomness used for shard distribution. For example, Elastico [68] uses PoW to distribute nodes among shards, whereas Omniledger [35] uses a bias-resistant distributed randomness protocol (e.g., RandHound [69]). Poor design of sharding protocols may also lead to vulnerabilities such as replay attacks [70].

2) Side Effects of Countermeasures: Some generic countermeasures presented in the above text might impact the features of the blockchains. For example, the application layer countermeasure that uses the timestamping authority brings centralization issues, which might impact the availability of the service and enable misbehaving of the authority that might provide imprecise time. To cope with this issue, instead of using the application layer countermeasure, the computation of block timestamps can be embedded into the consensus protocol, e.g., by ensuring that multiple consensus nodes simultaneously contribute to global time using partial solutions [18].

Enriching a fork-choice rule by statistical analysis of the history [62] might protect against sudden changes in the distribution of the overall consensus power (e.g., rent-and-attack in PoW protocols), but not against a slow gradual increase of the consensus power by a coalition. Furthermore, this mitigation technique incentivizes consensus nodes to use stable identifications (i.e., the same key pairs) to increase the strength of the honest chain and thus the chances that it becomes the main chain after a temporary fork. This might impact the privacy and security of consensus nodes, who are disincentivized to rotate keys. Note that some privacy issues can be resolved at the RSM layer (see Section VII).

Using a BFT consensus protocol helps to significantly decrease the likelihood of double-spending attacks, but on the other hand, it worsens the scalability of the blockchain and thus throughput, especially in the case of a high number of consensus nodes. These issues can be resolved by combining a BFT voting protocol with lottery-based protocols that reduce the size of the actively communicating nodes to a small committee (e.g., [31], [32], [33], [34], [29]). The scalability of BFT protocols can be also improved by using threshold-based signature aggregation with a gossip-based communication pattern (e.g., [71], [72]).

Distributed randomness protocols and PoW for distribution of shards bring additional overheads, which may reduce the throughput of the blockchain; however, this reduction is negligible in contrast to the throughput improvement due to sharding.

B. Proof-of-Resource Protocols (PoR)

Protocols from this category require nodes to prove the spending of a scarce resource in a lottery-based fashion [19]. Scarce resources may stand for: *(1) Computation* that is represented by Proof-of-Work (PoW) protocols (e.g., Bitcoin, Ethereum). *(2) Storage* used in the setting of Proof-of-Space protocols [73] (e.g., Spacecoin [74], SpaceMint [75]). *(3) Crypto-tokens* spent for Proof-of-Burn protocols [76] (e.g., Slimcoin [77]). *(4) Combinations and modification* of the previous types, such as storage and computation, called Proof-of-Retrievability (e.g., Permacoin [78]) and storage over time, which is represented by Proof-of-Space protocols (e.g., Filecoin [79]). Another hybrid example of this category is a combination of PoR with elapsed time, such as in PeerCoin [80]. However, it is a philosophical question whether to consider elapsed time as a resource that is spent or as a stake that is invested – note that literature often categorizes PeerCoin as the first instance of a (hybrid) PoS protocol, hence we incline towards the second option.

PoR protocols belong to the first generation of consensus protocols, and they are mostly based on Nakamoto Consensus [17] that utilizes PoW, inheriting its pros (e.g., high scalability) and cons (e.g., low throughput). For the detailed analysis of several PoW designs, we refer the reader to [81].

1) Pros: In PoR protocols, malicious overriding of the history of the blockchain (or part of it) requires spending at least the same amount of resources as was spent for its creation. This is in contrast to the principles of PoS protocols, where a big enough coalition may override the history at almost no cost.

2) Cons: PoR protocols imply high operational costs. Moreover, these protocols provide only probabilistic finality, which enables attacks forking the last few blocks of the chain.

3) Security Threats and Mitigations: We present a taxonomy of the attacks related to PoR protocols, their origins, and defenses against them in Figure 10, while we list several real-world incidents in Table V of Appendix. In the following, we describe these attacks as well as possible defense techniques.

Selfish Mining: In selfish mining [82],[4] an adversary attempts to privately build a secret chain and reveal it to the public only when an honest chain is "catching up" with the secret one. The longest-chain rule causes honest miners to adopt the attacker's chain and invalidate the honest chain, thus wasting their consensus power. This attack is more efficient when the consensus power of a selfish miner reaches some threshold (e.g., 30%). The selfish mining strategy was later generalized [84] and extended to other variants that increase the profit of the attacker [85]. *Countermeasures:* For the case of the longest-chain rule, the first introduced mitigation is uniform tie-breaking [82], which tells consensus nodes to choose the chain to extend uniformly at random, regardless of which one they received first. However, this technique is less effective when assuming network delays [84]. As the longest-chain rule enables this attack, it is recommended to use other fork-choice rules that also

[4] Note that selfish mining is theoretically possible even in PoS protocols [83] but requiring them to have predictable randomness for the leader election, which is usually a design-oriented vulnerability (see Section III-C). However, in PoR, selfish mining is possible even with unpredictable randomness for the leader election.

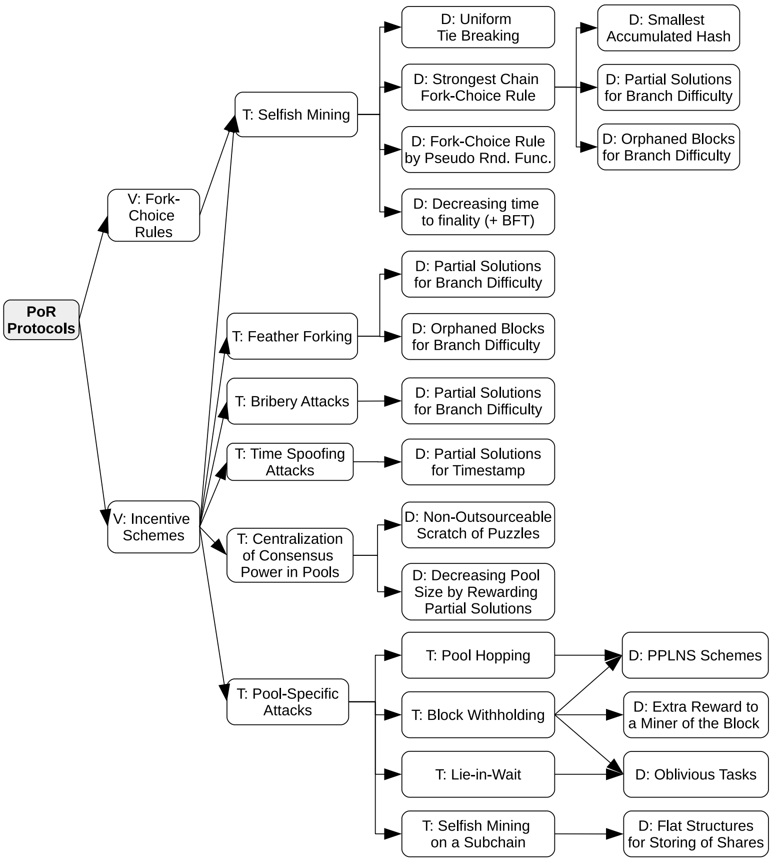

PoR Protocols
V: Fork-Choice Rules
V: Incentive Schemes
T: Selfish Mining
D: Uniform Tie Breaking
D: Strongest Chain Fork-Choice Rule
D: Fork-Choice Rule by Pseudo Rnd. Func.
D: Decreasing time to finality (+ BFT)
D: Smallest Accumulated Hash
D: Partial Solutions for Branch Difficulty
D: Orphaned Blocks for Branch Difficulty
T: Feather Forking
D: Partial Solutions for Branch Difficulty
D: Orphaned Blocks for Branch Difficulty
T: Bribery Attacks
D: Partial Solutions for Branch Difficulty
T: Time Spoofing Attacks
D: Partial Solutions for Timestamp
T: Centralization of Consensus Power in Pools
D: Non-Outsourceable Scratch of Puzzles
D: Decreasing Pool Size by Rewarding Partial Solutions
T: Pool-Specific Attacks
T: Pool Hopping
T: Block Withholding
T: Lie-in-Wait
T: Selfish Mining on a Subchain
D: PPLNS Schemes
D: Extra Reward to a Miner of the Block
D: Oblivious Tasks
D: Flat Structures for Storing of Shares

Fig. 10. Vulnerabilities, threats, and defenses of PoR protocols (consensus layer).

account for the quality of solutions and make the decision deterministic, as opposed to a uniform tie-breaking. An example of such a rule is to select the block based on the smallest hash value of the header. Another example is to include partial solutions [18], [25], [86] or solutions representing full (orphaned) blocks [23], [87] in the computation of a chain quality. These partial or orphaned solutions can be incentivized by rewards to further improve decentralization. Another option for a deterministic fork-choice rule is using a pseudo-random function [88], which moreover provides unpredictability,[5] hence an attacker cannot determine his chances to win a tie. Finally, PoW protocols can be combined with BFT protocols, where PoW is used only for joining the protocol and BFT for consensus itself (e.g., [34], [35], [36], [68], [88]).

Feather Forking: In this attack [89], the adversary creates incentives for rational miners to collectively censor certain transactions. Before a mining round begins, an adversary announces that he will not extend the block containing blacklisted transactions, and thus will attempt to extend a forked chain. Although this strategy is not profitable for the adversary and the success rate is dependent on his consensus power, rational nodes prefer to join the censorship to avoid the potential loss. *Countermeasures:* design-oriented protection is to minimize the chance of the attacker being successful, which can be done by including (and rewarding) partial solutions [18], [25], [86], [90] or full orphaned blocks [23], [87] into branch difficulty computation.

Bribery Attacks: Whereas feather forking involves adversaries who try to influence the behavior of miners by threatening to hurt their profits, bribery attacks involve the offering of direct rewards to miners. For example, consensus nodes could be bribed to enable double-spending attacks [91] or to reorder transactions within a block, and thus enable the transaction front-running by other means than natural priority gas auctions (PGAs) [92].[6]

Countermeasures: assuming that the miners who accept bribes constitute a minority, a possible mitigation technique is to utilize partial solutions [18], [25], [86], [90], which reduce the likelihood that the double-spending attacks succeed. Regarding “bribed” transaction front-running, the misbehavior happens entirely off-chain

[5] As opposed to uniform-tie breaking that provides only unpredictability, this solution additionally brings determinism.

[6] In PGAs, users (arbitrage bots) compete with each other to be the first to interact with a smart contract (e.g., due to profit from intra-chain exchanges – see Section VIII-B)

since miners have complete control over the transaction ordering process, so on-chain mitigation is challenging. Moreover, the likelihood of this attack is directly proportional to the consensus power of the bribed miner; hence, this attack is more feasible for mining pools. However, since no evidence confirming collusion between mining pools and bots has been found yet, mining pools are likely to be discouraged from accepting bribes due to the fear of consequences (e.g., a decrease in the market value of the crypto-tokens after these attacks are publicly disclosed).

Time-Spoofing Attacks: Time-spoofing attacks target a time-based difficulty computation algorithm in a PoR protocol with the intention to decrease the difficulty of the puzzle and thus minimize the effort for obtaining the same reward. In particular, the attacker is a consensus node that mines blocks with delayed timestamps, which indicates that a puzzle is too hard to meet block creation rate, and therefore difficulty needs to be decreased. *Countermeasures:* A solution that improves the accuracy of the timestamps may utilize partial solutions found by all nodes into an averaged timestamp computation [18]. Note that the impact of time spoofing attack might be significant also at the application layer, especially in use cases that rely on timestamp accuracy (see Section VIII-F).

Pool Specific Attacks: Since PoR protocols are usually based on a lottery having a single winner [17], rewards for participation impose a high payout variance for solo miners (i.e., once in a few years). As a consequence, mining pools emerged and caused centralization of the mining power, which may result in selfish mining, double-spending, or 51% attacks. *Countermeasures:* Non-outsourceable scratch-off puzzles [60] avoid the creation of pools but require each consensus node to meet high demands on connectivity and storage, as opposed to centralized pools, where only a pool operator needs to meet these demands. If pools are acceptable, their size can be controlled by protocols that reward partial solutions [18], [25], [86], [90] and thus minimize payout variance. For a detailed analysis of rewarding schemes in pools, we refer the reader to [93]. In the following, we describe several types of pool-specific attacks.

a) Pool hopping: The individual contribution of miners in a pool is proved by broadcasting partial solutions, called *shares*. If pay-per-share (PPS) rewarding is employed (i.e., pool operator instantly rewards miners showing shares), an attacker may jump into another pool after his mining time in a victim pool reaches a certain threshold [94] since mining at the early stages of a round is statistically more profitable than mining at the end of the round. As a countermeasure pay-per-last-N-shares (PPLNS) scheme and its variants [95] can be used. PPLNS removes the concept of rounds and instead of immediate payments, it employs deterred payments after N shares are submitted by a miner.

b) Block withholding: An attacker may try to sabotage a victim pool – after mining a block in a victim pool, the attacker discards this block and continues mining at another pool [93]. Such withholding does not mean a direct gain for the attacker, but she may do a secret agreement with concurrent pool(s) that may reward the attacker for showing a withheld block [96] (a.k.a., sponsored block withholding). Mitigation for this kind of attack is using the PPLNS scheme, giving an extra reward to the miner of the block [97], precluding miners from distinguishing between a share and a full solution (i.e., oblivious tasks).

c) Lie-in-wait: If the miner finds a block in a victim pool, she does not immediately submit it to the pool operator, but instead focuses all her available mining power on the victim pool to increase her relative shares within a pool; after some time attacker releases the formerly found block. A countermeasure for this attack is an oblivious task [96].

d) Selfish mining on a subchain: Decentralized mining pools, such as p2pool [98], achieve decentralization by updating an intermediary coinbase transaction with mined shares. To preserve consistency with the previous versions of the coinbase transaction within a mining round, its history is kept in a subchain. However, a chaining data structure enables selfish mining on a subchain, besides the fact that it implies a high stale rate of shares in a subchain.[7] A possible countermeasure is to use flat data structures for aggregation of shares, such as the Merkle tree or hash of a set [18].

4) Side Effects of Countermeasures: Partial solutions for difficulty computation might cause additional network overheads and thus decrease the throughput of the protocol. However, this is not the case for sufficiently long rounds of PoW protocols, such as in StrongChain [18]. On the other hand, rewarding partial solutions, apart from mitigating some threats, helps to promote decentralization – payout variance is decreased and thus mining pools are not needed in some cases, and in other cases, they can be much smaller.

C. Byzantine Fault Tolerant (BFT) Protocols

BFT protocols represent voting-based [19] consensus protocols that utilize Byzantine agreement and a state machine replication [37]. These protocols assume a fully connected topology, broadcasting messages, and a master-replicas hierarchy. Synchronous examples of this category are PBFT [26], RBFT [27], eventually synchronous examples are BFT-SMaRt [99], Tendermint [28], Byzantine Paxos [100], BChain [30], and asynchronous examples are SINTRA [41] and HoneyBadgerBFT [63]. For more details, we refer the reader to a review of BFT protocols and their practical applications in both permissioned and permissionless blockchains [101].

1) Pros: BFT protocols provide high throughput and fast finality. Another advantage of BFT protocols is that they can be combined with PoS or PoR protocols to achieve reasonable scalability and retain their other properties. This is in line with a lottery approach [19] for selecting a portion of all nodes,

[7] Note that the same applies for Flux [25] and Subchains [90] that maintain a subchain but at the level of the whole network (as opposed to p2pool).

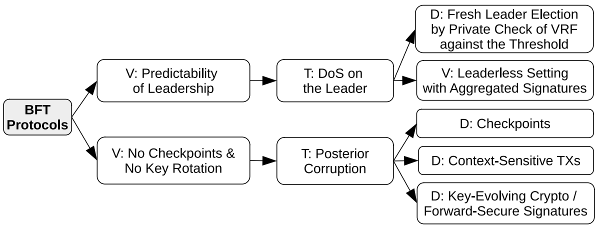

BFT Protocols
V: Predictability of Leadership
T: DoS on the Leader
D: Fresh Leader Election by Private Check of VRF against the Threshold
V: Leaderless Setting with Aggregated Signatures
V: No Checkpoints & No Key Rotation
T: Posterior Corruption
D: Checkpoints
D: Context-Sensitive TXs
D: Key-Evolving Crypto / Forward-Secure Signatures

Fig. 11. Vulnerabilities, threats, and defenses of BFT protocols (consensus layer).

referred to as a committee, which further runs BFT consensus or its part (e.g., Algorand [31], Zilliqa [34], DFINITY [33]).

2) Cons: The main con of traditional BFT protocols [26], [100] is low scalability caused by a high communication complexity (i.e., $\Theta(n^2)$). Since these protocols can work efficiently only with a limited number of consensus nodes, they can be used in their pure form only in permissioned blockchains.

3) Improvements: The issues with scalability vs. throughput of BFT protocols can be partially addressed by applying threshold signatures (e.g., HotStuff [102], ByzCoin [88], Theta Blockchain [71]) as well as by optimizing communication patterns from original broadcast to gossiping [71] or communication trees [88]. On top of that, consensus nodes might be partitioned into shards that process transactions in parallel (e.g., Omniledger [35], RapidChain [36]).

4) Security Threats and Mitigations: We present a taxonomy of the attacks related to BFT protocols, their origins, and defenses against them in Figure 11. In the following, we describe these attacks as well as possible defense techniques.

Denial of Service on a Leader: Since BFT protocols are mostly intended for (private) permissioned blockchains that are run by trusted participants, they do not assume the existence of malicious nodes whose goal is to sabotage the protocol. However, assuming such an adversary, a leader of the round might be DoS-ed since her leadership is known before the round starts, which might result in a restart of the round. *Countermeasures:* To prevent this attack, a node can privately determine whether it is a potential leader by using Verifiable Random Function (VRF) [31], and immediately release a block candidate; hence, after publishing this data, it is too late for a DoS attack on the node. Another option for coping with this attack can be implemented by aggregating threshold signatures in a leaderless setting [71].

Posterior Corruption: Posterior corruption is a specific instance of a violation of protocol assumptions (see Section VI-A) in which the adversarial consensus power reaches $\frac{2}{3}$. In posterior corruption, the adversary has to steal private keys of $\frac{2}{3}$ possibly "retired" consensus nodes and then rerun the consensus protocol, rewriting the history of the blockchain. Although this attack is mainly discussed in the context of PoS protocols (see Section VI-D), we note that in PoS the attacker's motivation is typically financial and exploits an incentive scheme of (semi-)permissionless blockchains. On the other hand, the attacker's motivation might be different in BFT protocols in permissioned settings (e.g., governance or sabotage) since many such BFT protocols do not contain an incentive scheme. To mitigate posterior corruption, key-evolving cryptography [103] and forward-secure digital signatures (e.g., d-ary certificate trees [104]) can be employed, requiring users to evolve their private keys and erase already used keys. Another option is to employ irreversible checkpoints after a fixed number of blocks or context-sensitive transactions, which put the hash of a recent valid block into a transaction itself [105].

5) Side Effects of Countermeasures and Improvements: A problem of some threshold signatures (e.g., BLS signatures) is a lack of forward secrecy, which might enable posterior corruption attacks. Forward secrecy might be provided by schemes such as d-ary certificate trees [104]. On the other hand, a disadvantage of some forward-secure digital signatures (e.g., [104]) is an extra overhead required for their verification and updating, which negatively influences the throughput of the blockchains. This problem was addressed in Pixel signatures [106], in which the authors proposed aggregated signatures supporting forward secrecy and demonstrated bandwidth and storage savings in contrast to d-ary certificate trees [104].

Pruning the number of nodes that run BFT into committees [31] reduces the security level of BFT and provides only probabilistic security guarantees depending on the committee size. This phenomenon might be also seen as a deterioration of decentralization.

D. Proof-of-Stake Protocols (PoS)

Similar to the PoR category, PoS protocols are based on the lottery approach [19]. However, in contrast to PoR, no scarce resource is spent; instead, the nodes are required "to prove investment" of crypto-tokens to participate in a protocol, and thus eventually earn interest from the invested amount. The concept of PoS was for the first time proposed in the Bitcointalk forum [107]. The first technical realization of PoS is Peercoin [80], which is a combination with PoW – each node has its particular difficulty for PoW, which is based on the age of the coins a node owns. Although there exist a couple of pure PoS protocols (e.g., Chains of Activity [108], Ouroboros [21]), the trend is to combine them in a hybrid setting with PoR (e.g., Proof-of-Activity [109], Peercoin [80], Snow White [110]) or BFT protocols (e.g., Algorand [31], Theta Blockchain [71]). In particular, a combination of PoS with BFT represents a promising approach, which takes advantage of both lottery and voting (i.e., scalability and throughput), where no resources are wasted.

1) Pros: The main feature of PoS protocols, as compared to PoR, is their energy efficiency. Although some PoS protocols are often combined with a PoR technique (e.g., [80], [110]), the overall energy spent is much smaller than in the case of pure PoR protocols.

2) Cons: The introduction of PoS protocols has brought PoS specific issues and attacks, while these protocols are, at the time of writing, still not formally proven to be secure. Next, PoS protocols are semi-permissionless – a node needs

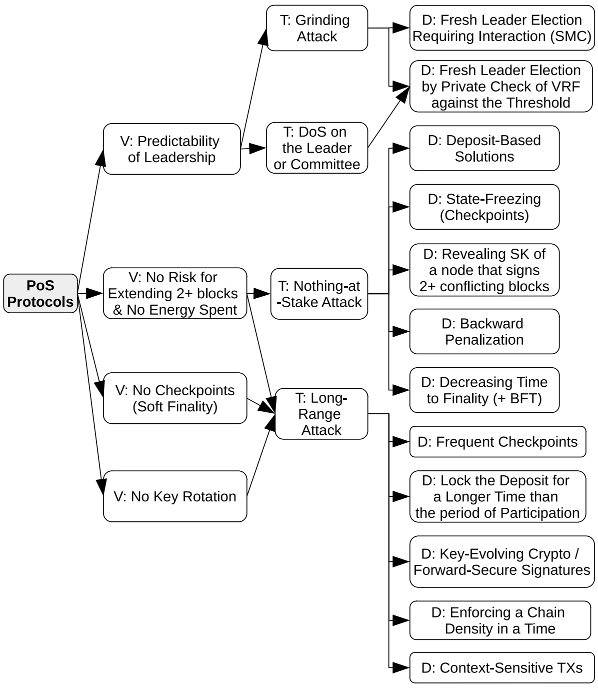

PoS Protocols
V: Predictability of Leadership
V: No Risk for Extending 2+ blocks & No Energy Spent
V: No Checkpoints (Soft Finality)
V: No Key Rotation
T: Grinding Attack
T: DoS on the Leader or Committee
T: Nothing-at -Stake Attack
T: Long-Range Attack
D: Fresh Leader Election Requiring Interaction (SMC)
D: Fresh Leader Election by Private Check of VRF against the Threshold
D: Deposit-Based Solutions
D: State-Freezing (Checkpoints)
D: Revealing SK of a node that signs 2+ conflicting blocks
D: Backward Penalization
D: Decreasing Time to Finality (+ BFT)
D: Frequent Checkpoints
D: Lock the Deposit for a Longer Time than the period of Participation
D: Key-Evolving Crypto / Forward-Secure Signatures
D: Enforcing a Chain Density in a Time
D: Context-Sensitive TXs

Fig. 12. Vulnerabilities, threats, and defenses of PoS protocols (consensus layer).

to first obtain a stake from any of the existing nodes to join the protocol.

3) Security Threats and Mitigations: We present a taxonomy of the attacks related to PoS protocols, their origins, and defenses in Figure 12. In the following, we describe these attacks as well as possible defense techniques.

Nothing-at-Stake: Since generating a block in PoS does not cost any energy, a node can extend two or more conflicting blocks without risking its stake, and hence increase its chance to be rewarded. Such behavior increases the number of forks and thus time to finality. *Countermeasures:* Deposit-based solutions (e.g., [61]) require nodes to make a deposit during some fixed period/round and checkpoint-based solutions (e.g., [32], [61], [80], [111]) employ "state freezing" at periodic snapshots of the blockchain, while the blockchain can be reversed maximally up to the recent checkpoint. Another option is to punish a node that signs two conflicting blocks by embedding cryptographic solutions [112] that enable anybody to reveal the identity and a private key of such a node. Another countermeasure is to use backward penalization of nodes that produced two or more conflicting chains [32], [61]. Finally, PoS protocols can be combined with BFT approaches, and thus the probability of forks is negligible (e.g., [31]).

Grinding Attack: If the leader or committee producing a block is determined before the round starts, then the attacker can bias this process to increase her chances of being selected in the future. For example, if a PoS protocol takes only a hash of the previous block for the election process, the leader of a block may bias a hash value by suitably adjusting the content of the block in a few attempts. *Countermeasures:* The grinding attack can be prevented by performing a fresh leader election by an interaction of consensus nodes within some committee (e.g., the secure multiparty coin-flipping protocol [21]) or by privately checking whether the VRF output is below a certain stake-specific threshold (e.g., [31]). The input of the VRF is the user's private key and the randomness unambiguously bound to the previous block; hence each consensus node might compute the only VRF output during each round.

Denial of Service on a Leader/Committee: Alike in BFT protocols (see Section VI-C), if a leader or a committee is publicly determined before the round starts [21], then the adversary may conduct a DoS attack against them and thus cause a restart of the round – this might be repeated until the adversary's desired nodes are elected. *Countermeasures:* A prevention technique was proposed in Algorand [31] – a node privately determines whether it is a potential leader (or committee member), and immediately releases a block candidate (or a vote) – hence, after publishing this data, it is too late for a DoS attack. The concept of the VRF was also utilized in other protocols (e.g., [33], [113]).

Long-Range Attack: In this attack [114] (a.k.a., posterior corruption [32]), an adversary can "bribe" previously influential consensus nodes to sell their private keys or steal the private keys by other means. Since consensus nodes may exchange their crypto-tokens for fiat money anytime, selling their keys imposes no expenses and risk. If the attacker accumulates keys with enough stake in the past, he may rerun the consensus protocol and rewrite the history of the blockchain. A variant of long-range attack that considers only transaction-fee-based rewarding and infrequent or no check-points is denoted as a *stake-bleeding* attack [105]. *Countermeasures:* One mitigation is to lock the deposit for a longer time than the period of participation in the consensus [115]. The next mitigation technique is frequent periodic checkpointing, which causes the irreversibility of the blockchain with respect to the last checkpoint. Another option is to apply key-evolving cryptography [103] and forward-secure digital signatures [104], which require users to evolve their private keys, while already used keys are erased [113]. Hence, signatures cannot be forged in the case of compromise. The third mitigation technique is enforcing a chain density in a time-domain [105] for the protocols where the expected number of participants in each round is known (e.g., [21]). The last mitigation technique is context-sensitive transactions, which put the hash of a recent valid block into a transaction itself [105].

4) Side Effects of Countermeasures: Some countermeasures for threats in PoS protocols might impact the features of the blockchains. We note that secure multiparty coin-flipping protocol brings requirements on additional interactions among the consensus nodes, and thus it deteriorates the throughput of the protocol. On the other hand, the throughput does not deteriorate when the leader and the committee are elected non-interactively by VRF.

VII. Replicated State Machine Layer

The Replicated State Machine (RSM) layer is responsible for the interpretation and execution of transactions that are already ordered by the consensus layer. Concerning security threats for this layer are related to the privacy of users, privacy and confidentiality of data, and smart contract-specific bugs. We split the security threats of the RSM layer into two parts: standard transactions and smart contracts.

A. Transaction Protection

Transactions containing plain-text data are digitally signed by private keys of users, enabling anybody to verify the validity of transactions with the corresponding public keys. However, such an approach provides only pseudonymous identities that can be traced to real IP addresses (and sometimes to identities) by a network-eavesdropping adversary, and moreover, it does not ensure the confidentiality of data [116]. Therefore, several blockchain-embedded mechanisms for the privacy of data and user identities were proposed in the literature, which we further elaborate on. Note that some privacy-preserving techniques can be applied also on the application layer of our stacked model but imposing higher programming overheads and costs (e.g., see Section IX-A, Section IX-F, and Section IX-G). This is common in the case of blockchain platforms that do not support them natively.

1) Security Threats and Countermeasures: We present a taxonomy of vulnerabilities, threats, and defenses related to the privacy of transaction data and user identities in Figure 13.

Privacy Threats to User Identity: In many blockchains, user identities can be linked with their transactions by various deanonymization techniques, such as network flow analysis, address clustering, or transaction fingerprinting [116], [117], [118]. Moreover, blockchains designed with anonymity and privacy features (e.g., Zcash, Monero) are also vulnerable to a few attack strategies [119], [120]. *Countermeasures:* Various means are used for obfuscating user identities, including centralized [121], [122] and decentralized [123], [124], [125] mixing services, ring signatures [126], and non-interactive zero-knowledge proofs (NIZKs) [127], [128]. Some mixers enable internal linkability by involved parties [123] or linkability by the mixers [121], which are also potential threats. Unlinkability for all parties can be achieved by multiparty computation (MPC) [125], blinding signatures [122], or layered encryption [124]. Ring signatures provide unlinkability to users in a signing group [126], enabling only the verification of correctness of a signature, without revealing an identity of a signer.

Privacy of data: Blind signatures [129] and NIZKs such as zk-SNARKs [128] or (shorter) Bulletproofs [130] can be used for the preservation of data privacy. Another method is homomorphic encryption, which enables the computation of certain operations over encrypted messages (e.g., ElGamal encryption provides additive homomorphism). Privacy and confidentiality for smart contract platforms can be achieved through trusted transaction managers [131] utilizing zk-SNARKs, trusted hardware [132], and secure multiparty computations [133] embedded into these platforms. Privacy of data can be achieved even on blockchain platforms without embedded support of privacy-preserving constructs. For example, Zether [134] is built on top of the public smart contract platform Ethereum, and it provides a confidential payment mechanism that embeds the balance of users into (secret) exponents of ElGamal encryption. Other similar examples that deal with the privacy of data at the application layer of our stacked model are presented in Section IX-A, Section IX-G, Section IX-F.

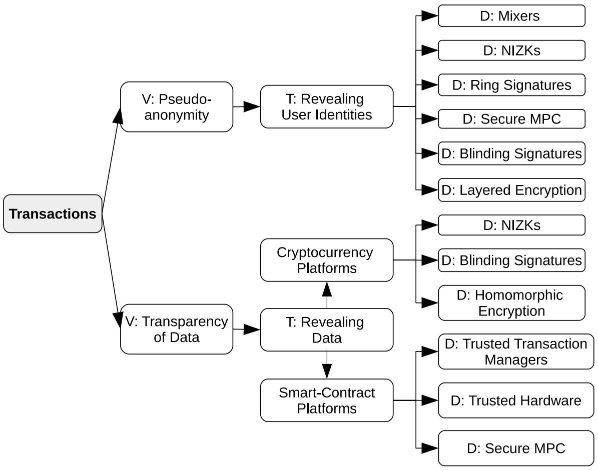


Fig. 13. Vulnerabilities, threats, and defenses of privacy threats (RSM layer).

2) Side Effects of Countermeasures: Since protocols of mixing services usually contain a few rounds (that may involve the creation of several transactions), all mixing services slow down the transaction throughput. The next blockchain feature that is influenced by some mixers is decentralization. As a consequence, centralized mixing services may misbehave and reveal linkable information of transactions, they can be DoS-ed, or they can steal the funds. Accountability for the misbehavior of centralized mixing service is provided in MixCoin [121] and Blindcoin [122] while the latter additionally provides internal unlinkability by a mixing service. In contrast to centralized mixers, decentralized mixers remove a trusted third party (i.e., no theft is possible) and provide stronger guarantees for unlinkability of transactions, e.g., CoinShuffle [124] and CoinParty [125] require at least two and $\frac{2}{3}m$ honest participants, respectively, to provide full unlinkability. Decentralization and availability are also impacted in solutions that utilize trusted hardware [131], [132].

The throughput of blockchains is also impacted in cryptographic countermeasures such as NIZKs, ring signatures, and blinding signatures. Ring signatures cause the large transaction size, which is linear with the number of participants in the anonymity set. The size of the ring signatures was optimized by cryptographic accumulators in [135], which in turn enabled the improvement of the throughput. NIZKs utilized in ZeroCoin [127] produce large proofs as well. The proof size (and thus throughput) was further optimized by zk-SNARKs in ZeroCash [128]. However, the disadvantage of zk-SNARKs is the requirement for a trusted setup. This requirement is eliminated in Bulletproofs [130], which further decrease the size of proofs in transactions and thus improves on throughput.

B. Smart Contracts

Smart contracts introduced to automate legal contracts, now serve as a method for building decentralized applications on blockchains. They are usually written in a blockchain-specific programming language that may be Turing-complete (i.e., contain arbitrary programming logic) or only serve for limited purposes. In the following, we describe these two contrasting types of smart contract languages and their security aspects.

1) Security Threats and Countermeasures: We present a taxonomy of vulnerabilities, threats, and defenses inherent to smart contract platforms in Figure 14.

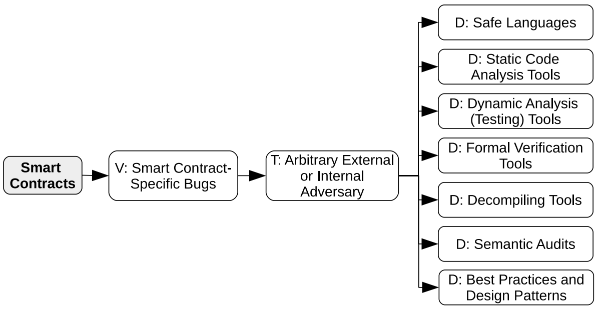

Smart Contracts
V: Smart Contract-Specific Bugs
T: Arbitrary External or Internal Adversary
D: Safe Languages
D: Static Code Analysis Tools
D: Dynamic Analysis (Testing) Tools
D: Formal Verification Tools
D: Decompiling Tools
D: Semantic Audits
D: Best Practices and Design Patterns

Fig. 14. Vulnerabilities, threats, and defenses of smart contract platforms (RSM layer).

Turing-Complete Languages: An important aspect of this smart contract language category is a large attack surface due to the possibility of arbitrary programming logic. Examples of this category are Serpent, Vyper, Yul, Flint, LLL, and Solidity, while as of now Solidity is the most popular and widely-used one. *Serpent*[8] is a high-level language that was designed to be simple and similar to the Python language. However, Serpent was designed in an untyped fashion, lacking out-of-bound access checks of arrays and accepting invalid code by compilers [136], which opened the door for plenty of vulnerabilities. Hence, Serpent showed to be an unsuccessful attempt to simplify the coding phase. *Vyper*[9] is an experimental language designed to ease the audit of smart contracts and increase security – it contains strong typing, bounds checks, and overflows. *Yul*[10] is a typed intermediate language for Ethereum, which can be compiled to bytecode for the EVM 1.0, EVM 1.5 and eWASM platforms. Snippets of Yul code can be inserted as an inline assembly within Solidity code to perform optimizations that are applicable for these three platforms. *Flint* [137] is a type-safe language for Ethereum smart contracts. The major focus of this language is its robustness, and it also provides some special features such as caller protection, which can help to produce robust contracts. *Lisp Like Language (LLL)*[11] is a low-level language that is similar to Assembler. It aims to be simple and to support the creation of clean code; for example, it removes the need to code the stack and jump management. Moreover, it enables a focus on the resource-constrained nature of Ethereum and allows optimized use of the resources. *Solidity*[12] is an object-oriented statically-typed language that is primarily used by the Ethereum platform. Contracts written in Solidity can contain various types of vulnerabilities [138], [139], [140], which resulted in many incidents in the past. Table IV of Appendix outlines the most prominent incidents and the associated vulnerabilities. In addition, the table classifies vulnerabilities according to an existing smart contract weakness classification (SWC) registry [141].[13]

Countermeasures: Mitigation techniques for such vulnerabilities are static or dynamic analysis (testing) tools, formal verification tools, security audits, as well as respecting best practices and using known design patterns [141], [142]. The literature contains various smart contract analysis tools for the detection of vulnerabilities [143], [144]. In the following, we give an overview of them:

- *Static analysis tools* such as linters, try to find vulnerabilities by inspecting the source code. For example, SmartCheck [145], Solhint,[14] Solium [146], and Slither[15] belong to this category. Another example, sCompile [147], works statically, but it also includes a dynamic component.
- *Dynamic analysis tools* seek vulnerabilities while executing the code of smart contracts. For example, simple forms of dynamic analysis are unit testing with handcrafted tests or replay testing [148], where existing executions (or manually captured ones) are used to check if the same results can be reproduced. A more automated form of testing is fuzzing [149], which generates unexpected, undefined, random, or invalid inputs to trigger a crash or reveal defects and vulnerabilities. Fuzzers, like ContractFuzzer [150], Echidna,[16] and Harvey [151] can be used for automated smart contract testing as well. Another technique of dynamic analysis is symbolic execution [152], where a program is executed with symbolic values (i.e., logical expressions) that make it possible to explore all reachable paths of a program. For this technique, there are tools

[8] https://github.com/ethereum/wiki/wiki/Serpent-%5BDEPRECATED%5D

[9] https://vyper.readthedocs.io/en/v0.1.0-beta.9/

[10] https://solidity.readthedocs.io/en/v0.5.10/yul.html

[11] https://lll-docs.readthedocs.io/

[12] https://solidity.readthedocs.io/

[13] Note that for some vulnerabilities there are no publicly available references on incidents supporting the existence of vulnerabilities.

[14] https://github.com/protofire/solhint

[15] https://github.com/crytic/slither

[16] https://github.com/crytic/echidna/

like Securify [153], Manticore [154], Oyente [155], and Osiris [156].

- *Formal verification tools* usually apply an abstract model or a semantic definition to check for the security and/or correctness properties of smart contracts. One type of this category is represented by semantic-based approaches that work with a semantic language specification defining the expected behavior of smart contracts. Examples of this type are FSolidM [157], and Kevm [158]. Other types of formal verification tools are semantic-based approaches that work with a behavioral model [159]. Several formal verification methods [160], [161], [162] apply abstract models (e.g., finite state machines) that define the expected states and outputs of smart contracts for a given input. The underlying models are used for checking the functional correctness or the presence of vulnerabilities (e.g., Zeus [163]). Additionally, such models can be applied to proving certain security properties [164].
- *Decompiling tools.* The source code of contracts is often not public in contrast to their bytecode. For this reason, bytecode decompilers, like Erays [165], Eveem,[17] or Porosity [166] can be used to (partially) reconstruct the source code of a contract. Additionally, there exist various static bytecode analyzers, like Maian [167], MadMax [168], Vandal [169], and automated exploit generators, like Teether [170] that can be utilized to find vulnerabilities in the bytecode.

Turing-Incomplete Languages: The main pro of this category is its design-oriented goal of a small attack surface and the emphasis on safety, which is achieved at the cost of limited expressiveness. Examples of this category are Pact, Scilla, Bitcoin Script, Ivy, and Simplicity. *Pact* [171] is a declarative language intended for the Kadena blockchain and provides type inference and module-guarded tables to prevent direct access to the module. Pact is equipped with the ability to express and check properties of its programs, also leveraging satisfiability modulo theories (SMT) solvers. *Scilla* [172] is designed to achieve expressiveness and tractability while enabling formal reasoning about contract behavior. Every computation utilizes an automata-based model, and computations are realized as standalone atomic transitions that strictly terminate. Scilla enables external calls only in the last instruction of a contract, which simplifies proving safety and thus mitigates a few vulnerabilities. *Bitcoin Script* [173] is a stack-based language for the Bitcoin platform. It has limited complexity and processing requirements, and its main purpose is transaction processing. *Ivy* is a high-level declarative predicate language for the Bitcoin platform. It can be compiled to a Bitcoin script and its main advantage is its comprehensibility, which enables fast writing and an easy understanding of the code. *Simplicity* [174] is a typed functional language that works with combinators. It is equipped with (formal) denotational and operational semantics, which facilitate the estimation of the required computing resources.

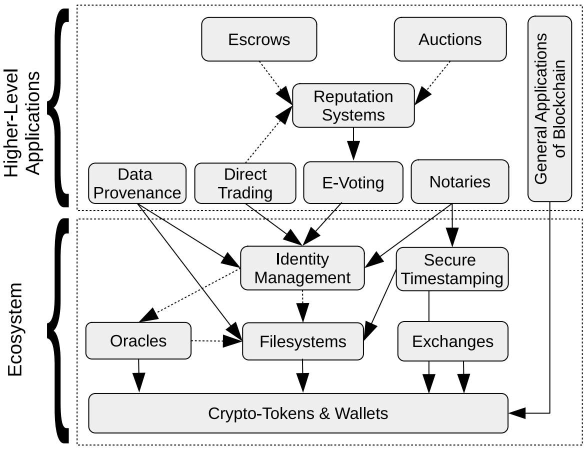


Fig. 15. Hierarchy in inheritance of security aspects across categories of the application layer. Dotted arrows represent application-specific and optional dependencies.

2) Side Effects of Countermeasures: Since all of the smart contract related countermeasures are performed before the deployment of smart contracts as part of the development stage of the blockchain-based applications, they do not negatively impact any blockchain features. Note that only countermeasures related to the operational stage of blockchain-based applications might influence blockchain features.

VIII. Application Layer: Ecosystem Applications

We present a functionality-oriented categorization of the applications running on or utilizing the blockchain in Figure 15, where we depict hierarchy in the inheritance of security aspects among particular categories. In this categorization, we divide the applications into categories according to the main functionality/goal that is to be achieved by using the blockchain. Security threats of this layer are mostly specific to particular types of applications. Nevertheless, there are a few application-level categories that are often utilized by other higher-level applications. In the current Section, we isolate such categories into a dedicated application-level group denoted as an *ecosystem*, while we describe the rest of the applications in Section IX. The group of ecosystem applications contains five categories: (1) **crypto-tokens and wallets**, (2) **exchanges**, (3) **oracles**, (4) **filesystems**, (5) **identity management**, and (6) **secure-timestamping**. We accompany the application layer with several incidents in Table III of Appendix.

A. Crypto-Tokens & Wallets

Besides blockchains that provide cryptocurrencies with native crypto-tokens, there are blockchain applications that use crypto-tokens to provide owners with rights against a third party (i.e., counter-party tokens) or with the possibility of transferring asset ownership (i.e., ownership/colored tokens) [175]. All types of tokens require the protection

[17] https://eveem.org/

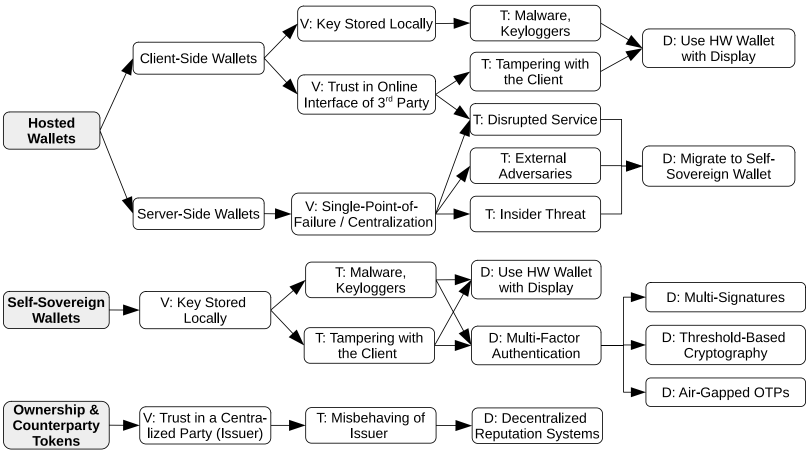

Hosted Wallets
Client-Side Wallets
Server-Side Wallets
V: Key Stored Locally
V: Trust in Online Interface of 3rd Party
V: Single-Point-of-Failure / Centralization
T: Malware, Keyloggers
T: Tampering with the Client
T: Disrupted Service
T: External Adversaries
T: Insider Threat
D: Use HW Wallet with Display
D: Migrate to Self-Sovereign Wallet
Self-Sovereign Wallets
V: Key Stored Locally
T: Malware, Keyloggers
T: Tampering with the Client
D: Use HW Wallet with Display
D: Multi-Factor Authentication
D: Multi-Signatures
D: Threshold-Based Cryptography
D: Air-Gapped OTPs
Ownership & Counterparty Tokens
V: Trust in a Centralized Party (Issuer)
T: Misbehaving of Issuer
D: Decentralized Reputation Systems

Fig. 16. Vulnerabilities, threats, and defenses of the crypto-token & wallets category (application layer).

of private keys and secrets linked with user accounts. For this purpose, two main categories of wallets have emerged: *self-sovereign wallets* (a.k.a., non-custodial) and *hosted wallets* [176], [177]. All crypto-tokens are exposed to technical and regulatory risks, while non-native tokens are also exposed to legal risks [175].

Self-Sovereign Wallets: Users of self-sovereign wallets locally store their private keys and directly interact with the blockchain platform using these keys, while they verify the inclusion of their transactions by SPV client software. The instances of these wallets differ in several points. One of them is the manner in which the keys are isolated – there are software wallets that store the keys within the user PC (e.g., Bitcoin Core,[18] MyEtherWallet[19]) as well as hardware wallets that store keys in sealed storage, while they expose only signing functionality (e.g., Trezor,[20] Ledger[21]). The next type of wallets enables functionality and security customization through a smart contract (e.g., TrezorMultisig2of3,[22] Ethereum MultiSigWallet,[23] SmartOTPs [178]).

Hosted Wallets: Hosted wallets require a centralized party, which provides an interface for interaction with the wallet and the blockchain. If a hosted wallet has full control over private keys, it is referred to as a *server-side wallet* (e.g., Coinbase[24]) while in the case when keys are stored in the user's browser, a wallet is referred to as a *client-side wallet* (e.g., Blockchain Wallet[25]). We refer the reader to works [176], [178] for a security overview of miscellaneous wallet solutions.

1) Security Threats and Mitigations: We present a taxonomy of vulnerabilities, threats, and defenses related to the crypto-token wallets category in Figure 16. Server-side wallets pose a single-point-of-failure, which can be exploited by external or internal adversaries, and moreover, it can be subjected to availability attacks such as DoS. Since server-side wallets have been the target in several security incidents [179], [180], [181], their popularity has declined in favor of client-side wallets. Client-side wallets do not expose private keys to a centralized party but store it locally. Nevertheless, they still trust in the online interface provided by such a party, and thus their availability is dependent on this party. Other threats with client-side wallets are client tampering and malware/keyloggers, which focus on deceiving the user while signing a transaction or stealing the key. Possible mitigations of these attacks include hardware wallets that display details of transactions to the user, while the user confirms signing by a button (e.g., Trezor, Ledger).

In contrast, self-sovereign wallets do not trust in a third party nor rely on its availability. However, these wallets are susceptible to key theft (i.e., malware [182], keyloggers [177], [183]). One protection is to use a hardware wallet with a display as described above. Another option is to protect self-sovereign wallets by multi-factor/(-step) authentication using multi-signatures (e.g., TrezorMultisig2of3, Ethereum MultiSigWallet), threshold-based cryptography [184], or air-gapped OTPs [178]. In the case of counter-party and ownership tokens presented at the application layer of existing public blockchains, we emphasize the additional vulnerability caused by trusting in the centralized party that issues such tokens – the provided counter-party and ownership rights are only virtual, which imposes a significant risk. While this risk cannot be eliminated in this application scenario, a possible mitigation technique for preventing fraudulent issuers is to use decentralized reputation-based systems (see Section IX-B) and notaries (see Section IX-D) that might be built on top of them.

2) Side Effects and Implications of Countermeasures: A disadvantage of some multi-factor authentication solutions is that they require the execution of smart contracts, which increases the costs and might slow down the throughput of the system. In contrast, threshold-based cryptography constructs save these costs since they produce only a single signature,

[18] https://bitcoin.org/en/download
[19] https://www.myetherwallet.com/
[20] https://trezor.io/
[21] https://www.ledger.com/products/ledger-nano-s
[22] https://github.com/unchained-capital/ethereum-multisig
[23] https://github.com/ConsenSys/MultiSigWallet
[24] https://www.coinbase.com/
[25] https://blockchain.info/wallet/

which appears as it was made by a single party. However, threshold-based cryptography requires off-chain computation, in which the duration of execution is dependent on the number of co-signing parties.

Although self-sovereign wallets provide higher security in contrast to client-side and server-side hosted wallets, they impose overheads for storing the non-negligible part of the blockchain (i.e., headers) to validate the inclusion of signed transactions within their SPV client software – this is especially a concern when users have multiple SPV clients for multiple blockchains. BTC Relay is an early optimization attempt that reduces storage requirement by outsourcing the validation of transactions from the source blockchain to the target blockchain in exchange for a small fee paid to the relay nodes that submit headers to the target blockchain. However, the costs of BTC Relay were too high and its users tend to rather store headers of source blockchain on their own. Zk-relay [185] optimizes these costs by using zk-SNARKs with batched block validation, achieving an improvement by a factor of 187.

B. Exchanges

If the user wishes to exchange crypto-tokens, she might either directly find a counter-party wishing to exchange the opposite pair or approach an exchange that might be centralized or decentralized (DEX). In the case of centralized exchanges, the security threats and implications are due to centralization, and the only countermeasure is to use decentralized exchange solutions that we further focus on.

Direct Cross-Chain Exchange With Atomic Swap: Atomic swaps[26] assume two parties owning crypto-tokens in two different blockchains, and these parties wish to execute exchange atomically, i.e., either both of the parties receive the agreed amount or neither of them. The atomic swap protocol enables conditional redemption of the funds in the first blockchain upon revealing the hash pre-image (i.e., secret) that redeems the funds on the second blockchain. This protocol is based on two Hashed Time-Lock Contracts (HTLC) that are deployed by both parties in both blockchains, and it requires 4 transactions (see details in Appendix B).

Cross-chain DEX: Although atomic swaps are, in theory, sufficient means for the execution of fair cross-chain exchange, the situation is more complicated in practice. In particular, there might not exist a contra-party exchanging the opposite pair or the user might not be aware of it. This motivates DEXes, which facilitate the process of maintaining and matching the existing orders, act as a contra-party or intermediary, while guaranteeing fairness (e.g., Komodo[27]). The users match the orders, reward DEX, and afterward perform an atomic swap on their own. If users wish to trade more obscure crypto-tokens, for which there is no matching counter-order, DEX may serve as a counter-party and do the atomic swap with the user. Moreover, if the users wish to trade colored tokens for native crypto-tokens of different blockchains (e.g., $\mathbb{A}$ sells an asset for BTC and $\mathbb{B}$ buys it for ETH), DEX might serve as an intermediary who executes the three-way atomic swap [186] (see details in Appendix B).

Intra-Chain DEX: Some intra-chain DEX designs (e.g., Maker Market, EtherOpt, and Intrinsically Tradeable Tokens) require parties to post buy&sell offers on the blockchain, while smart contracts perform matches and execute trades. However, each placing of an order or its modification requires a payment for the inclusion of a transaction. Therefore, designs with off-chain order matching became more popular; in these designs, only trades are executed on-chain, while orders and their matching is performed off-chain. An example is 0x [187] protocol handling DEX of ERC20 tokens (e.g., applied in EtherDelta[28]). The next intra-chain exchange design is known as the *automated market maker* (AAM) [188]. AAM is applicable within a smart contract-based DEX that contains deposited reserves of traded ERC20 tokens; examples are Euler [189], Bancor [190], and Uniswap.[29] AAM provides high liquidity since users are not required to match their orders, and they can directly do the exchange with the smart contract.

Cross-Chain Communication: The concept of cross-chain exchange can be further generalized into cross-chain communication (CCC), which deals with the interoperability of applications running on different blockchains. The security aspects of CCC are very similar to the exchanges, and we refer the interested reader to the work of Zamyatin *et al.* [191].

1) Security Threats and Mitigations: We present an overview of vulnerabilities, threats, and defenses related to the exchanges category in Figure 17.

In **centralized exchanges**, threats are caused by external and internal adversaries, and they are identical to those of server-side hosted wallets (see Section VIII-A) since server-side wallets always provide exchange services. So far centralized exchanges posed the most attractive target for adversaries that have caused huge financial losses [192]. There are many operation security (OPSEC) countermeasures such as multi-factor authentication, split of the funds to hot and cold wallets, however, none of them eliminates the single-point-of-failure coming from the centralization. Therefore, effective mitigation from the user point of view is to use decentralized exchange solutions such as DEXes and atomic swaps; however, they also contain some vulnerabilities.

Different blockchains of **cross-chain** decentralized (direct and DEX-based) exchanges might have a different time to finality, and thus the likelihood that one blockchain will be overturned is higher than in the case of the other one. Therefore, the number of required confirmations might be agreed upon by involved parties beforehand. However, this results in longer delays for the execution of the protocol and the need for both parties to be online. In some cases, such long delays might cause fluctuation in the exchange rate, making the exchange not attractive at a later time. As a mitigation technique, off-chain exchanges (within side-chains) might be used, where each blockchain is updated only with the final transaction. Off-chain real-time exchanges might also be achieved

[26] https://en.bitcoinwiki.org/wiki/Atomic_Swap

[27] https://komodoplatform.com/atomic-swap-technology/

[28] https://etherdelta.com/

[29] https://uniswap.io/

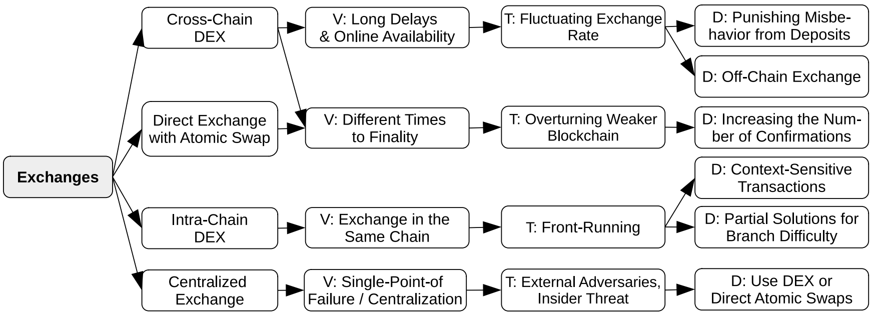

Exchanges
Cross-Chain DEX
Direct Exchange with Atomic Swap
Intra-Chain DEX
Centralized Exchange
V: Long Delays & Online Availability
V: Different Times to Finality
V: Exchange in the Same Chain
V: Single-Point-of Failure / Centralization
T: Fluctuating Exchange Rate
T: Overturning Weaker Blockchain
T: Front-Running
T: External Adversaries, Insider Threat
D: Punishing Misbe-havior from Deposits
D: Off-Chain Exchange
D: Increasing the Num-ber of Confirmations
D: Context-Sensitive Transactions
D: Partial Solutions for Branch Difficulty
D: Use DEX or Direct Atomic Swaps

Fig. 17. Vulnerabilities, threats, and defenses of the exchanges category (application layer).

with the use of TEE [193] (see below). Another mitigation for intentional delaying of the exchange by any party is to use deposit-based bonds, which will be restored only when a particular party acts timely.

Since **intra-chain** exchanges are executed in the single blockchain, they give rise to the transaction front-running, in which the adversary (a.k.a, arbitrage bots) front-run user trades by transactions containing higher fees and by optimizing network latency [92]. Moreover, such adversaries might compete with each other by "bidding" a higher transaction fee [92], which in turn targets the ordering mechanism of the consensus layer, where miners might tend to overturn or fork the blockchain while including only transactions with the highest fees. A mitigation technique to these threats is represented by context-sensitive transactions [105], which do not allow overturning of the blockchain, only its extension. The same effect can be achieved by partial solutions included in branch difficulty computation [18], [25], [86], [90]. Note that context-sensitive transactions and partial solutions are a means of the consensus layer.

2) Side Effects and Implications of Countermeasures: We assume centralized exchanges as the baseline that requires only two transactions for settlement of cross-chain exchange or one transaction for intra-chain exchange. Note that such settlement is done only sporadically since centralized exchanges inherently work off-chain in real-time. In contrast, direct atomic swaps require 4 transactions while three-way atomic swaps require 6 transactions. Therefore, using these constructs negatively affects the throughput of blockchains. Nevertheless, the performance can be improved by off-chain execution of atomic swaps, which provides almost immediate response, e.g., off-chain swaps are possible in two-parties payment chanels as well as their extension to multiple parties known as the lightning network [194]. Such off-chain solutions greatly improve the scalability, since parties can transact directly and involve blockchain consensus only when they wish to settle their balances (e.g., once per day or week). However, to avoid misbehavior in which a "stale" balance is settled on-chain, these systems require that the parties constantly monitor the blockchain state. Such an always-online assumption can be relaxed by employing watching services [195], [196], [197] (a.k.a., watchtowers), which, however, incur extra costs.

TEE can be also utilized as an off-chain means that improves the throughput of exchange service. For example, Tesseract [193] is a real-time exchange that leverages TEE for communication with users and a target blockchain. To enter the system, users submit time-locked refill transactions paying to Tesseract's controlled address in the target blockchain, and then Tesseract uses its SPV client to verify their inclusion. Existing users submit bid&sell requests to Tesseract, which performs matching and executes trades within TEE. When users want to sync with the on-chain state, they ask Tesseract to generate settlement transactions. Since decentralization would be impacted by using a single service of Tesseract, which might censor user requests, the authors incorporate the Paxos consensus protocol among multiple mutually untrusted Tesseract nodes; this also increases the fault-tolerance and avoid funds of users becoming stuck in contrast to one instance of Tesseract. We note that censorship evidence (not resistance) could be alternatively ensured in Tesseract by smart contract-based censorship resolution [198], which, however, implies some extra costs for smart contract execution. Finally, TEE-based solutions might be vulnerable to attacks on trusted hardware. As a mitigation technique to reliance on a trusted manufacturer of TEE, it is possible to use a quorum of several redundant TEEs from multiple manufacturers. Furthermore, it is important to note that the assumption about the code executed in TEE is its bug-freeness, and thus one might not use return-oriented programming or other techniques to ex-filtrate sealed secrets or private parameters; this is an out-of-the-scope attacker model for TEE.

C. Oracles

Oracles (a.k.a., *data feeds*) are trusted entities that provide plausible data that reflects the state of the world beyond the blockchain. The authors of [199] and [200] define a few security properties of oracles in smart contract platforms:

Authenticity: Data are authentic if they are produced by content providers agreed by the consumers of the data.

Integrity: Provided data should not be modified nor deleted after creation. Therefore, content providers should guarantee the correctness of the newly created data and publicly prove their consistency with the past.

Confidentiality: Sometimes, input parameters may contain confidential or private data. Therefore, an oracle should support such parameters and their handling.

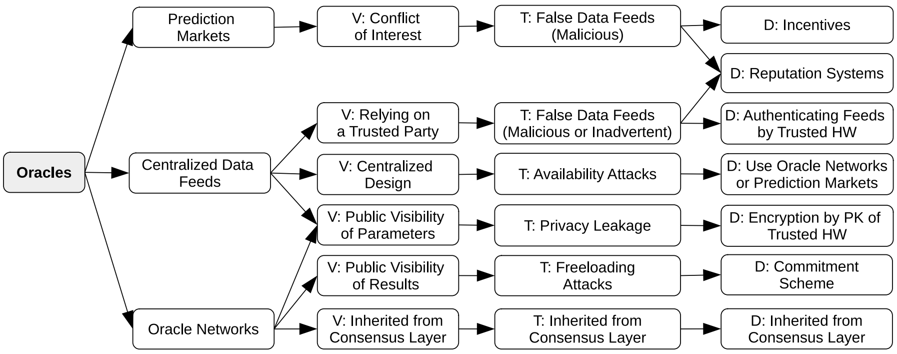

Oracles
Prediction Markets
Centralized Data Feeds
Oracle Networks
V: Conflict of Interest
V: Relying on a Trusted Party
V: Centralized Design
V: Public Visibility of Parameters
V: Public Visibility of Results
V: Inherited from Consensus Layer
T: False Data Feeds (Malicious)
T: False Data Feeds (Malicious or Inadvertent)
T: Availability Attacks
T: Privacy Leakage
T: Freeloading Attacks
T: Inherited from Consensus Layer
D: Incentives
D: Reputation Systems
D: Authenticating Feeds by Trusted HW
D: Use Oracle Networks or Prediction Markets
D: Encryption by PK of Trusted HW
D: Commitment Scheme
D: Inherited from Consensus Layer

Fig. 18. Vulnerabilities, threats, and defenses of the oracles category (application layer).

Availability: Since the execution of dependent smart contracts relies on data feeds delivered by oracles, they need to provide high availability.

We categorize existing oracles according to security implications into three categories. *Prediction markets* (e.g., Augur [201], Gnosis[30]) were created for the purpose of trading the outcome of events – individuals are incentivized to accurately wager on outcomes serving as data feeds, while outcomes are provided by either a centralized reporter or a quorum of reporters. *Centralized data feeds* provide arbitrary data from a single centralized source, and they build on existing blockchain platforms (e.g., Oraclize,[31] Town Crier [202], PDFS [199]). Finally, *oracle networks* internally run a consensus protocol for decentralized agreement on data (e.g., ChainLink [200], Witnet [203]).

Prediction Markets: Augur [201] is a solution designed as the Ethereum smart contract, and it uses its own reputation token. The user creating the market specifies a designated reporter, who delivers the result of the event after it happens. However, the reporter might not report the result or report incorrect results. When the reporter does not report within the specified time frame, Augur shifts the role of a designated reporter on a first-come-first-serve basis. After reporting the outcome, the Augur users have a specific time frame to run a decentralized dispute resolution, and thus obtain a different outcome of the event in the case of misreporting. Augur incentivizes its users to report correct outcomes and file disputes only in justified cases by rewards and deposit-based bonds. Another example of prediction markets is Apollo from Gnosis, which had originally a centralized data source facilitated by Gnosis but was replaced by a decentralized one in version 2.0.

Centralized Data Feeds: Oraclize enriches the data provided to smart contracts by authenticity proofs that are built upon various technologies such as auditable virtual machines and trusted computing. Since authenticity proofs can be large, Oraclize can store these proofs in the distributed file system IPFS[32] instead of directly providing them to the smart contracts. Town Crier [202] is an approach that provides authenticated data feeds to smart contracts by bridging them with public webs through a TEE component. A linkage of TEE with a smart contract is made by storing a public key (PK) of TEE at the smart contract of Town Crier. It relies on the X.509 public key infrastructure (PKI), due to which, the provided data are provably authenticated. PDFS [199] allows content providers to link their Web resources with corresponding smart contracts in the blockchain. In PDFS, the data of content providers are managed in an auditable manner, enabling publicly-verifiable data transparency and consistency of data with the past. Besides content providers, PDFS introduces two entities that arrange a smart-contract-based agreement by specifying a particular content provider required for the execution of the code in the agreement. To ensure that updates are consistent with the past, the authors apply a history tree data structure [204]. The authors additionally support the means to publicly prove censorship by a content provider.

Oracle Networks: ChainLink [200] builds on top of existing blockchains with support for smart contracts, and it distributes the provisioning of data feeds to multiple oracle providers. In detail, ChainLink maintains oracle providers and their reputation, who are selected by a smart contract based on their reputation to form an aggregated final result. When building the final result, ChainLink discards outliers and utilizes the BFT protocol to reach a consensus on a final aggregated value. Witnet [203] is an approach similar to ChainLink but in contrast to ChainLink, Witnet runs its own oracle network with a native token. Witnesses (i.e., content providers) earn reputation points when the content that they deliver matches with the majority's content, and they lose reputation points otherwise. The reputation points serve as a stake in the consensus protocol of the oracle network, hence the higher a node's reputation, the higher the chance that it produces a block. Since more witnesses might become block producers, Witnet allows multiple chains in parallel, forming a DAG.

1) Security Threats and Mitigations: We present a taxonomy of vulnerabilities, threats, and defenses related to the oracle category in Figure 18. In the following, we describe these threats as well as possible defense techniques.

Prediction Markets: May suffer from conflict-of-interest since the creator of the market specifies a data reporter, who might also participate in the market and later report false data for her convenience. Since a prediction market might

[30] https://gnosis-pm-contracts.readthedocs.io/en/latest/
[31] https://www.oraclize.it/
[32] https://ipfs.io/

yield significant financial value for the users with a "correct" guess, the malicious reporter might bribe other reporters of dispute resolution round and still be profitable. Therefore, the incentive protocols of prediction markets must count on this situation and incorporate feasible rewards for honest reporters. Another mitigation for similar attacks is to keep reporters accountable and maintain their reputation in a decentralized fashion (Section IX-B), which involves identity management and verification (see Section VIII-E).

Centralized Data Feeds: Rely on a trusted party [199], [205] that may misbehave or accidentally produce wrong data. For both cases, decentralized identity management can bring accountability while reputation systems further build on it, which disincentivizes malicious behaviors. Another option to cope with possible misbehavior of a trusted party of oracle service is to embed the logic of oracle service into a TEE component [202], whose code is publicly attested. TEE component can interact with the external world using X.509 public key infrastructure (PKI), due to which obtained data are provably authenticated. Since some requests of data feeds might contain private parameters,[33] they can be encrypted by a PK of the TEE and further processed within TEE that communicates with its remote data provider through an encrypted channel, while communication is facilitated by oracle's service operator, as demonstrated in Town Crier [202]. Centralized data feeds might be subject to attacks on availability, leading to interrupted service. A mitigation technique is to use solutions with a higher redundancy, such as oracle networks.

Oracle Networks: Eliminate trust in a single party by running a consensus protocol either natively [203] or utilizing an existing smart contract platform to facilitate the service and its consensus [200]. Running the native consensus protocol of an oracle network imposes the security threats related to the consensus layer (see Section VI). Specific threats to oracle networks are freeloading attacks, in which an oracle provider might copy a publicly visible value provided by other oracles without any effort. The authors of ChainLink [200] propose the usage of a commitment scheme to cope with this attack.

2) Side Effects and Implications of Countermeasures: The data provision time of prediction markets may be too long for many applications, and they are convenient to use only for specific use cases that are limited to provided data events. The data provision time is prolonged even more in the case of disputes, whose resolution may require several days or weeks. In contrast to limited data of prediction markets, centralized data feeds enrich the data domain and significantly shorten the provisioning time.

In the case of oracle providers that offer authenticated data feeds using trusted hardware [202], [205], a vulnerability in trusted hardware (caused by a manufacturer) may result in the entire data feed being compromised. As a mitigation technique to reliance on a trusted manufacturer of TEE, it is possible to use a quorum of several redundant TEEs from multiple manufacturers.

Since ChainLink [200] is an oracle network running over the public smart contract platform, it imposes significant execution costs. To reduce the on-chain cost of BFT execution, the authors discuss the use of threshold-based signatures for collective off-chain signing of the final value; however, freeloading attacks remain unresolved. When freeloading attacks are resolved by a commitment scheme, it negatively impacts the provisioning delay and costs for data providers, who have to submit another transaction with a commitment.

D. Filesystems

Filesystems (FS) serve as a distributed data storage infrastructure that borrows ideas from peer-to-peer storage systems, while additionally incentivizing data preservation by tokens.

1) Fully Replicated FS With Ledger: A naive approach is to store the full content of data at the blockchain, and thus achieve full data replication, extremely high *data durability* (i.e., availability of data) as well as *network expansion factor* (i.e., storage overhead). An example is storing data using the instruction *OP_RETURN* in Bitcoin or storing data as key:value pairs within Namecoin.[34] However, such an approach results in high storage overheads required for full replication of the data among the consensus nodes.

2) Partially Replicated FS With Ledger: To decrease the costs while preserving reasonable durability, partial replication of the data with erasure encoding is often used (e.g., Permacoin [78], Storj [206], and KopperCoin [207]). In erasure encoding, the data block is encoded using two numbers (*k*, *n*), where *n* represents the number of total erasure shares and *k* represents the minimum number of shares required for data recovery. Permacoin incorporates Proof-of-Retrievability in the consensus layer, where consensus nodes store large segments of data provided by an authoritative file dealer. KopperCoin follows a similar approach, but it does not need the trusted dealer for the initial distribution of data files since files are uploaded by the users. Storj uses a 3rd party distributed ledger for storage of metadata, such as file hash, network locations of copies, and Merkle roots of data. Permacoin, KopperCoin, and Storj enable probabilistic audits using Proof-of-Retrievability, which proves that a node stores certain data at the time of the challenge. Filecoin is an incentive mechanism of any distributed FS (e.g., IPFS), and in contrast to previous works, it can guarantee the data possession over a certain time range in a setting of Proof-of-Spacetime. Moreover, Filecoin uses Proof-of-Replication [208], which guarantees physically unique copies of data for each node.

3) Partially Replicated FS Without Ledger: IPFS and Swarm[35] utilize the concept of distributed hash tables (DHT). DHT provides a decentralized data lookup service with key:data mappings, in which the set of nodes storing the data is unambiguously determined by the key associated with the data (i.e., its hash). Since the lookup service and data storage are (partially) distributed, a change in the set of participants causes only a negligible disruption of availability. IPFS does not contain any incentive mechanism and the availability of the data is dependent on its popularity. Although IPFS does

[33] All transactions of permissionless blockchains are visible to public.

[34] https://bit.namecoin.org/

[35] https://swarm-guide.readthedocs.io/en/latest/

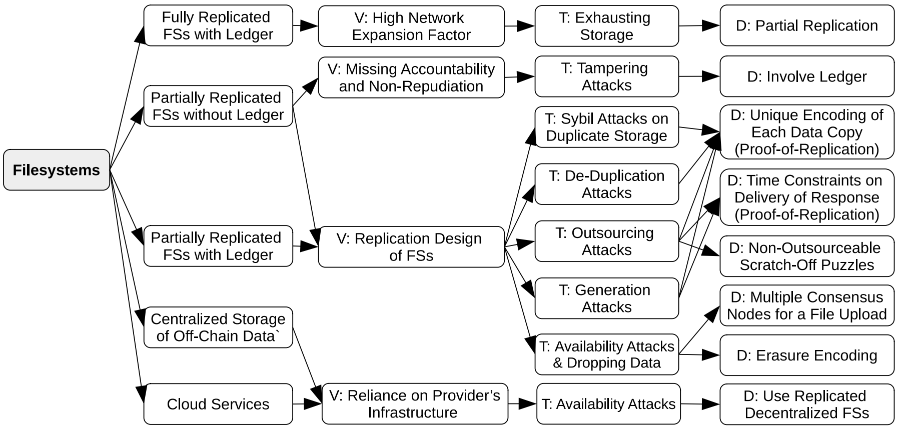

Filesystems
Fully Replicated FSs with Ledger
Partially Replicated FSs without Ledger
Partially Replicated FSs with Ledger
Centralized Storage of Off-Chain Data`
Cloud Services
V: High Network Expansion Factor
V: Missing Accountability and Non-Repudiation
V: Replication Design of FSs
V: Reliance on Provider's Infrastructure
T: Exhausting Storage
T: Tampering Attacks
T: Sybil Attacks on Duplicate Storage
T: De-Duplication Attacks
T: Outsourcing Attacks
T: Generation Attacks
T: Availability Attacks & Dropping Data
T: Availability Attacks
D: Partial Replication
D: Involve Ledger
D: Unique Encoding of Each Data Copy (Proof-of-Replication)
D: Time Constraints on Delivery of Response (Proof-of-Replication)
D: Non-Outsourceable Scratch-Off Puzzles
D: Multiple Consensus Nodes for a File Upload
D: Erasure Encoding
D: Use Replicated Decentralized FSs

Fig. 19. Vulnerabilities, threats, and defenses of the filesystems category (application layer).

not involve a blockchain, it may achieve its properties. In particular, nodes may optionally store a BitSwap ledger that logs data transfers with other nodes.

4) Centralized Storage of Off-Chain Data: Alternatively to decentralized filesystems, decoupling of the data from the blockchain itself through the storage of on-chain integrity proofs (see Section VIII-F) and off-chain data is also an option; however, it introduces a single-point-of-failure and thus may not provide sufficient availability. Besides centralized storage of data, cloud services are promising approaches for decentralized yet manageable data storage, for which integrity and consistency proofs are stored on some blockchain.

a) Security threats and mitigations: We present a taxonomy of vulnerabilities, threats, and defenses related to the filesystems category in Figure 19. While fully replicated and partially replicated decentralized filesystems handle **availability** using decentralized infrastructure, centralized storage with integrity proofs and cloud services relies on a centralized provider. In a **Sybil attack** on a replicated FS, a malicious node claims the storage of multiple copies of the same data. Similarly, in a **de-duplication** attack, more consensus nodes may collude to claim that each of them is storing an independent copy of the data, while only one of the nodes stores the data. These attacks can be prevented by a unique encoding of each data copy proposed in Proof-of-Replication [208]. In an **outsourcing attack**, a malicious consensus node claims the storage of more data than it can physically store while relying on data retrieval from outsourced data providers. In a **generation attack**, a malicious node can re-generate the previously uploaded data upon request using some algorithm, which may increase its chances to be rewarded: a node might commit to storing of a huge volume of generated data.

On top of the unique encoding of each data copy, a mitigation technique for outsourcing and generation attacks is to put time constraints on the delivery of the response by a prover, as proposed in Proof-of-Replication [208]. In detail, the function used for the encoding of data replicas must not be parallelizable (e.g., symmetric encryption in CBC mode) to mitigate generation attacks. In the case of outsourcing attacks, the time constraints must distinguish whether cloud access was made or not. Similar mitigation for outsourcing attacks is the use of non-outsourceable scratch-off puzzles [60], [78], in which the computation of the puzzle requires access to the storage in random order; hence, many round-trips are incurred during one attempt of solving a puzzle. Another attack might target the reputation of a network by dropping data and its redundant copies. A simple mitigation technique is to use multiple consensus nodes for a file upload, which diminishes the chance of the attack being successful. Another mitigation is to increase the durability by erasure encoding.

b) Side effects and implications of countermeasures: Although unique encoding of each data copy thwarts several attacks, on the other hand, it imposes higher overheads for file distribution on clients, which might negatively influence the throughput of data. Similarly, additional overheads on the client during the file upload is imposed by the use of multiple consensus nodes for file upload. Although erasure encoding aims to decrease costs, it must be viewed in a trade-off with the availability of data that is negatively affected by it.

E. Identity Management

Identity management refers to binding identities of entities to their public keys. This concept is also referred to as Public Key Infrastructure (PKI), and it has a few security goals [209]:

Accurate Registration: The user must be unable to register an identity that she does not own.

Identity Retention: The user must be unable to impersonate an identity already registered.

Censorship Resistance: The user must be able to register any identity that she owns.

In computer science, some have conjectured that it is highly unlikely to design an identity management system in which identifiers would be selected in a *distributed* fashion while remaining *secure* and *human-readable*. These three properties are often referred to as Zooko's triangle [210]. However, this situation has been changed with the invention of blockchains; in particular, their immutability feature (Appendix A1).

Namecoin[36] is a native blockchain that facilitates identity management since it allows for the unique registration of key:value mappings. However, searching for a value associated with a key requires full storage and traversal of the blockchain, which is costly. Blockstack [211] is a similar approach as Namecoin, providing decentralized DNS, but in contrast to Namecoin, it off-chains the data storage of domain name mappings and keeps only references to hashes of zone-files in its blockchain. Zone-files (and their referred DNS entries) are stored off-chain. Certcoin [209] is built on top of the Namecoin blockchain, where entities publish their public keys (PK) by posting an identity-PK pair to the Namecoin blockchain. Certcoin utilizes cryptographic accumulators, which represent a space-efficient data structure that supports the verification of set membership within the $\{ID, PK\}$ domain, imposing only a logarithmic time complexity in the number of registered users (in contrast to linear time complexity of Namecoin). Furthermore, to speed up the PK lookup queries, Certcoin leverages the concept of DHT (see Section VIII-D), which enables it to achieve a constant lookup time complexity. Ethereum Name Service[37] (ENS) maps human-readable domain names to Ethereum addresses in a similar fashion as in DNS. The root domain of ENS is maintained by a multi-signature smart contract owned by trustworthy individuals from the Ethereum community. Similarly, uPort [212] utilizes smart contracts to keep a registry that maintains a mapping of the user addresses to hashes of claims[38] that are stored off-chain. In contrast to ENS, uPort does not provide human-readable user identifiers and discusses the possibility of using ENS as a naming layer. Smart contracts for managing identities of humans, groups, objects, and machines are also used in the ERC 725 standard,[39] in which, identity is associated with several keys serving various purposes. ShoCard [213] is another example that builds on top of existing public blockchains, but in contrast to the previous examples, it builds a sidechain containing encrypted identity-specific data such as biometric data, scans of IDs, etc. The user may then decide to whom she will reveal the encrypted data. The Sovrin [214] is an example providing a public permissioned blockchain that consists of consensus nodes approved by Sovrin. It focuses on high throughput and low operational costs. Decentralized Identifiers (DIDs) [215] represent a new type of universally unique identifiers whose control is decentralized since all roots of trust are contained in the blockchain and each entity might create its own root of trust. DID employs the same hierarchical scheme for globally unique strings as URI, and it maps strings to DID documents containing data such as PKs, endpoints of the entity, or links to off-chain data. Only the owner can create, manage, and prove ownership of her DID entries.

1) Security Threats and Mitigations: We present a taxonomy of vulnerabilities, threats, and defenses related to the identity management category in Figure 20. As mentioned by Kalodner *et al.* [216], most of the solutions address the problem of mapping names to values but for identity management, it is essential to build a mapping of entities (i.e., persons, companies) to values. However, to establish such a mapping in a trustworthy way, a human arbitration or a trusted party is needed. For this purpose, oracles (see Section VIII-C) might deliver verified data about the identity of users.

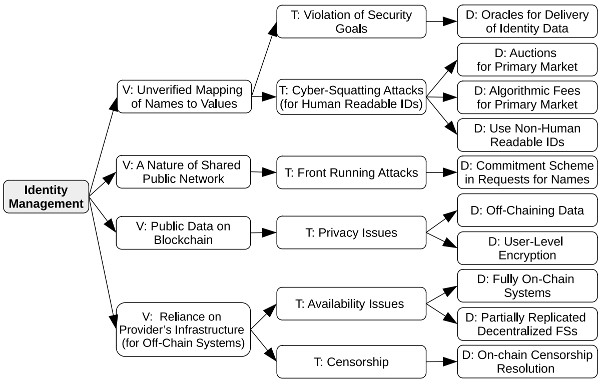

Identity Management
V: Unverified Mapping of Names to Values
V: A Nature of Shared Public Network
V: Public Data on Blockchain
V: Reliance on Provider's Infrastructure (for Off-Chain Systems)
T: Violation of Security Goals
T: Cyber-Squatting Attacks (for Human Readable IDs)
T: Front Running Attacks
T: Privacy Issues
T: Availability Issues
T: Censorship
D: Oracles for Delivery of Identity Data
D: Auctions for Primary Market
D: Algorithmic Fees for Primary Market
D: Use Non-Human Readable IDs
D: Commitment Scheme in Requests for Names
D: Off-Chaining Data
D: User-Level Encryption
D: Fully On-Chain Systems
D: Partially Replicated Decentralized FSs
D: On-chain Censorship Resolution

Fig. 20. Vulnerabilities, threats, and defenses of the identity management category (application layer).

Since the space of human-readable IDs is scarce, they hold some market value in contrast to an almost infinite number of non-human-readable IDs, such as hashes. This opens a door to cyber-squatting attacks, in which anybody might seize an ID that does not belong to her and then sell the ID in the secondary market at an inflated price. Kalodner *et al.* [216] found in 2015 that from around 120,000 registered names in Namecoin, only 28 were not squatted and had human-readable content. They discussed two strategies to prevent such attacks within the primary market, in which names are issued for the first time. These two strategies stand for *auctions* and *algorithmic fees*, both having their respective cons. Auctions are problematic since they may be initiated at any time, and some potentially interested bidders might not be available. An improvement is to specify a fixed time when auctions start. Another approach for coping with cyber-squatting is an algorithmic fee solution, which assigns the price based on the deterministic observables, such as length of the name, a rank of the domain, occurrence of human-readable words, etc. Nevertheless, this approach may require a data feed provider (see Section VIII-C). In contrast to human-readable identifiers, non-human readable identifiers, such as DIDs, do not suffer from the cyber-squatting threat.

Another challenge is a front-running attack (i.e., a MITM attack), in which an adversary may intercept and override the user's transaction with a malicious transaction containing the same domain name but a higher fee. A prevention technique is a variant of the commitment scheme where the user first publishes a sealed (domain) name and public bid, while in the second step she submits the plain text of the name.

Further, identity-related user data that are published on the blockchain are subject to certain privacy issues. Mitigation is to keep only integrity information (such as hashes) on

[36] https://www.namecoin.org/
[37] https://ens.domains/
[38] Claims as such belong to the notaries category (see Section IX-D).
[39] https://erc725alliance.org/

the blockchain, while data should be stored off-chain [211], [212], [213] or encrypted by the user's private key [213]. Moreover, all the approaches that rely on off-chain storage and service provisioning are vulnerable to availability issues (e.g., [212], [215]) and censorship attacks (e.g., [214]). Mitigation that provides censorship evidence is an on-chain censorship resolution [198].

2) Side Effects and Implications of Countermeasures: Side effects of oracles depend on their category, and we mention them in Section VIII-C. If a beginning time of the auctions on the primary market is fixed, bidders can be DoS-ed and prevented from bidding. Anonymization networks and VPNs used by bidders might mitigate this problem. While off-chaining of identity-related data helps to cope with privacy issues and decreases operational costs, on the other hand, it means dependency on centralized storage, which negatively impacts the availability of data. Possible mitigations are partially replicated decentralized filesystems (see Section VIII-D).

F. Secure Timestamping

The role of secure timestamping is to prove that some data existed prior to some point in time – also referred to as proof-of-existence. In the decentralized setting of blockchains, the blockchain serves as a trusted notary that enables such proofs since it provides immutability of the history. Nevertheless, the blockchain "does not understand" the semantics of data that are timestamped, and thus it cannot vet or certify them.

The simple examples of secure timestamping are CommitCoin [217], STAMPD,[40] Bitcoin.com Notary,[41] and OriginStamp [218], all enabling to post a document's hash into a single blockchain transaction. OpenTimestamps[42] and POEX.IO[43] are examples that define a set of operations for creating timestamps and their verification as part of a Merkle tree that aggregates hashes of timestamped objects. The root of the Merkle tree is then stored in the blockchain and later used for verification of timestamped data.

1) Security Threats and Mitigations: Figure 21 depicts a taxonomy of vulnerabilities, threats, and defenses related to the secure timestamping systems. Since these systems have a narrow principle of operation and provided functionality, their attack surface is very limited, too. The main security threats stem from inaccuracy and imprecision of timestamps provided by blockchain network as well as aggregation delays of certain secure timestamping services. As a consequence, the result of certain disputes might be influenced. A possible mitigation technique to improve the accuracy of timestamps is to use timestamping authorities [65] or partial solutions for block timestamp computation [18].[44] A mitigation technique for long aggregation delays is to employ timestamping authorities or use one transaction per hash of the timestamped record. Another class of attacks concerns the availability of timestamped data, for which decentralized filesystems might be utilized as a mitigation technique while storing data in encrypted or plaintext form (depending on the use case).

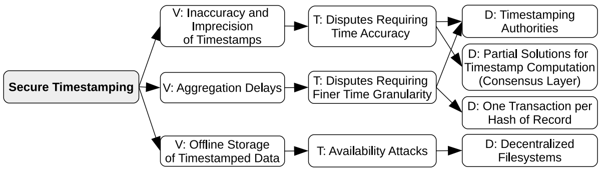

Secure Timestamping
V: Inaccuracy and Imprecision of Timestamps
V: Aggregation Delays
V: Offline Storage of Timestamped Data
T: Disputes Requiring Time Accuracy
T: Disputes Requiring Finer Time Granularity
T: Availability Attacks
D: Timestamping Authorities
D: Partial Solutions for Timestamp Computation (Consensus Layer)
D: One Transaction per Hash of Record
D: Decentralized Filesystems

Fig. 21. Vulnerabilities, threats, and defenses of the secure timestamping category (application layer).

2) Side Effects and Implications of Countermeasures: Although one transaction per hash of a timestamped record mitigates the impact of aggregation delays in solutions such as OpenTimestamps and POEX.IO, on the other hand, it requires a higher amount of data posted to blockchains, and thus it deteriorates the throughput and imposes higher costs. Therefore, the choice of either an aggregated solution or a single hash per record depends on a particular use case.

IX. Application Layer: Higher-Level Applications

In this Section, we elaborate on more specific higher-level applications as opposed to ecosystem applications. In detail, we deal with the following categories: (1) **e-voting**, (2) **reputation systems**, (3) **data provenance**, (4) **notaries**, (5) **direct trading**, (6) **escrows**, (7) **auctions**, and (8) **general application of blockchains**. We describe each of the categories, present a few examples, and then summarize the potential security vulnerabilities, threats, and defenses. For other detailed reviews of blockchain applications, we refer the reader to Casino *et al.* [11] and Zheng *et al.* [12], which in contrast to our work follow domain-oriented classification.

A. E-Voting

Kiayias and Yung *et al.* [219] and Groth [220] state several properties that are desirable in e-voting applications:

Perfect Ballot Secrecy: Implies that finding partial results (i.e., partial tally) before the voting finishes is possible only if all voters are involved in its computation.

Fairness: The final tally can be computed only when all participants already had a chance to cast their vote.

Public Verifiability: Any public observer can verify the validity of all votes and final tally. This is achieved by using a public bulletin board (e.g., blockchain). A consequence of the public verifiability is *dispute-freeness*, i.e., the result of the voting is indisputable.

Self-Tallying: Once the voting stage has finished, anyone can compute the final tally. This property together with fairness ensures that the last voter is unable to compute the tally before casting her vote.

Fault Tolerance/Robustness: A voting protocol is able to recover from a fixed number of faulty voters who do not vote or whose vote is invalid.

Receipt-Freeness: A participant is unable to supply a receipt of her vote after casting the vote. The goal is to prevent vote-selling and post-election coercion.

[40] https://stampd.io/

[41] https://notary.bitcoin.com/

[42] https://opentimestamps.org/

[43] https://poex.io/

[44] Note that this is a countermeasure specific to the consensus layer.

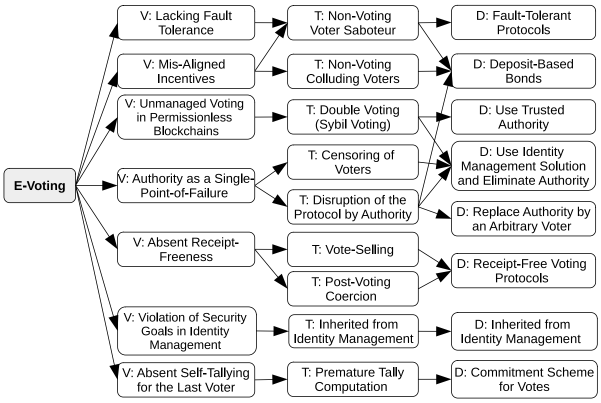

E-Voting
V: Lacking Fault Tolerance
V: Mis-Aligned Incentives
V: Unmanaged Voting in Permissionless Blockchains
V: Authority as a Single-Point-of-Failure
V: Absent Receipt-Freeness
V: Violation of Security Goals in Identity Management
V: Absent Self-Tallying for the Last Voter
T: Non-Voting Voter Saboteur
T: Non-Voting Colluding Voters
T: Double Voting (Sybil Voting)
T: Censoring of Voters
T: Disruption of the Protocol by Authority
T: Vote-Selling
T: Post-Voting Coercion
T: Inherited from Identity Management
T: Premature Tally Computation
D: Fault-Tolerant Protocols
D: Deposit-Based Bonds
D: Use Trusted Authority
D: Use Identity Management Solution and Eliminate Authority
D: Replace Authority by an Arbitrary Voter
D: Receipt-Free Voting Protocols
D: Inherited from Identity Management
D: Commitment Scheme for Votes

Fig. 22. Vulnerabilities, threats, and defenses of the e-voting category (application layer).

E-voting [221], [222] has tried to mimic many of the security properties provided by paper voting. In decentralized e-voting, the protocol is carried out in phases and requires a multiparty computation (MPC) [219], [220] executed by the voters. Decentralized voting involves an interaction among participants and is less robust concerning fault tolerance – i.e., if voters drop out midway, a recovery round has to be initiated. The main advantages of using the blockchain for e-voting are its immutability, public verifiability, enforcing protocol rules by the smart contract, and higher availability [223].

An example of a decentralized e-voting is the Open Voting Network (OVN) [224], which for the first time utilizes blockchain as an instantiation of the public bulletin board. OVN is implemented using Ethereum smart contracts, and it enables boardroom voting of up to $\sim$50 voters with support for two choices. OVN requires the authority to initialize e-voting, compute multiparty keys from data submitted by voters, and reveal the result of the final tally. However, the authority is unable to influence the outcome of voting or compromise the privacy of voters (i.e., cast votes). Although OVN preserves the privacy of voters and provides self-tallying, it does not provide receipt-freeness. OVN uses deposit-based penalties to incentivize the authority and voters to actively participate. Zhang *et al.* [225] present a distributed e-voting scheme that uses the blockchain, but the proposed protocol (like OVN) does not provide a fault tolerance – a restart of voting is required if any voter does not cast her vote. This enables sabotaging of the voting process by a single malicious voter. Venugopalan *et al.* [226] propose a boardroom voting over Ethereum, which supports n voting choices and provides a fault tolerance mechanism. Li *et al.* [227] propose an approach that assumes an interactive honest verifier for the zero-knowledge proof presented; however, the verifier can select a biased challenge, which enables collusion of the verifier and the authority.

1) Security Threats and Mitigations: We present a taxonomy of vulnerabilities, threats, and defenses related to the decentralized e-voting category in Figure 22. The first e-voting-specific group of threats represents vote-selling and post-voting coercion. In vote-selling, the voter can prove to a briber that she voted as agreed, while in post-election coercion the voter is coerced to show her vote by decryption of the blinded vote. Mitigation is to use receipt-free voting protocols that thwart both attacks [228], [229]. Existing solutions for achieving receipt-freeness assume a secret channel (bi-directional [228] or uni-directional [230]), use deniable encryption [231], [232], or employ randomizers [229].

The next threat is double-voting (with Sybil accounts) in the case of unmanaged public voting in permissionless blockchains. To prevent double-voting and ensure that only eligible voters can vote, it is usually required that a voting authority permits voters to vote.[45] Another issue is a possibility of voters not voting despite enrollment, resulting in the sabotage of the voting round (possible in [220], [224], [225]) or privacy issue related to more fine-grained inference of the actual votes of the remaining voters who voted. Deposit-based bonds might be employed as penalties for saboteurs and disincentivize such behaviors [224]. Another countermeasure for saboteurs is a fault-tolerant voting protocol (e.g., [226], [233]), in which the remaining honest voters can recover the final tally even without votes of saboteurs. Since e-voting might assume verified identities of all participants, the next group of threats that is worth noting is inherited from identity management (see Section VIII-E).

The last threat relates to the self-tallying property, which might not hold if no countermeasure is applied, as the last participant can compute the tally and only after that decide on her vote. Simple mitigation is to use an additional "dummy" participant that is handled by the voting authority. Another solution is to enforce participants to first commit to their votes and then cast the committed votes in the later stage [224].

2) Side Effects and Implications of Countermeasures: Receipt-free voting protocols can protect against vote-selling or coercion but on the other hand, they imply additional computational overhead for voters, which increases the cost of running a decentralized protocol. Although authority can prevent double voting, it is trusted with managing voters honestly and completing all actions required for progressing the protocol from one stage to the next stage (e.g., [224], [226]). However, the authority represents a single-point-of-failure, since it might disrupt the execution of the protocol. Deposit-based bonds can be employed as a mitigation technique, or the authority can be replaced by an arbitrary voter for the purpose of execution of the voting protocol (but not for managing voters). Regarding the management of voters, an important threat is the possibility that the authority might censor some eligible voters. A solution for eliminating the authority (or a delegation of this problem to a different application type) is permitting voters to vote upon successful registration of their identities within a certain identity management application (see Section VIII-E). Fault-tolerant voting protocols impose additional overheads, where all remaining voters must actively participate in the recovery round – this implies additional costs on such voters, and it slows down the protocol. Similarly, a commitment scheme for coping with violation of self-tallying

[45] This is represented by know your client (KYC) compliance, and it is related to the principles of permissioned blockchains.

in the case of the last voter implies additional overheads and costs for a dedicated commitment stage.

B. Reputation Systems

Reputation systems commonly serve as a means to (1) measure the level of trust in particular entities that provide a certain service, (2) verify claims of user achievement or authenticity of issued counter-party/ownership tokens. The reputation is usually quantified based on the voting of parties/users that independently analyze the history of interactions/records produced by entities in a reputation query.

In reputation assessment, there are two options to determine the eligibility to rate. In the first option, an arbitrary legitimate participant can rate a product that she has bought or a service that she has consumed. In the second option, only a limited number of selected participants can vote on the authenticity of individual records (e.g., accreditation). In reputation-based systems, identity management is two-fold since the identity of both voters and the record owners/merchants/service providers needs to be verified.

1) Rating by Arbitrary Participants: A privacy-preserving reputation system for e-commerce was proposed in [234]. The authors utilize blinding signatures and merchant-issued tokens to achieve the privacy of reviews and avoid bad-mouthing and ballot-stuffing attacks. A feedback-based reputation approach that utilizes the incentive-based scheme of the Bitcoin network is proposed in [235]. In this approach, any consumer might rate the service of the producer, while obtaining a voucher for the feedback. Zhao *et al.* [236] propose a reputation management scheme that utilizes additive secret sharing to achieve the privacy of participants in reputation assessments.

2) Rating by Several Selected Participants: An example of such a reputation-based application is related to accreditation of educational institutions by other higher-level institutions and organizations [237].

3) Security Threats and Mitigations: We present a taxonomy of vulnerabilities, threats, and defenses related to the reputation systems category in Figure 23. Specific security threats to reputation systems with an arbitrary number of legitimate participants are **bad-mouthing**, **ballot-stuffing**, and **whitewashing** attacks [234]. In bad-mouthing attacks, the customer (e.g., competitor) lies about the product or service, while in the ballot-stuffing attacks, the service provider might increase her reputation by herself. Bad-mouthing can be mitigated by filtering only authorized participants to submit reputation assessments (e.g., by review tokens [234], [235]). Although bad-mouthing cannot be completely prevented, it requires participants to spend resources (e.g., buying a product or paying transaction fees) to be eligible for rating a service provider. Similarly to bad-mouthing, the ballot-stuffing attack cannot be eliminated but only mitigated by requiring to spend resources (i.e., tokens) for each rating entry. If a service provider accumulates a significant negative reputation, it has an incentive for a whitewashing attack – the service provider creates a new service with a neutral reputation, which is unlinkable to her previous service. To mitigate this attack, oracles for obtaining verified data about identities of entities can be employed, possibly as part of the identity management system.

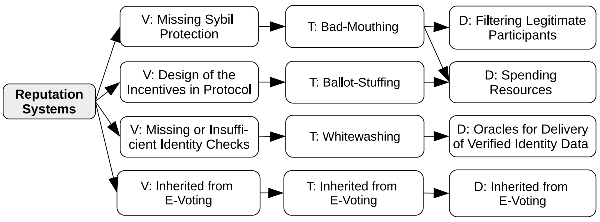

Reputation Systems
V: Missing Sybil Protection
V: Design of the Incentives in Protocol
V: Missing or Insufficient Identity Checks
V: Inherited from E-Voting
T: Bad-Mouthing
T: Ballot-Stuffing
T: Whitewashing
T: Inherited from E-Voting
D: Filtering Legitimate Participants
D: Spending Resources
D: Oracles for Delivery of Verified Identity Data
D: Inherited from E-Voting

Fig. 23. Vulnerabilities, threats, and defenses of the reputation systems category (application layer).

Since reputation systems resemble e-voting in general, concerning security threats are inherited from there (see Section IX-A) and its dependency on identity management (Section VIII-E). In particular, only authorized participants are allowed to participate in the voting process and no duplicate votes are allowed (ensured by identity management), while the votes/ratings should remain blinded (ensured by e-voting) unless the particular use case requires otherwise.

4) Side Effects of Countermeasures: Since some countermeasures in reputation systems are inherited from the e-voting and oracles categories, the side effects are also inherited from these categories.

C. Data Provenance

Data provenance represents the ownership history of an arbitrary object. However, in the cyber-world, objects are represented by data that can be changed, and thus the history must account also for the modifications [238]. Data provenance with the use of blockchains has the potential to resolve various disputes and issues related to intellectual property, authorship, the validity of certificates or other issued documents, etc. Data provenance assumes known verified identities of the involved parties.

An application of data provenance is supply-chain management [239], [240], where the goal is to resolve potential issues in the traceability of goods and provenance of associated data [241]. Blockcloud is an approach that utilizes blockchains for data provenance in cloud computing [242]. The authors aim at the accountability of data creation and manipulation with the intention to detect malicious insiders and intrusions. The idea of using the blockchain for tracking packages and mails as part of supply chain management was proposed in [243]. ChainAnchor [244] is a framework for the commissioning and decommissioning of IoT devices in a cloud-based ecosystem. In a commissioning procedure, devices prove their manufacturing provenance to a verifier in a privacy-preserving fashion without a need for interaction with the manufacturer. An additional goal of ChainAnchor is to reward owners of IoT devices for sharing data in a privacy-preserving manner. A data provenance approach that focuses on the integrity of IoT-generated data is proposed in [245]. A framework to achieve data provenance of multimedia objects such as artworks and books was proposed in [246]. The authors use watermarking techniques to embed transaction metadata of objects into the objects themselves to prove data tampering.

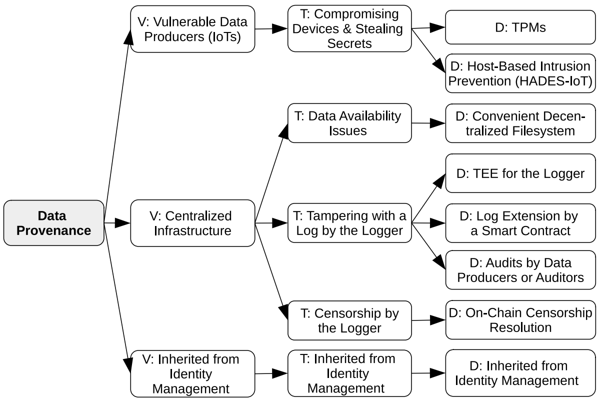

Data Provenance
V: Vulnerable Data Producers (IoTs)
T: Compromising Devices & Stealing Secrets
D: TPMs
D: Host-Based Intrusion Prevention (HADES-IoT)
V: Centralized Infrastructure
T: Data Availability Issues
D: Convenient Decentralized Filesystem
T: Tampering with a Log by the Logger
D: TEE for the Logger
D: Log Extension by a Smart Contract
D: Audits by Data Producers or Auditors
T: Censorship by the Logger
D: On-Chain Censorship Resolution
V: Inherited from Identity Management
T: Inherited from Identity Management
D: Inherited from Identity Management

Fig. 24. Vulnerabilities, threats, and defenses of the data provenance category (application layer).

Catena [247] guarantees a non-equivocation of its append-only log and relatively low storage overheads (i.e., storing all the blockchain headers). The hash of each off-chain data block is added to the append-only log of Catena as a single transaction, which is bound to the previous Catena transaction. The same method of binding consecutive transactions in an append-only log was utilized in Contour [248]. In contrast to general Catena, Contour focuses on non-equivocation during the distribution of open-source software packages by a specified authority. Grech and Camilleri [237] elaborate on the usage of blockchain for the issuance of educational certificates by academic institutions. Such certificates might be issued by institutions whose identities are verified (see Section VIII-E).

1) Security Threats and Mitigations: We present a taxonomy of vulnerabilities, threats, and defenses related to the data provenance category in Figure 24. Since data provenance infrastructure might involve simple IoT devices and sensors that are often not updated nor physically protected, it is important to ensure that these devices are tamper-proof and no secret can be stolen from them. As a countermeasure, Trusted Platform Modules (TPMs) may be used if they are available. We note that the current trend is to involve TPMs in the hardware of contemporary IoT devices. However, TPMs are often not available in numerous legacy IoT devices. In such cases, it is possible to leverage kernel-space and user-space memory isolation as part of the intrusion prevention system (e.g., [249]). Another vulnerability originates from a possible centralized logger component of data provenance solutions (e.g., [199], [247], [248], [250]). Hence, availability issues for data storage must be considered and possibly resolved by a convenient decentralized filesystem approach (see Section VIII-D). Another threat concerning the centralized infrastructure of loggers is the possibility of data tampering and censorship. Data tampering can be detected by data producers or auditors that do periodic audits [251]. To cope with censorship, an on-chain smart contract-based censorship resolution can be utilized [198], [250].

2) Side Effects and Implications of Countermeasures: Although auditors might detect tampering with data by a logger entity, their periodic activity implies high operational costs. A possible option to save such costs is an auditor-free update of the log using a smart contract [199], alternatively combined with TEE [250] to enforce even stronger properties (i.e., the correctness of the execution by logger). However, TEE has also its cons, which we discussed above. Moreover, since countermeasures of the data provenance category depend on the identity management and filesystems category, the side effects are inherited from them as well.

D. Notaries

In contrast to secure timestamping, the role of the notary system is not only to prove the existence of documents at certain points in time but also to vet and certify documents [252]; hence, notary systems assume known verified identities of involved parties who do the vetting. In addition to the above two functionalities, the definition of the notary system involves document preservation, which, however, in the context of the public blockchain is optional. The involved parties may decide whether to store vetted documents in a database of a notary service provider (e.g., PADVA [198]) or whether to keep it privately at the client-side (e.g., Blockusign[46]).

A blockchain-based notarization platform on Ethereum was proposed in the post [253], where an arbitrary number of users/entities with verified identities may sign/approve the documents and their new versions, respectively. The proposal assumes a certification authority that verifies the identities of involved entities, and as an example, the authors suggest the use of the ERC 725 standard [254]. ADVOCATE [255] is an approach for notarization of agreements about personal data processing in IoT between owners of IoT devices and data processing services – both must co-sign an agreement. Mizrahi [256] proposes a system for property ownership, where, all ownership transfers can be executed without any trusted party, but the trusted party is required for introducing the initial ownership record to the blockchain. The ownership register for vehicles was proposed by Notheisen *et al.* [257], where trusted third parties, such as police departments and transport authorities provide and verify vehicle-specific information. SilentNotary[47] is a smart contract-based system for self-certifying of files produced by registered users. PADVA [198] is a Transport Layer Security (TLS) notary service realized as a smart contract-based two-party agreement (i.e., Service Level Agreement). PADVA introduces notary entities that are obligated to periodically check the validity of PKs in a specified set of certificates.

1) Security Threats and Mitigations: We present a taxonomy of vulnerabilities, threats, and defenses related to the notaries category in Figure 25. Notaries inherit security threats related to timestamp accuracy from the secure timestamping category (see Section VIII-F). In addition, since they assume verified identities of involved parties, they inherit security issues from the identity management category (see Section VIII-E). In particular, many notary services assume a centralized identity management system, which might be subject to tampering or censorship issues. Next, a very specific

[46] https://blockusign.co/

[47] https://silentnotary.com/

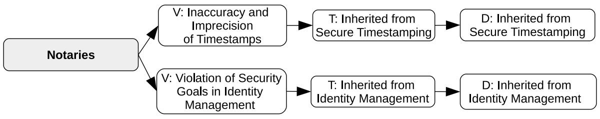

Notaries
V: Inaccuracy and Imprecision of Timestamps
T: Inherited from Secure Timestamping
D: Inherited from Secure Timestamping
V: Violation of Security Goals in Identity Management
T: Inherited from Identity Management
D: Inherited from Identity Management

Fig. 25. Vulnerabilities, threats, and defenses of the notaries category (application layer).

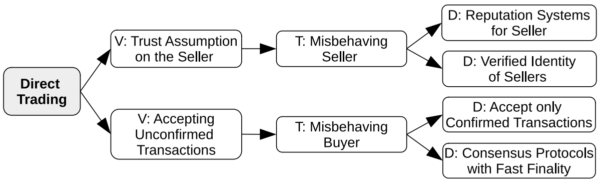

Direct Trading
V: Trust Assumption on the Seller
T: Misbehaving Seller
D: Reputation Systems for Seller
D: Verified Identity of Sellers
V: Accepting Unconfirmed Transactions
T: Misbehaving Buyer
D: Accept only Confirmed Transactions
D: Consensus Protocols with Fast Finality

Fig. 26. Vulnerabilities, threats, and defenses of the direct trading category (application layer).

threat to PADVA is a cheating notary, who quietly does not run the service periodically or runs it only sporadically. Such cheating can be revealed by clients on an ad-hoc basis, who punish notaries by the smart contract logic that causes notaries to lose their deposit.

2) Side Effects of Countermeasures: Since security aspects in notaries are inherited from the secure timestamping and identity management categories, the side effects are also inherited from these categories.

E. Direct Trading

While blockchain-based cryptocurrencies enable native secure transfers of crypto-tokens among their owners, a challenge arises when owners want to exchange crypto-tokens they hold for goods outside of the cryptocurrency blockchain. This challenge is also referred to as the buyer/seller dilemma: "*Should the buyer trust the seller and pay her before receiving goods or should the seller trust the buyer and ship the goods before receiving the payment?*" In the direct trading category, this problem is resolved directly between the buyer and seller, without the need for a mediator, under the assumption of a trusted seller with a verified identity. For example in BIP-70 [258], the buyer first verifies the authenticity of the seller using its X.509 certificate and then issues a payment transaction.

1) Security Threats and Mitigations: We present a taxonomy of vulnerabilities, threats, and defenses related to the direct trading category in Figure 26. The first vulnerability represents the assumption of trust in the seller. For example in BIP-70, the buyer might ask the seller to interrupt the request and get a refund but the seller may misbehave, and thus risk a reputation loss. On the other hand, this might be tolerated if the seller spoofed her identity. A mitigation technique is to use a strong means for identity verification, including assessments from reputation systems. Another attack on BIP-70 that is worth mentioning is the Silkroad trader attack [259], in which a malicious buyer might replace her refund address and then ask the seller for a refund. After a refund, the buyer might plausibly deny receipt of a refund (and ask for a refund again) due to missing authentication on the refund address. Another potential attack related to direct trading is double-spending performed as part of the selfish mining or 51% attacks – therefore, it is important to wait for enough confirmations by the seller before releasing the goods or use consensus protocols having a fast finality (see Section VI-A).

2) Side Effects and Implications of Countermeasures: Enough confirmations by the seller imply a long waiting time for the buyer before the seller releases the goods and issues a receipt for it. In particular, this might be problematic in the case of on-premise purchases. The waiting time is dependent on the time to the finality of the underlying consensus protocol, and thus a consensus protocol with a low time to finality represents a solution. However, it is the means of the consensus layer (see Section VI).

F. Escrows

Escrows address the same problem as direct trading but in contrast to direct trading, escrows do not assume a trusted seller; instead, escrows outsource the trust into the third party, referred to as a *mediator*. The mediator might actively participate in the escrow protocol or she might be involved only in the case of a dispute. According to the decentralization of the mediator, escrow protocols can be split into *single mediator* protocols and protocols with a *group-based mediator*. Goldfeder *et al.* [260] propose a few escrow protocols from both categories, which we briefly review.

1) Single Mediator: Several proposed protocols contain a single mediator and involve 2-of-3 multi-signatures for splitting the control, threshold-based signatures for improving privacy, and protocols leveraging homomorphic properties of elliptic cryptography to achieve privacy (i.e., by blinding the mediator's next address) and non-interactiveness. Another protocol combines multi-signatures with bonds deposited by a mediator to avoid DoS by the mediator. Note that blinding of the mediator's next address hides the execution of the protocol to the mediator only in the case when no dispute has arisen.

2) Group-Based Mediator: In these protocols, disputes are resolved by a majority vote. DoS attack is thwarted as long as the majority of mediators is willing to finish the execution of the protocol. The privacy of some protocols is preserved by blinding, similarly as in the case of single mediator protocols.

An example of a distributed marketplace was proposed in [261], where a marketplace contract lists the products and an escrow agent contract serves for resolution of disputes by a mediator (viz. single or group-based mediator). The authors discuss the integration of logistic parties with verified identities and reputation systems to assess these parties and mediators. OpenBazaar[48] is a distributed marketplace that uses smart contract-based escrows with 2-of-3 multi-signatures, where the mediator is agreed by the buyer and seller. A similar example is Escaroo[49] but in contrast to Openbazaar, it has its own trusted mediator. Natmin[50] is an escrow example that utilizes a public group of mediators for dispute resolution, while the

[48] https://openbazaar.org/

[49] https://escaroo.com/

[50] https://www.natmin.io/

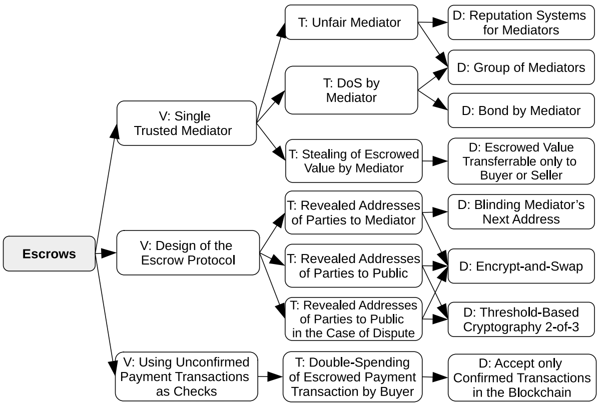

Escrows
V: Single Trusted Mediator
V: Design of the Escrow Protocol
V: Using Unconfirmed Payment Transactions as Checks
T: Unfair Mediator
T: DoS by Mediator
T: Stealing of Escrowed Value by Mediator
T: Revealed Addresses of Parties to Mediator
T: Revealed Addresses of Parties to Public
T: Revealed Addresses of Parties to Public in the Case of Dispute
T: Double-Spending of Escrowed Payment Transaction by Buyer
D: Reputation Systems for Mediators
D: Group of Mediators
D: Bond by Mediator
D: Escrowed Value Transferrable only to Buyer or Seller
D: Blinding Mediator's Next Address
D: Encrypt-and-Swap
D: Threshold-Based Cryptography 2-of-3
D: Accept only Confirmed Transactions in the Blockchain

Fig. 27. Vulnerabilities, threats, and defenses of the escrows category (application layer).

reputation of the mediators is adjusted according to their votes and a result of the dispute.

3) Security Threats and Mitigations: We present a taxonomy of vulnerabilities, threats, and defenses related to the escrow category in Figure 27. The first group of threats refers to a trusted mediator who represents a single-point-of-failure. The mediator might disrupt the execution of the escrow protocol or decide unfairly in the case of a dispute. The existence of these threats depends on the design of an escrow protocol. For example, in the Silk Road marketplace [262], a mediator requests the sending of crypto-tokens to her address, while she is trusted to send the crypto-tokens to the seller upon delivery confirmation from the buyer.[51] However, the mediator might refuse to do so and keep the value for herself. The mitigation techniques are group-based mediators, requiring a consensus of the majority to decide on a case or requiring the mediator to put a bond into the escrow protocol. Another mitigation technique is to use reputation systems for the assessment of a single mediator.

To avoid stealing of the escrowed value by the mediator, the protocol should by design allow releasing value only to the buyer or seller (using rules of a smart contract), while assuming a timeout. For example, an early version of OpenBazaar utilized smart contracts for trading but without any timeout. As a result, many buyers did not release the funds to the seller upon successful delivery. However, when a timeout is adopted, upon its expiration, a seller can unilaterally release and acquire the funds from the escrow.

The next class of threats is related to revealing the private information about running the protocol to the public or mediator (e.g., involved parties, occurrences of disputes). The countermeasures are blinding the mediator's address, threshold-based cryptography, including its special variant *encrypt-and-swap* [260], which uses a 3-of-3 threshold signature protocol that assigns one private share to the buyer and one to the seller, while the third share is known to both parties. Both parties reveal their private share to the mediator, who, upon dispute, provides the winning party with the missing share.

Another possible threat is the double-spending of an unconfirmed escrowed payment transaction by the buyer. For example, if a (naive) escrow protocol requires a mediator to escrow a signed transaction by the buyer and release it to the blockchain only upon delivery confirmation, the buyer might not confirm delivery and perform a double-spending attack (see Section VI). As a prevention technique, unconfirmed transactions should not be accepted by the sellers at all. Moreover, we highlight that some escrow protocols (similarly as atomic swaps) are sensitive to double-spending performed by the selfish mining or 51% attacks – therefore, in these protocols, it is important to wait for enough confirmations or use consensus protocols having a fast finality (see Section VI-A).

4) Side Effects and Implications of Countermeasures: Although group-based mediators are more robust to attacks misusing a trust in a single mediator, they are more expensive to run, requiring the interaction of enough mediators with the blockchain. This in turn slows down the throughput of the escrow protocol. The extra operational overheads are also imposed by the reputation systems for single mediators. Although encoding the escrow logic into a smart contract is supported in some implementations (e.g., OpenBazaar, Escaroo), they require the deployment of a new smart contract per each trade, which is a costly option. Such logic can be implemented even within a single smart contract. Similar to direct trading, not accepting unconfirmed transactions implies a long waiting time for the buyer; this is not the case for the consensus protocols with a fast finality. Finally, using reputation systems brings their security aspects.

G. Auctions

In auctions, sellers promote the sale of their goods through blockchain while buyers place bids for them. Galal and Youssef [263] specify several desired properties of auctions:

Privacy of bids ensures that values of particular bids are not revealed to anybody before committing to them.

Posterior privacy ensures that all bids remain private after the auction ends.

Publicly verifiable correctness enables anybody to verify the results of the auction through the blockchain.

Resistance against DoS ensures that no bidder or auctioneer can prematurely abort a protocol without being penalized.

The authors of [263] instantiate the auction as a smart contract, to which, bidders submit homomorphic commitments of their sealed bids and then reveal their commitments to the auctioneer via a PK encryption. Afterward, the auctioneer deciphers the bids, determines the winner, and announces it to the public while providing zero-knowledge proofs of the correctness. As part of their other contribution [264], the same authors improved on high costs intrinsic to their former work [263] by using zk-SNARKs and its off-chain computation, which requires only a single on-chain proof verification of the whole auction process. However, in both approaches [263], [264], the auctioneer might compromise the privacy of all bids, which led the same authors to propose Trustee [265],

[51] Instead of a single mediator, the Silk Road marketplace utilized several intermediaries to increase the anonymity of the buyer and seller.

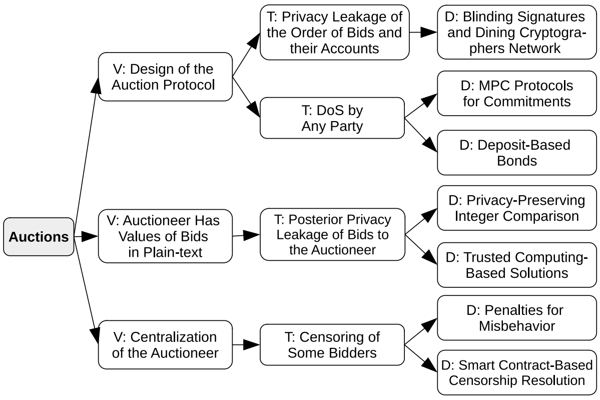

Auctions
V: Design of the Auction Protocol
V: Auctioneer Has Values of Bids in Plain-text
V: Centralization of the Auctioneer
T: Privacy Leakage of the Order of Bids and their Accounts
T: DoS by Any Party
T: Posterior Privacy Leakage of Bids to the Auctioneer
T: Censoring of Some Bidders
D: Blinding Signatures and Dining Cryptographers Network
D: MPC Protocols for Commitments
D: Deposit-Based Bonds
D: Privacy-Preserving Integer Comparison
D: Trusted Computing-Based Solutions
D: Penalties for Misbehavior
D: Smart Contract-Based Censorship Resolution

Fig. 28. Vulnerabilities, threats, and defenses of the auctions category (application layer).

an approach based on TEE. In Trustee, the bidders submit encrypted bids to the auctioneer's TEE, which confidentially evaluates a winner and generates a blockchain transaction proving it.

Strain [266] is an auction protocol that guarantees the privacy of bids against malicious bidders and, in contrast to [263], [264] also against the auctioneer. Strain is executed in four rounds (i.e., four blocks), and it assumes a semi-honest (i.e., passive) auctioneer who acts as a judge verifying the correctness of zero-knowledge proofs. Neither the auctioneer nor a malicious bidder learns anything about bids of honest bidders; however, the order of the bids is leaked to the public. Strain requires each bidder to commit publicly to her bid, while the proposed scheme enables a majority of honest bidders to open other bidders' commitments in the case that they abort the protocol. For the sake of efficiency (i.e., a constant number of rounds), the authors provide weaker security properties in contrast to MPC protocols, where no semi-honest judge is required. Finally, the authors propose an extension of their scheme to support the anonymity of all bidders by blinding RSA signatures and the Dining Cryptographers (DC) network.

Another approach that preserves the privacy of the bid values (but not the privacy of their order) is proposed in [267]. The protocol requires off-chain interaction for two-party computation protocol that performs a pairwise comparison of blinded bids among bidders and the auctioneer.

1) Security Threats and Mitigations: We present a taxonomy of vulnerabilities, threats, and defenses related to the auctions category in Figure 28. There are several possible issues related to privacy leakage in the auction protocols. The first privacy issue stands for revealing addresses of bidders and/or order of their bids to the public. For example, the authors of [263] do not provide anonymity of bidders, since bidders use their existing Ethereum addresses to interact with the protocol. A mitigation technique using blinding RSA signatures and the DC network was proposed in [266]; however, network-level attacks on revealing locations/IP addresses of parties remain possible (see Section V).

Since the auctioneer of some protocols (e.g., [263], [264]) sees the bidders and their bids in plain-text, she might either intentionally or accidentally (e.g., an external compromise) leak these data (and their corresponding proofs) that are attributable to particular bidders. The protection technique is to avoid the auctioneer from seeing the plain-text of the bids, and instead use privacy-preserving integer comparison (e.g., [266]) or trusted computing-based solutions (e.g., [265]).

Another threat originates from a centralized auctioneer who might censor some bidders (e.g., due to collusion with another bidder) by claiming that their bids are invalid, i.e., claiming that a commitment does not open to the sealed bid. To cope with this threat, the auction can utilize a smart contract-based resolution mechanism in which the bidder might prove the opposite by revealing her value of the bid, causing a penalization of the auctioneer [263]. Deposit-based bonds and penalization of the involved parties can be used as a protection against abortion of the protocol (i.e., DoS) by any party. For example, the protocol of [263] splits penalties to honest participants in the case of abortion by some party. Another option to cope with the abortion of the auction protocol is to use multiparty computation (MPC) for the commitment of sealed bids [266], which enables the opening of the commitment of the aborting party and thus to continue in the auction protocol.

2) Side Effects and Implications of Countermeasures: Although censoring of bids can be prevented by a smart contract-based resolution mechanism [263], it has privacy consequences since it leaks the value of the bid and its corresponding bidder. Deposit-based bonds can disincentivize the abortion of the auction protocol but they require a restart of the round. In contrast to it, MPC protocols for commitments can recover the current auction round but with additional overheads and costs imposed on a smart contract platform. In the case of using a TEE-based solution, the malicious auctioneer might perform censorship of some bidders due to collusion with other bidders. To avoid this threat, the authors of [265] propose a smart contract-based mechanism that verifies whether the set of sealed bids submitted to the smart contract corresponds to the list of bids in the proof generated by TEE. Authors also discuss another option to cope with this attack by embedding an SPV client within the TEE that would evaluate the state of the blockchain; however, this solution would impose a high memory consumption of already constrained TEE and would expose TEE to vulnerabilities presented in SPV client. Another implication of using TEE-based auction is the possibility of a local replay attack discussed in [265], where the auctioneer might provide different instances of TEE with a different subset of the bids, and hence obtain the values of particular bids. As described in [265], such a privacy threat can be prevented by a TEE specific construct called *hardware monotonic counter*, which cannot be reverted once incremented. Finally, one has to consider that a vulnerability in trusted hardware may result in the unfairness of the auction process and compromise of its privacy.

H. General Applications of Blockchains

There are many use cases of applying blockchain to a particular domain that contains mutually untrusted participants:

these participants are represented by consensus nodes executing a consensus protocol. Applications from this category focus on leveraging inherent blockchain features and sometimes even on non-inherent features (see Appendix A).

An example that uses permissioned blockchain for the management of healthcare data is proposed in [268], in which new data are included in the blockchain upon majority agreement. The consensus nodes of this approach are represented by a patient, her family, and healthcare providers. The authors discuss an issue regarding the right to delete personal data, which is contrary to the inherent features of the blockchain. A data protection framework for energy grids and power systems was proposed in [269]. The authors suggest that smart meters act as consensus nodes and store the data of the blockchain in their memory.[52] DistBlockNet [270] is the approach for ensuring state consistency among multiple SDN controllers with the utilization of dedicated blockchain. In detail, flow rule tables of SDN controllers are managed by these controllers, while other entities in the system use blockchain as a reference point for downloading these tables. A federated permissioned blockchain used as a cloud-based data storage requiring the consensus of all nodes[53] is proposed in work [271]. The authors propose to use two tiers of blockchain. The blockchain at the first tier is private and serves for consensus of participants, while the blockchain of the second tier is public PoW (e.g., Bitcoin) and serves for periodic publishing of integrity proofs of the first tier blockchain. Two-tier blockchain was also applied in the domain of IoT [272]. The authors of [272] deem the first tier blockchain as local to a group of IoT devices owned by a single party, while the second tier blockchain serves for sharing the data among multiple untrusted parties. The authors demonstrate the applicability in a case study involving several smart homes.

1) Security Threats and Mitigations: Security threats of this general category vary case by case and usually concern the privacy of data shared among involved parties [268], [269], [272]. Another common issue is an application of the blockchain with unrealistic assumptions about the target environment, e.g., low processing or storage performance of smart meters/IoT devices, no HW support for asymmetric cryptography, no tamper-proof means in devices producing transactions, etc. Some applications try to optimize throughput or finality of the blockchain by introducing their own consensus mechanisms (e.g., [271], [272]); however, this might not be the best option since new attack vectors might be created. A general recommendation is to study and understand security issues and countermeasures of the state-of-the-art approaches of the consensus layer (see Section VI) as well as privacy concerns presented at the RSM layer (see Section VII).

X. Lessons Learned

In this Section, we summarize lessons learned concerning the security reference architecture (SRA) and its practical utilization. First, we describe the hierarchy of security dependencies among particular layers of the SRA. Second, assuming such a hierarchy, we describe a security-oriented methodology for designers of blockchain platforms and applications. Next, we summarize the design goals of particular blockchain types and discuss the security-specific features of the blockchains. Finally, we analyze observations from the incidents that occurred in practice, limitations in the literature, and we propose future research directions.

A. Hierarchy of Dependencies in the SRA

In the proposed model of the SRA, we observe that consequences of vulnerabilities presented at lower layers of the SRA are manifested in the same layers and/or at higher layers, especially at the application layer. Therefore, we refer to *security dependencies* of these layers on lower layers or the same layers, i.e., *reflexive* and *bottom-up* dependencies. We describe these two types of dependencies in the following.

1) Reflexive Dependencies: If a layer of the SRA contains some assets, it also contains a reflexive security dependency on the countermeasures presented in the same layer. It means that a countermeasure at a particular layer protects the assets presented in the same layer. For example, in the case of the consensus layer whose protocols reward consensus nodes for participation, the countermeasures against selfish mining attacks protect rewards (i.e., crypto-tokens) of consensus nodes. In the case of the RSM layer, the privacy of user identities and data is protected by various countermeasures of this layer (e.g., blinding signatures, secure multiparty computations). Another group of reflexive security dependencies is presented at the application layer. Although the application layer contains some bottom-up security dependencies (see Figure 15), we argue that with regard to the overall stacked model of the SRA they can be viewed as reflexive security dependencies of the application layer.

2) Bottom-Up Dependencies: If a layer of the SRA contains some assets, besides reflexive security dependencies, it also contains bottom-up security dependencies on the countermeasures of all lower layers. Hence, the consequences of vulnerabilities presented at lower layers of SRA might be manifested at the same layers (i.e., reflexive dependencies) but more importantly, they are manifested at higher layers, especially at the application layer. For example, context-sensitive transactions and partial solutions as countermeasures of the consensus layer can protect against front-running attacks of intra-chain DEXes, which occur at the application layer. Another example represents programming bugs in the RSM layer, which influence the correct functionality at the application layer. The eclipse attack is an example that impacts the consensus layer from the network layer – a victim consensus node operates over the attacker-controlled chain, and thus causes a loss of crypto-tokens by a consensus node and at the same time it decreases honest consensus power of the network. In turn, this might simplify selfish-mining attacks at the consensus layer, which in turn might impact the correct functionality of a blockchain-based application at the application layer. Bottom-up security dependencies are also presented in the context of the application layer, as we have already mentioned in Section VIII.

[52] These are unrealistic assumptions.

[53] Note that such a proposal has very low fault tolerance.

TABLE I
PROS AND CONS OF VARIOUS CATEGORIES WITHIN THE FIRST THREE LAYERS OF THE STACKED MODEL

Layer			Category in a Layer	Pros	Cons
Network Layer			**Private Networks**	● low latency, high throughput ● centralized administration, ease of access control ● the privacy of data and identities ● meeting regulatory obligations ● resilience to external attacks	● VPN is required for geographically spread participants ● suitable only for permissioned blockchains ● insider threat and external attacks at nodes with administrative privileges
			Public Networks	● high decentralization ● high availability ● openness & low entry barrier (low cost of broadband connection, resistance to regulations)	● high and non-uniform latency ● single-point-of-failure (DNS, IP, and ASes are managed by centralized parties) ● external adversaries (botnets, compromised BGP/DNS servers) ● stolen identities
Consensus Layer			**PoR**	● high cost of overriding the history of blockchain ● high scalability	● high operational costs ● low throughput ● low finality
			BFT	● high throughput (with a small number of nodes) ● fast finality	● low scalability ● high communication complexity ● limited number of nodes (efficient use only in permissioned blockchains)
			PoS	● energy efficiency	● PoS specific attacks and issues ● supports only semi-permissionless setting ● slow finality
			PoS+BFT	● energy efficiency ● high scalability ● probabilistic security guarantees ● lower communication overheads than BFT	● some PoS specific attacks ● supports only semi-permissionless setting
			PoR+BFT	● high scalability ● fast finality	● spending some scarce resources
			PoR+PoS (i.e., PoA)	● high scalability	● spending some scarce resources ● some PoS specific attacks ● slow finality
Replicated State Machine Layer	**Transactions**	**Protecting Identities**	**Standard Approach**	● fast processing ● ease of verification	● identities are only pseoudonymous and can be traced to IPs ● all data of transactions are publicly visible
			Standard Approach + Mixers	● privacy identity protection of users in a group ● ease of verification	● additional complexity, in some cases unlinkability by the mixer or involved parties in a group ● all data of transactions are publicly visible
			NIZKs and Ring-Signatures	● identities are anonymized to the extend of the group	● additional computation overheads for running the schemes
			MPC Blinding Signatures, Layered-Encryption	● unlinkability for all involved parties	● additional computation overheads for running the schemes
		Protecting Data	**NIZKs, Blinding Signatures, Homomorphic Encryption**	● privacy of data in cryptocurrency platforms	● additional computation overheads for running the schemes
			Trusted Transaction Managers, Trusted Hardware, MPC	● privacy of data in transactions of smart contract platforms	● additional computation overheads for running the schemes
	Smart Contracts		**Turing-Complete Languages**	● smart contracts may contain an arbitrary programming logic	● wide surface for making the programming bugs that often results in vulnerabilities
			Turing-Incomplete Languages	● small attack surface and emphasis on safety	● the programming logic serves only for limited purposes

B. Methodology for Designers

A hierarchy of security dependencies in the SRA can be utilized during the design of new blockchain-based solutions. When designing a new **blockchain platform** or a new **blockchain application**, we recommend designers to specify requirements on the blockchain features (see Appendix A) and afterward analyze design options and their attack surfaces at the first three layers of the stacked model of SRA. We briefly summarize the pros and cons of particular categories within the first three layers of SRA in Table I, while security threats and mitigations are covered in Section V, Section VI, and Section VII.

TABLE II
PROS AND CONS OF SOME CATEGORIES FROM THE APPLICATION LAYER

Application Category	Subcategory	Pros	Cons
Wallets	**Server-Side Hosted Wallets**	• simplicity of control for end-users • no storage requirements for end-users	• keys stored at the server, susceptibility to the theft of keys by external or internal attacks • single-point-of-failure, availability attacks
	Client-Side Hosted Wallets	• simplicity of control for end-users • no storage requirements for end-users • keys stored locally	• single-point-of-failure, availability attacks • possibility of key theft by malware • possibility of tampering attacks
	Self-Sovereign Wallets	• keys stored locally or in a dedicated hardware device	• moderate storage requirements for end-users • more difficult control for end-users • extra device to carry in the case of hardware wallet
Exchanges	**Centralized Exchange**	• a high throughput and speed of operations • the simplicity of control for end-users • low costs for exchange transactions • trading of obscure crypto-tokens	• risk of insider threat due to centralization • external threats to exchange infrastructure • overheads for secure storage of secrets • a fee specified by the operator
	Direct Cross-Chain Exchange	• fairness of the exchange • no fee to any operator	• costs for 4 transactions of the atomic swap • user has to find the counter-order on her own • counter-orders might not exist • a lower throughput than in a centralized exchange • a higher complexity for end-users
	Cross-Chain DEX	• fairness of the exchange • order matching made by DEX • trading of obscure crypto-tokens	• costs for 4 or 6 transactions of the atomic swap • a lower throughput than in a centralized exchange • a fee specified by the operator
	Intra-Chain DEX	• fairness of the exchange • uniform finality for every pair • a high speed of operations	• a limited number of pairs that are specific to the target platform • a fee specified by the operator • costs for smart contract execution
Oracles	**Prediction Markets**	• early (close to accurate) estimation of the future event's result • decentralization	• possible conflict of interest • a limited set of data specific to a few events • a long time to obtain a final result, especially in the case of disputes
	Centralized Data Feeds	• wide range of data • fast provisioning time • handling of private parameters of requests • censorship evidence	• centralization (accidentally or intentionally wrong data) • availability issues
	Oracle Networks	• decentralization • wide range of data • fast provisioning time	• unsupported private parameters of requests • publicly visible data and requests
Filesystems	**Fully Replicated FSs with Ledger**	• a high availability • accountability and auditability	• a high storage overheads and operational costs • a high price
	Partially Replicated FSs with Ledger	• reasonably high availability • accountability and auditability • a lower price than in a full replication	• attack vectors specific to partial replication
	Partially Replicated FSs without Ledger	• reasonably high availability • a lower price than in a full replication	• a lack of native accountability and auditability • low durability due to a lack of incentives for storage
	Centralized Storage of Off-Chain Data	• a low price • accountability and auditability	• a low availability

On top of that, we recommend the designers of a new **blockchain application** to analyze particular options and their security implications at the application layer of SRA. We list the pros and cons of a few categories from the application layer in Table II,[54] while security threats and mitigation techniques of this layer are elaborated in Section VIII and Section IX. During this process, we recommend the designers to follow security dependencies of the target category on other underlying categories (see Figure 15) if their decentralized variants are used (which is a preferable option from the security point-of-view). For example, if one intends to design a decentralized reputation system, she is advised to study the security threats from the reputation system category and its recursive dependencies on e-voting, identity management, crypto-tokens & wallets, and (optionally) filesystems.

1) Divide and Conquer: If a designer of the blockchain application is also designing a blockchain platform, we recommend her to split the functionality of the solution with the divide-and-conquer approach respecting particular layers of our stacked model. In detail, if some functionality is specific to the application layer, then it should be implemented at that

[54] Note that the table contains only categories with specified subcategorizations that represent the subject to a comparison.

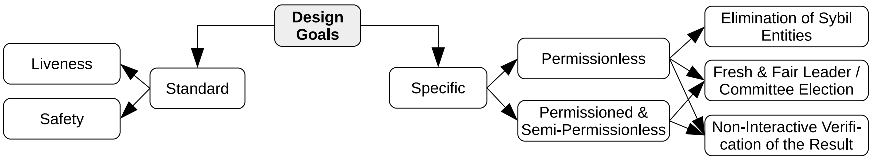

Design Goals
Liveness
Safety
Standard
Specific
Permissionless
Permissioned & Semi-Permissionless
Elimination of Sybil Entities
Fresh & Fair Leader / Committee Election
Non-Interactive Verification of the Result

Fig. 29. Standard and specific design goals of consensus protocols.

layer. Such an approach minimizes the attack surface of a solution and enables isolating the threats to specific layers, where they are easier to protect from and reviewed by the community. A contra-example is to incorporate a part of application layer functionality/validation into the consensus layer. The consensus layer should deal only with the ordering of transactions, and it should be agnostic to the application.

Nevertheless, it is worth noting that the divide-and-conquer approach might not be suitable for some very specific cases. For example, some decentralized filesystems (see Section VIII-D) might combine data storage as an application-layer service with the proof-of-storage consensus algorithm, presented at the consensus layer. Therefore, the consensus layer also embeds a part of functionality from the application layer. However, when filesystems are in security dependencies of the target application other than filesystems, one should realize that they are usually running on a different blockchain or infrastructure than the target application, and this exception is not a concern.

C. Blockchain Types & Design Goals

We learned that the type of a blockchain (see Section III-B) implies the specific design goals of its consensus protocol (see Figure 29), which must be considered on top of the standard design goals (i.e., liveness and safety) and the inherent features (see Appendix A1) during the design of a particular blockchain platform and its consensus protocol. In the following, we elaborate on such specific design goals.

1) Permissionless Type: The first design goal is to *eliminate Sybil entities* – such elimination can be done by requiring that some amount of scarce resources is spent for extension of the blockchain, and hence no Sybil entity can participate. This implies that no pure PoS protocol can be permissionless since it never spends resources on running a consensus protocol. The next design goal is *a fresh and fair leader/committee election*, which ensures that each consensus node influences the result of a consensus commensurately to the number of scarce resources spent. Moreover, freshness avoids the prediction of the selected nodes, and therefore elected nodes cannot become the subject of targeted DoS attacks. The last design goal is the *non-interactive verification* of the consensus result by any node – i.e., any node can verify the result of the consensus based on the data presented in the blockchain.

2) Permissioned and Semi-Permissionless Types: These types of blockchains require *fresh and fair leader/committee election* as well as *non-interactive verification* of the result of the consensus. However, in contrast to the permissionless blockchains, they do not require a means for the elimination of Sybil entities, as permission to enter the system is given by a centralized entity (i.e., permissioned type) or any existing consensus node (i.e., semi-permissionless type).

3) Blockchain Types and Incentives: We observed that no application running on a *public* (permissioned) blockchain has been able to work without introducing crypto-tokens (i.e., an incentive scheme), even if the use case is not financial in nature, e.g., e-voting, notaries, secure timestamping, or reputation systems. In these blockchains, incentive schemes serve as a means for the elimination of Sybil entities, besides other purposes. The situation is different in the context of *private* (permissioned) blockchains, which are usually provisioned by a single organization or a consortium and do not necessarily need crypto-tokens to operate. Misaligned incentives can cause consensus-level vulnerabilities, e.g., when it becomes profitable to drop blocks of other nodes to earn higher mining rewards [82] or transaction fees [273]. The design of incentive mechanisms is a research field by itself and we refer the reader to the work of Leonardos *et al.* [274].

D. Security-Specific Features of Blockchains

We realized that consensus protocols are the target of most financially-oriented attacks on the decentralized infrastructure of blockchains, even if such attacks might originate from the network layer (e.g., routing and eclipse attacks). The goal of these attacks is to overturn and re-order already ordered blocks while doing double-spending. Hence, the finality is the most security-critical feature of the consensus layer. The finality differs per various categories of the consensus layer. The best finality is achieved in the pure BFT protocols, and the worse finality is achieved in the single-leader-based PoR and PoS protocols. On the other hand, combinations of the BFT with PoS protocols (i.e., introducing committees) slightly deteriorate the finality of BFT in a probabilistic ratio that is commensurate to the committee size. In the case of PoR protocols with partial solutions, finality is improved as opposed to pure PoR protocols; however, it is also probabilistic, depending on the number of partial solutions.

E. Incidents in Practice

We list several incidents at each layer of the SRA in Table VI, Table V, Table IV, and Table III of Appendix. The observations about the number of different incident types vary layer by layer. In the case of the network layer, many of the attacks described in the SRA occurred or were demonstrated with a proof-of-concept. However, incidents that occurred at the consensus layer mostly contained 51% attacks with double-spending, while incidents that occurred on the application layer were mostly caused by the exploitation of centralized components. In the case of the RSM layer, most of the incidents occurred due to bugs in smart contracts. Finally, we observed that most of the incidents that occurred at the application layer were caused due to a single-point-of-failure, e.g., centralized components or the insider threat.

Although the number of occurred incident types is low as compared to described vulnerabilities and threats, we argue that the adoption of blockchains for practical applications is still in its infancy, and thus we may expect that the number of different incident types observed in practice will grow.

F. Limitations in the Literature and Practice

1) Applications of Blockchains: Although the literature contains surveys and overviews [11], [12], [275] of blockchain-based applications, these works introduce only domain-oriented categorizations (i.e., categories such financial, governance, security, education, supply chain, etc.) and they do not investigate the security aspects and functionalities that these applications leverage on and whether some of the applications do not belong to the same category from the security and functionality point-of-view. To address this limitation, we provide a security-driven functionality-oriented categorization of blockchain-based applications (see Section VIII), which is agnostic to an application domain and thus can generalize different application scenarios. Furthermore, our proposed categorization enables us to model security and functionality-based dependencies among particular categories, which is not possible with state-of-the-art categorizations.

2) Centralization: Even though blockchains are meant to be fully decentralized, we have seen that this does not hold at some layers of the SRA – the network and application layers, in particular. In the network layer, some attacks are possible due to centralized DNS bootstrapping, while in the application layer a few categories utilize centralized components to ensure some functionality that cannot run on-chain or its provisioning would be too expensive and slow, which, however, forms the trade-off with the security. Some applications might depend on components from other application categories (e.g., identity management) but implementing these components in a centralized fashion, even though there exist some decentralized variants that are gaining popularity (e.g., DIDs [215] for identity management).

G. Future Research Directions

1) Fast Finality: Although finality is the most security-critical feature of the consensus layer (see Section X-D), it forms the trade-off with scalability. Therefore, we believe that the future focus of the consensus research should be in a thorough evaluation of this trade-off across various consensus protocols.

2) Network-Layer Security: We learned that a substantial body of security research in blockchains is focusing on the consensus and RSM layers since these layers are mostly identified with the blockchains. As opposed to them (see Figure 1), the network layer is not so popular even though the serious threats originating from this layer might hurt the higher layers and their assets. Therefore, a potential direction for future research lies in studying the security aspects of network protocols, their suitability for a decentralized environment, and potential improvements.

3) Privacy Preservation & Performance: All cryptographic privacy preservation techniques (see Section VII-A1) bring additional computation overhead, and thus they negatively impact the throughput of the blockchains. On the other hand, privacy-preserving solutions that are based on the trusted hardware might provide higher performance, but they rely on the manufacturer of trusted hardware and the assumption that it will not be compromised. Therefore, we believe that optimizing the trade-off between performance and privacy-preservation is an important future research direction concerning the RSM layer.

4) Security Analysis of the Application Layer: Although many references included in this study are presented in the application layer, only a very few of them analyze thoroughly security aspects of a particular application layer category or its instance. Therefore, as a future research direction, we recommend the authors of the blockchain-based applications to analyze the resistance of their applications to all known threats of a particular application category (e.g., with help of our work), while broadly think of new vulnerabilities and threats that might be specific to their application type.

5) Decentralization: Since some blockchain applications utilize centralized components while their decentralized variants already exist Section X-F, we suggest that a potential future direction for researchers and practitioners might be the concept of a fully decentralized blockchain ecosystem. Such an ecosystem might consist of only decentralized (or partially decentralized) application types, for example, the ones that we reviewed in the application layer of the SRA.

XI. Discussion

In this Section, we first discuss the versatility and modularity of the stacked model that is the proposed security reference architecture (SRA) based on. Then, we outline a few additional security aspects related to blockchains, which, for clarity and simplicity, we have not pursued throughout this article or mentioned them only tangentially. Finally, we discuss a few types of blockchain-oriented applications that directly inherit security aspects from already existing categories; therefore, we omitted such application types in our work.

A. Stacked Model

1) Versatility: The hierarchical stacked model that is the SRA based on was already utilized in other domains before. A well-known example is the ISO/OSI model with seven layers or later derived TCP/IP model with four layers in the field of communication networks. The stacked model was also applied in cloud computing, referred to as cloud stack [276], in which, each layer represents one service model in the model's hierarchy. Nevertheless, the stacked model was also applied in the field of blockchains [16].

The versatility of the stack model allows not only for modeling the hierarchy in a particular domain but also for partitioning the corresponding security issues and their countermeasures based on the layers of the model. This was done for the ISO/OSI model [277], TCP/IP model [278], and cloud computing [8], while in this work we focus on the security threats related to blockchains and propose the SRA.

2) Modularity: The stack model of SRA enables extensions of particular categories within each layer by adding new vulnerabilities, threats, and their respective countermeasures. Likewise, the new categories can be modularly added to each of the SRA layers. Afterward, the security implications of a new category, threat, or a countermeasure within some layer should be studied with regard to particular categories in higher layers – a new category might be beneficial or detrimental to them from a security point-of-view. When introducing a new defense or mitigation technique, it is also important to evaluate its side effects and implications on the features of the blockchain that are manifested at the same and higher layers.

A general guideline for extending the SRA is to introduce only such categories that have unique features from the security point-of-view, while in the case of the application layer, functionality point-of-view should be considered as well.

B. Additional Security Aspects

1) Secure Cryptography Primitives: We emphasize that for each layer of our stacked model, we assume the use of secure cryptographic primitives with recommended key lengths[55] that are based on existing standards (e.g., [279], [280]). Examples include secure communication (i.e., network layer), the use of private keys for transaction signing (i.e., consensus layer), and password management for blockchain-based services (i.e., application layer). Since the area of cryptographic primitives is standardized and extensively covered in existing research, we treat security incidents that break these primitives as out-of-scope in the current article.

2) Semantic Bugs: We deal with semantic bugs only at the level of the RSM layer as part of the smart contracts (see Section VII-B). However, we emphasize that semantic bugs in the blockchain infrastructure may occur at each of the proposed layers, whereas in the case of the RSM layer, besides smart contracts, they may occur in compilers, interpreters, etc. In this work, we assume that the software of the blockchain-related infrastructure does not contain any programming semantic bugs at each of the layers, and it provides the expected functionality. On the other hand, we emphasize that these semantic bugs had already accounted for several incidents in the past, e.g., [281], [282], [283], [284]. To achieve safe and correct software at each of the layers, similar to the case of the RSM layer, developers and designers should utilize verification tools, testing, code reviews, audits, known design patterns, best practices, etc.

C. Other Blockchain-Oriented Applications

There are several other applications of blockchain that we do not mention in our work because their security aspects are inherited from one or more categories presented in Section VIII and Section IX. For example, insurance applications running on smart contract platforms inherit security aspects from the oracles category, as they require data to be delivered into the blockchain from the outside world. The next example is the trading of crypto-tokens within the same blockchain platform – it inherits security aspects from the crypto-tokens and wallets category (see Section VIII-A). Another example is cross-chain communication, which is a generalization of the exchanges category, and it also inherits most of the security aspects from it.

XII. Related Work

The security reference architecture that has been presented in our work offers a comprehensive overview of blockchain-related security vulnerabilities, threats, and mitigation techniques. We adapted a custom version of the four-layer stacked model, initially presented in the work of Wang *et al.* [16]. In the following, we present an overview of the state-of-the-art survey papers related to blockchain research while we highlight the differences in contrast to our work. We consider three groups of blockchain-oriented research: (1) papers that use a flat categorization of threats and vulnerabilities, (2) papers that use a stacked or other multi-layered models, and (3) papers that focus on incidents that belong to a single layer.

A. Research With Flat Categorization

Bonneau *et al.* [177] present the first major survey of blockchain-specific security aspects, with a particular focus on Bitcoin and cryptocurrencies. The authors aim at the consensus-layer properties, although some network-layer aspects (e.g., DoS attacks) are discussed as well. Since smart contract functionality was in its early stages of development at that time, not much is said about RSM-layer properties, and little is said about applications beyond cryptocurrencies and data storage. Similarly, Tschorsch and Scheuermann [285] and Yli-Huumo *et al.* [286] present early survey papers that focus mostly on consensus- and network-layer attacks, but they also deal with user privacy. The latter [286] has a particular focus on the publication details of blockchain research until 2016, e.g., the venues and the countries of the authors' institutions. Li *et al.* [287] present a high-level overview of blockchain security threats and incidents, but the categorization is lacking. The authors deal with selfish mining, the DAO hack, BGP hijacking, and eclipse attacks, while all of them are mentioned as individual incidents. Conti *et al.* [288] present an overview of consensus- and network-layer attacks inherent to the Bitcoin blockchain. One interesting contribution is its overview of client-side attacks and attacks on exchange systems. Many attacks presented in this work are supported by evidence of incidents. On the other hand, the authors spent only a little effort on the issues related to the RSM and application layers.

B. Research With Layered or Stacked Categorization

Wang *et al.* [16] are the first to propose a 4-layer model denoted as "*a network implementation stack*." Despite

[55] https://www.keylength.com

proposing the stacked model, the authors do not focus on attacks and countermeasures concerning each of the layers. The main focus of their work is on the consensus layer, where the authors dedicate most of their attention to PoR, PoS, and BFT protocols as well as improving blockchain performance by sharding, side-chains, and non-linear data organization. In the application layer, the authors discuss a few types of emerging blockchain-based applications, such as general-purpose data storage and access control. Saad *et al.* [289] identify three categories of attacks: blockchain structure attacks, peer-to-peer system attacks, and application-oriented attacks. As compared to our work, it mostly holds that their *peer-to-peer system attacks* encompass our network and consensus layer attacks, whereas their *application-oriented attacks* include the RSM and application-layer attacks from our work. In contrast to our work, Saad *et al.* [289] put double-spending attacks into application-oriented attacks, whereas in our case, they are part of the consensus layer since the means for their realization resides in this layer. Moreover, the authors of this article deal with crypto-jacking attacks, which are out-of-the-scope for our reference architecture, as they are not related to the infrastructure of the involved parties we consider (see Section III-A). Chen *et al.* [290] propose a 4-layer model similar to ours, which is used to study vulnerabilities in Ethereum. The authors identify 44 vulnerabilities, 26 attacks, and 47 defenses in total. In contrast to our model, the authors use the "*data*" layer in place of our RSM layer. This leads to a difference in interpretation between their framework and ours: e.g., they consider reentrancy bugs as an application layer vulnerability, whereas we treat them as an RSM-layer vulnerability. Since the authors focus on Ethereum, most vulnerabilities belong to the RSM layer; however, some of the other vulnerabilities (e.g., the BGP hijacking attack against MyEtherWallet [291], [292]) do not seem to be specific for Ethereum. In contrast, our work takes a broader view, and we do not constrain it to a single blockchain. Another stack-based model was proposed by Zhang *et al.* [293] and consists of six layers, where the layers stand for *the application, contract, incentive, consensus, network, and a data layer*. The works of Alkhalifah *et al.* [294] and Zhu *et al.* [295] feature groupings consisting of five (*network, consensus, mining pool, smart contract*, and *client* vulnerabilities) and four (*data privacy, data availability, data integrity*, and *data controllability* attacks) categories, respectively. Natoli *et al.* [296] focus mainly on the consensus layer but include some network-layer attacks as well (e.g., eclipse attacks and BGP hijacking).

C. Research Focusing on a Particular Layer

Finally, there is a number of survey papers that explicitly focus on specific layers: the network layer [297], the RSM layer (in particular smart contracts) [138], [298], [299], protocols of the consensus layer [101], [115], [274], [300], [301], [302], incentives at the consensus layer [303], [304], and application layer from the general standpoint [11], or with a focus on the IoT domain [305], [306], [307], [308].

XIII. Conclusion

In this article, we focused on the systematization of the knowledge about security aspects of blockchain systems, while we aimed to create a standardized model for studying vulnerabilities and security threats. We proposed a stack-modeled security reference architecture (SRA) consisting of four layers, and at each of the layers, we surveyed categories and options for their instantiation with their respective security implications and properties. We modeled particular categories as vulnerability/threat/defense graphs, which we provided as a means for reasoning about imposed security aspects. Next, we collected a sample of blockchain-related incidents that occurred in practice, which we further categorized using our proposed model. We observed that the number of incident types occurred in practice is substantially smaller than the number of described threats, especially in the consensus and application layer. In the case of the application layer, most of the incidents occurred due to exploiting a centralized component by external or internal attackers, while in the case of the consensus layer, most of the incidents occurred due to temporary violation of protocol assumptions by 51% attacks. Finally, we presented a security-oriented methodology for designers of blockchains platforms and applications, respecting the proposed SRA.

Appendix

A. Features of Blockchains

Blockchains were initially introduced as a means of coping with the centralization of monetary assets management, resulting in their most popular application – a decentralized cryptocurrency with native crypto-tokens. However, other blockchain applications have meanwhile started to proliferate as well, benefiting from features other than decentralization. We summarize the inherent and non-inherent features of blockchains in the following.

1) Inherent Features:

Decentralization: is achieved by a distributed consensus protocol – the protocol ensures that each modification of the ledger is a result of interaction among participants. In the consensus protocol, participants are equal, i.e., no single entity is designed as an authority. An important result of decentralization is resilience to node failures.

Censorship Resistance: is achieved due to decentralization, and it ensures that each valid transaction is processed and included in the blockchain.

Immutability: means that the history of the ledger cannot be easily modified – it requires a significant quorum of colluding nodes. The immutability of history is achieved by a cryptographic one-way function (i.e., a hash function) that creates integrity-preserving links between the previous record (i.e., block) and the current one. In this way, integrity-preserving chains (e.g., blockchains) or graphs (e.g., direct acyclic graphs [22], [309], [310] or trees [23]) are built in an append-only fashion. However, the immutability of new blocks is not immediate and depends on the time to the finality of a particular consensus protocol (see Section III-C).

TABLE III
INCIDENTS THAT OCCURRED AT THE APPLICATION LAYER

Acronym / Incident	**Threat / Vulnerability**	**Reference**	**Description**	**Impact ($)**
Gatecoin (May 2016)	Centralized server compromise (single-point-of-failure)	[314]	The exchange was the victim of a man-in-the-middle attack. The malicious external party was involved in this breach, and it managed to alter Gatecoin's system so that deposit transfers bypassed the multisig cold storage and went to hot wallets, which were exploited due to OPSEC issues.	25,160,000
Doge Vault (May 2014)	Centralized server compromise (single-point-of-failure)	[315]	The attacker gained access to the server where Doge Vault's virtual machines were running, providing him with full access to the systems.	56,000
BIPS.me (November 2013)	DDoS + access subversion on centralized server (single-point-of-failure)	[316]	An initial DDoS attack caused vulnerability to the system, which has then enabled the attacker to gain access and compromise several wallets.	554,260
Bitfinex (August 2016)	Centralized server (single-point-of-failure)	[317]	Although Bitfinex used 2-of-3 multisig wallets, two of these keys were owned by Bitfinex (one stored in cold wallet), while the user owned only a single key. It is not clear whether this incident involved insider threat or it was conducted externally.	71,288,416
Bitcoin Central (April 2013)	Centralized server compromise (single-point-of-failure)	[318]	Password was reset from the hosting provider's web interface, enabling the attacker to lock out of the interface and request a reboot of the machine into rescue mode. Next, the attacker has stolen private keys from the hot wallet.	—
Cyber-squatting attacks in NameCoin	Violation of accurate registration	[215]	Cyber-squatters seized identities that do not belong to them nor represent themselves.	—
Front-Running on Ethereum Exchanges	Front-running in Intra-Chain Exchanges	[92]	Arbitrage bots front-run user issued exchange transactions with the ones with the higher fees. Therefore, user issued transactions are discarded or traded with worse exchange rate as intended.	—

Availability: although distributed ledgers are highly redundant in terms of data storage (i.e., full nodes store replicated data), the main advantage of such redundancy is paid off by the extremely high availability of the system. This feature may be of special interest to applications that cannot tolerate outages.

Auditability: correctness of each transaction and block recorded in the blockchain can be validated by any participating node, which is possible due to the publicly-known rules of a consensus protocol.

Transparency: the transactions stored in the blockchain as well as the actions of protocol participants are visible to other participants and in most cases even to the public.

2) Non-Inherent Features: Additionally to the inherent features, blockchains may be equipped with other features that aim to achieve extra goals. Below we list a few examples of such non-inherent features.

Energy Efficiency: running an open distributed ledger often means that scarce resources are wasted (e.g., Proof-of-Work). However, there are available consensus protocols that do not waste scarce resources, but instead emulate the consumption of scarce resources (i.e., Proof-of-Burn), or the interest rate on an investment (i.e., Proof-of-Stake). See examples of these protocols in Section VI.

Scalability: describes how the consensus protocol scales when the number of participants increases. Protocols whose behavior is not negatively affected by an increasing number of participants have high scalability.

Throughput: represents the number of transactions that can be processed per unit of time. Some consensus protocols have only a small throughput (e.g., Proof-of-Work), while others are designed with the intention to maximize throughput (e.g., Byzantine Fault Tolerant (BFT) protocols with a small number of participants). See examples of BFT protocols in Section VI-C.

Privacy & Anonymity: by design, data recorded on a public blockchain is visible to all nodes or public, which may lead to privacy and anonymity issues. Therefore, multiple solutions increasing anonymity (e.g., ring signatures [311] in Monero) and privacy (e.g., zk-SNARKs [312] in Zcash) were proposed in the context of cryptocurrencies, while other efforts have been made in privacy-preserving smart contract platforms [131], [132].

Accountability and Non-Repudiation: if blockchains or applications running on top of them are designed in such a way that identities of nodes (or application users) are known and verified, accountability and non-repudiation of actions performed can be provided too.

B. Atomic Swap Protocols

1) Atomic Swap for Two Parties: Atomic swaps assume two parties $\mathbb{A}$ and $\mathbb{B}$ owning crypto-tokens in two different blockchains. $\mathbb{A}$ and $\mathbb{B}$ wish to execute cross-chain exchange atomically and thus achieve a *fairness* property, i.e., either both of the parties receive the agreed amount of crypto-tokens or

TABLE IV
INCIDENTS AND POSSIBLE VULNERABILITIES AT THE RSM LAYER

Acronym / Incident	**Threat / Vulnerability**	**Reference**	**Description**	**Impact ($)**	**SWC Entry**
DAO Attack (June 2016)	Reentrancy	[319]	A vulnerability in the code allowed a repeated withdrawal of ether, which could be exploited with a malicious fallback function.	70,000,000	SWC-107
King of the Ether Throne (February 2016)	Unchecked return value	[320]	An unchecked return value prevented the rightful compensation of the user.	300	SWC-104
RockPaperScissors	Storing secrets	[321]	A smart contract relied on a secret value that should not be visible to other users.	—	
	Integer overflow and underflow	[322]	The user's input was not properly sanitized and overflow / underflow was possible.		SWC-101
	Unexpected value received	[323]	A smart contract may contain conditions that rely on a certain balance, but the attacker is able to send an unexpected monetary value to the contract and disrupt the intended functionality.	—	SWC-132
	Delegatecall	[138]	Solidity has a feature called delegatecall that enables remote calls of other contracts. Such calls cause an execution of untrusted contracts in the context of the caller, hence all vulnerabilities of the remote code can be exploited.	—	SWC-112
Parity multisig hack (July 2017)	Default function visibility	[324]	Solidity functions are public per default. This can be easily exploited if a developer forgets to make critical functions private.	30,000,000	SWC-100
TheRun (April 2016)	Weak source of randomness	[325]	Relying on the block number or timestamp as a source for randomness is not safe, since it can allow a prediction of the next random number.	—	SWC-120
GovernMental (March 2016)	DoS	[326]	The gas limit or failing calls to external functions can lead to situations where a contract becomes unusable.	11,000	SWC-113, SWC-128
EXTCODESIZE DoS attack (September 2016)	DoS	[327]	The attack exploited a mismatch between the computational cost of some operations and their gas cost.	—	
Rubixi (March 2016)	Unprotected Ether withdrawal	[138]	The access to administration or initialization functions had not been adjusted properly. Missing modifiers or an improperly named function might lead to unauthorized access.	—	SWC-105, SWC-118
Bancor (June 2017)	Front-running / transaction order dependency	[138]	Some contracts rely on the transaction order to make a decision. For example, a winner of a game is chosen from the first transaction with a correct answer. Since the transactions can be inspected by the consensus nodes before they are included, the attacker might steal the answer and make a transaction with a higher fee, which will be included as the first one.	—	SWC-114
	Timestamp dependency	[139]	Some contracts might use the current timestamp to trigger certain events. However, timestamps can be adjusted by malicious consensus nodes.	—	SWC-116
	Write to an arbitrary storage location	[328]	By taking advantage of neighboring addresses in the storage and un-sanitized code, the unauthorized attacker might write to sensitive storage locations.	—	SWC-124
Parity multisig hack 2 (November 2017)	Unprotected selfdestruct call	[329]	The unprotected self destruct functionality can be exploited to destroy contracts (or libraries), and potentially freeze funds in them.	150,000,000	SWC-106

neither of them. First, this process involves an agreement on the amount and exchange rate, and second, the execution of the exchange itself.

In a centralized scenario [313], the approach is to utilize a trusted third party for the execution of the exchange. In contrast to the centralized scenario, blockchains allow us to execute such an exchange without a requirement of the trusted party. The atomic swap protocol [314] enables conditional redemption of the funds in the first blockchain to $\mathbb{B}$ upon revealing of the hash pre-image (i.e., secret) that redeems the

TABLE V
INCIDENTS THAT OCCURRED AT THE CONSENSUS LAYER

Acronym / Incident	**Threat / Vulnerability**	**Reference**	**Description**	**Impact ($)**
Ethereum Classic (January 2019)	51% attack & double spending (violation of assumptions)	[330]	A 51% attack on Ethereum Classic led to a deep chain reorganization, which included double-spending attacks against Binance and Bitrue wallets.	1,100,000
Monacoin (May 2018)	51% attack & double spending (violation of assumptions)	[331]	A block reorganization on Monacoin included double-spending attacks that targeted several "Western exchanges" (particularly Livecoin). Unconfirmed reports on Reddit mention losses of 90,000 USD.	90,000
Bitcoin Gold (May 2018)	51% attack & double spending (violation of assumptions)	[332]	A Block withholding attack on Bitcoin Gold led to 76 double-spent transactions, mostly targeting cryptocurrency exchanges.	18,600,000
Litecoin Cash (May 2018)	51% attack & double spending (violation of assumptions)	[333]	A similar double-spending attack targeted Litecoin cash – the lead developer "Tanner" stated that he believes the mining power may have been rented.	—
Zencash (June 2018)	51% attack & double spending (violation of assumptions)	[334]	A 51% attack led to several block reorganizations, with the largest one that reversed 38 blocks. Only two transactions included double spending, but these were large enough to cause considerable losses.	550,000
Verge (April 2018)	51% attack & semantic bug (violation of assumptions)	[283]	A vulnerability in Verge's difficulty adjustment algorithm was used to amplify a 51% attack.	3,850,000
Verge (May 2018)	51% attack & semantic bug (violation of assumptions)	[335]	An inadequate response by Verge developers to the previous attack led to it being repeated a month later.	1,750,000
Checklocktime, CensorshipCon, GoldfingerCon, etc.	Bribery attacks	[336]	Various bribery attacks have been listed in the literature. In these attacks, it is profitable for consensus nodes to accept bribes for deviation from the protocol.	—

funds on the second blockchain to $\mathbb{A}$. The atomic swap protocol is based on two Hashed Time-Lock Contracts (HTLC) that are deployed by both parties in both blockchains.

Although HTLCs can be realized by Turing-incomplete smart contracts with support for hash-locks and time-locks, for clarity, we provide a description assuming Turing-complete smart contracts, requiring four transactions:

1) $\mathbb{A}$ chooses a random string x (i.e., a secret) and computes its hash $h(x)$. Using $h(x)$, $\mathbb{A}$ deploys $HTLC_{\mathbb{A}}$ on the first blockchain and sends the agreed amount to it, which later enables anybody to do a conditional transfer of that amount to $\mathbb{B}$ upon calling a particular method of $HTLC_{\mathbb{A}}$ with $x = h(x)$ as an argument (i.e., hash-lock). Moreover, $\mathbb{A}$ defines a time-lock, which, when expired, allows $\mathbb{A}$ to recover funds into her address by calling a dedicated method: this is to prevent aborting of the protocol by another party.
2) When $\mathbb{B}$ notices that $HTLC_{\mathbb{A}}$ has been already deployed, she deploys $HTLC_{\mathbb{B}}$ on the second blockchain and sends the agreed amount there, enabling a conditional transfer of that amount to $\mathbb{A}$ upon revealing the correct pre-image of $h(x)$ ($h(x)$ is visible from already deployed $HTLC_{\mathbb{A}}$). $\mathbb{B}$ also defines a time-lock in $HTLC_{\mathbb{B}}$ to handle abortion by $\mathbb{A}$.
3) Once $\mathbb{A}$ notices deployed $HTLC_{\mathbb{B}}$, she calls a method of $HTLC_{\mathbb{B}}$ with revealed x, and in turn, she obtains the funds on the second blockchain.
4) Once $\mathbb{B}$ notices that x was revealed by $\mathbb{A}$ on the second blockchain, she calls a method of $HTLC_{\mathbb{A}}$ with x as an argument, and in turn, she obtains the funds on the first blockchain.

If any of the parties aborts, the counter-party waits until the time-lock expires and redeems the funds.

2) Atomic Swap for Three Parties: In the following, we outline a three-way atomic swap protocol, where party $\mathbb{A}$ wishes to sell an asset a for BTC, the party $\mathbb{B}$ wishes to buy a for ETH, and DEX $\mathbb{E}$ is inter-mediating the asset transfer:

1) $\mathbb{B}$ chooses a random string x (i.e., a secret) and computes its hash $h(x)$. Using $h(x)$, $\mathbb{B}$ deploys $HTLC_{\mathbb{B}}$ on the Ethereum blockchain and sends the agreed ETH amount there, which later enables anybody to do a conditional transfer of that amount to $\mathbb{E}$ upon calling a particular method of $HTLC_{\mathbb{B}}$ with $x = h(x)$ as an argument. Moreover, $\mathbb{B}$ defines a time-lock to handle abortion by any party.
2) Once $\mathbb{E}$ notices that $HTLC_{\mathbb{B}}$ has been already deployed on the Ethereum blockchain, she deploys $HTLC_{\mathbb{E}}$ on the Bitcoin blockchain and sends the agreed BTC amount there, enabling a conditional transfer of that amount to $\mathbb{A}$ upon revealing the correct pre-image of $h(x)$ (which is visible in already deployed $HTLC_{\mathbb{B}}$). $\mathbb{E}$ also defines a time-lock in $HTLC_{\mathbb{E}}$.
3) Once $\mathbb{A}$ notices that $HTLC_{\mathbb{A}}$ has been already deployed on the Bitcoin blockchain, she deploys $HTLC_{\mathbb{A}}$ on the asset blockchain and lock the asset a there, enabling a conditional transfer of a to $\mathbb{B}$ upon revealing the correct pre-image of $h(x)$ (which is visible in already deployed $HTLC_{\mathbb{B}}$ and $HTLC_{\mathbb{E}}$). $\mathbb{A}$ also defines a time-lock in $HTLC_{\mathbb{A}}$.
4) When $\mathbb{B}$ notices that both $HTLC_{\mathbb{E}}$ and $HTLC_{\mathbb{A}}$ have been already correctly deployed, she reveals the secret x

TABLE VI
INCIDENTS THAT OCCURRED AT THE NETWORK LAYER (PUBLIC NETWORKS)

Acronym / Incident	**Threat / Vulnerability**	**Reference**	**Description**	**Impact ($)**
Canadian Bitcoin hijack (February 2013)	BGP prefix hijacking (routing attack & eclipse attack)	[337]	Intercepted data between Bitcoin miners and Bitcoin mining pools. Tricked honest miners were mining on an attacker controlled pool.	83,000
Bitcoin deanonymization (March 2015)	Sybil attack (identity revealing attacks)	[59]	A large number of fake nodes were introduced to deanonymize client traffic.	—
MyEtherWallet hijack (April 2018)	BGP hijacking (routing attack)	[290], [291]	A BGP hijack have been performed against Amazon DNS, leading to the wallet's server domain name being resolved to a Russian phishing site in several cities.	152,000
Electrum DoS (August 2019)	Volumetric DoS	[338]	DoS of legitimate Electrum servers was conducted as an attempt to get connected vulnerable Electrum wallets to a malicious server, which resulted in a loss of wallet funds.	In millions
Bitcoin partitioning (proof-of-concept)	Prefix hijacking (routing attack)	[339]	An attack to partition Bitcoin network by hijacking IP prefixes.	—
Erebus (proof-of-concept)	Malicious ISP attack	[50]	ISP uses its topological advantage to launch a stealthy partition attack.	—
Eclipse attack on Bitcoin (proof-of-concept)	Un-authenticated and unreliable peers	[47]	A view of the network by eclipsed peers is fully under the attacker's control.	—
De-anonymization in Bitcoin with Tor (proof-of-concept)	Abusing Bitcoin DoS protection (identity revealing attack)	[57]	Bitcoin peers were forced to ban Tor exit nodes of attacker's choice by abusing Bitcoin DoS protection. Thus, the attacker was able to control all remaining Tor exit nodes and cause client traffic to pass through them.	—
Suspected Penny flooding on Bitcoin (December 2017)	DoS on a mempool (attacking local resources)	[340]	Flooding a mempool with low fee transactions resulted in clogged up memory of consensus nodes and increased transaction processing fees.	—
2X-Mempool-Attack on Bitcoin (suspected)	Flooding by high-fee transactions (DoS on processing of legitimate transactions)	[341]	Mempool is flooded by high-fee transactions, preventing regular-fee transactions being processed timely.	—

as a part of the transaction sent to $HTLC_{\mathbb{A}}$. This triggers a transfer of asset a to $\mathbb{B}$.

5) Once $\mathbb{A}$ notices that x was revealed, she sends a transaction with x to $HTLC_{\mathbb{E}}$, obtaining BTC from $\mathbb{E}$.

6) Once $\mathbb{E}$ notices that x was revealed, she sends a transaction with x to $HTLC_{\mathbb{B}}$, obtaining ETH from $\mathbb{B}$.

C. Examples of Incidents

In the current section, we list several incidents at each layer of the security reference architecture. In detail, Table VI contains incidents of public networks at the network layer, Table V lists incidents of the consensus layer, Table IV focuses on the RSM layer, and Table III shows a few examples of incidents at the application layer.

ACKNOWLEDGMENT

The authors would like to thank our colleagues Pieter Hartel, Stefanos Leonardos, and Mark van Staalduinen for their valuable feedback.

REFERENCES

[1] (2019). *Enterprise Ethereum Alliance*. [Online]. Available: https://entethalliance.org/

[2] ISO. (2019). *ISO/DTR 23245: Blockchain and Distributed Ledger Technologies—Security Risks, Threats, and Vulnerabilities*. [Online]. Available: https://www.iso.org/standard/75062.html?browse=tc

[3] ISO. (2019). *ISO/CD 23257: Blockchain and Distributed Ledger Technologies—Reference Architecture Vulnerabilities*. [Online]. Available: https://www.iso.org/standard/75093.html?browse=tc

[4] L. Goasduff. (2019). *Gartner Says Blockchain Deployments Across Financial Services Ecosystems Are At Least Three Years Away*. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/09-16-2019-gartner-says-blockchain-deployments-across-financial-services-ecosystems-are-at-least-three-years-away

[5] J. Barry. (2018). *Blockchain Technology Needs Standardization*. [Online]. Available: https://medium.com/blockchain-standards/blockchain-technology-needs-standardization-596fbea2d0cf

[6] H. Zimmermann, "OSI reference model—The ISO model of architecture for open systems interconnection," *IEEE Trans. Commun.*, vol. C-28, no. 4, pp. 425–432, Apr. 1980.

[7] F. Liu *et al.*, *NIST Cloud Computing Reference Architecture*, document SP 500, NIST, Gaithersburg, MD, USA, 2011.

[8] Z. Xiao and Y. Xiao, "Security and privacy in cloud computing," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 2, pp. 843–856, 2nd Quart., 2013.

[9] Common Criteria. (2017). *Common Criteria for Information Technologysecurity Evaluation. Part 1: Introduction and General Model*. [Online]. Available: https://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R5.pdf

[10] I. Homoliak, S. Venugopalan, Q. Hum, and P. Szalachowski, "A security reference architecture for blockchains," in *Proc. 2nd IEEE Int. Conf. Blockchain (Blockchain)*, 2019, pp. 390–397.

[11] F. Casino, T. K. Dasaklis, and C. Patsakis, "A systematic literature review of blockchain-based applications: Current status, classification and open issues," *Telemat. Informat.*, vol. 36, pp. 55–81, Mar. 2019.

[12] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *Int. J. Web Grid Services*, vol. 14, no. 4, pp. 352–375, 2018.

[13] J. F. Wolfswinkel, E. Furtmueller, and C. P. Wilderom, "Using grounded theory as a method for rigorously reviewing literature," *Eur. J. Inf. Syst.*, vol. 22, no. 1, pp. 45–55, 2013.

[14] R. Tahir *et al.*, "Mining on someone else's dime: Mitigating covert mining operations in clouds and enterprises," in *Proc. Int. Symp. Res. Attacks Intrusions Defenses*, 2017, pp. 287–310.

[15] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *Proc. 40th Annu. Symp. Found. Comput. Sci.*, 1999, pp. 120–130.

[16] W. Wang *et al.*, "A survey on consensus mechanisms and mining strategy management in blockchain networks," *IEEE Access*, vol. 7, pp. 22328–22370, 2019.

[17] S. Nakamoto. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[18] P. Szalachowski, D. Reijsbergen, I. Homoliak, and S. Sun, "StrongChain: Transparent and collaborative proof-of-work consensus," in *Proc. 28th USENIX Security Symp. (USENIX Security)*, Santa Clara, CA, USA, Aug. 2019, pp. 819–836.

[19] Hyperledger Team. (2017). *Hyperledger Architecture, Volume 1: Consensus*. [Online]. Available: https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf

[20] L. Chen, L. Xu, N. Shah, Z. Gao, Y. Lu, and W. Shi, "On security analysis of proof-of-elapsed-time (POET)," in *Proc. Int. Symp. Stabilization Safety Security Distrib. Syst.*, 2017, pp. 282–297.

[21] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Proc. CRYPTO*, 2017, pp. 357–388.

[22] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "Spectre: A fast and scalable cryptocurrency protocol," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 1159, 2016.

[23] Y. Sompolinsky and A. Zohar, "Accelerating bitcoin's transaction processing. Fast money grows on trees, not chains," in *Proc. IACR Cryptol. ePrint Archive*, vol. 2013, 2013, p. 881.

[24] W. Tang. (2017). *ECIP-1029: Include Uncles in Total Difficulty Calculation*. [Online]. Available: https://github.com/ethereumproject/ECIPs/pull/71

[25] A. Zamyatin, N. Stifter, P. Schindler, E. Weippl, and W. J. Knottenbelt. (2018). *FLUX: Revisiting Near Blocks for Proof-of-Work Blockchains*. [Online]. Available: https://eprint.iacr.org/2018/415/20180529:172206

[26] M. Castro *et al.*, "Practical Byzantine fault tolerance," in *Proc. OSDI*, 1999, p. 99.

[27] P.-L. Aublin, S. B. Mokhtar, and V. Quéma, "RBFT: Redundant Byzantine fault tolerance," in *Proc. IEEE ICDCS*, 2013, pp. 297–306.

[28] E. Buchman, J. Kwon, and Z. Milosevic, "The latest Gossip on BFT consensus," 2018. [Online]. Available: arxiv.abs/1807.04938.

[29] A. Kiayias and A. Russell, "Ouroboros-BFT: A simple Byzantine fault tolerant consensus protocol," in *Proc. IACR Cryptol. ePrint Arch.*, 2018, p. 1049.

[30] S. Duan, H. Meling, S. Peisert, and H. Zhang, "BChain: Byzantine replication with high throughput and embedded reconfiguration," in *Proc. OPODIS*, 2014, pp. 91–106.

[31] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proc. ACM SOSP*, 2017, pp. 51–68.

[32] P. Daian, R. Pass, and E. Shi, "Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake," in *Proc. Int. Conf. Financial Cryptogr. Data Security*. 2019, pp. 23–41.

[33] D. W. T. Hanke and M. Movahedi, "DFINITY technology overview series, consensus system," 2018. [Online]. Available: arXiv:1805.04548.

[34] *The ZILLIQA Technical Whitepaper*, ZILLIQA Team, Singapore, 2017.

[35] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "OmniLedger: A secure, scale-out, decentralized ledger via sharding," in *Proc. IEEE S&P*, San Francisco, CA, USA, May 2018, pp. 1–8. [Online]. Available: https://doi.org/10.1109/SP.2018.000-5

[36] M. Zamani, M. Movahedi, and M. Raykova, "RapidChain: Scaling blockchain via full sharding," in *Proc. ACM CCS*, 2018, pp. 931–948. [Online]. Available: https://doi.org/10.1145/3243734.3243853

[37] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Comput. Surveys*, vol. 22, no. 4, pp. 299–319, 1990.

[38] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, 1998.

[39] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proc. USENIX ATC*, 2014, pp. 305–319.

[40] B. M. Oki and B. H. Liskov, "Viewstamped replication: A new primary copy method to support highly-available distributed systems," in *Proc. ACM Symp. Principles Distrib. Comput.*, 1988, pp. 22–29.

[41] C. Cachin and J. A. Poritz, "Secure intrusion-tolerant replication on the internet," in *Proc. IEEE Int. Conf. Depend. Syst. Netw.*, 2002, pp. 167–176.

[42] M. L. Collins *et al.*, "Common sense guide to prevention and detection of insider threats 5th edition," in *Proc. CERT*, 2016, pp. 1–9.

[43] D. S. H. Rosenthal, P. Maniatis, M. Roussopoulos, T. J. Giuli, and M. Baker. (2004). *Notes on the Design of an Internet Adversary*. [Online]. Available: http://arxiv.org/abs/cs.DL/0411078.

[44] S. Bradshaw and L. DeNardis. (2016). *The Politicization of the Internet's Domain Name System: Implications for Internet Security, Universality, and Freedom*. [Online]. Available: https://journals.sagepub.com/doi/full/10.1177/1461444816662932

[45] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Routing attacks on cryptocurrencies," in *Proc. IEEE Symp. Security Privacy (SP)*, San Jose, CA, USA, May 2017, pp. 375–392.

[46] M. Apostolaki, G. Marti, J. Müller, and L. Vanbever, "SABRE: Protecting bitcoin against routing attacks," in *Proc. NDSS*, 2019, pp. 1–8.

[47] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *Proc. USENIX Security*, 2015, pp. 129–144.

[48] K. Wüst and A. Gervais, "Ethereum eclipse attacks," ETH Zürich, Zurich, Switerland, Rep. ethz-a-010724205, 2016.

[49] Y. Marcus, E. Heilman, and S. Goldberg, "Low-resource eclipse attacks on Ethereum's peer-to-peer network," in *Proc. IACR Cryptol. ePrint Arch.*, 2018, p. 236.

[50] M. Tran, I. Choi, G. J. Moon, A. V. Vu, and M. S. Kang, "A stealthier partitioning attack against bitcoin peer-to-peer network," in *Proc. IEEE Symp. Security Privacy (S&P)*, 2020, pp. 894–909.

[51] B. Alangot, D. Reijsbergen, S. Venugopalan, and P. Szalachowski, "Decentralized lightweight detection of eclipse attacks on bitcoin clients," in *Proc. IEEE Int. Conf. Blockchain (Blockchain)*, Greece, Rhodes Island, Nov. 2020, p. 1.

[52] S. Higgins. (2015). *Bitcoin Mining Pools Targeted in Wave of DDoS Attacks*. [Online]. Available: https://www.coindesk.com/bitcoin-mining-pools-ddos-attacks

[53] D. Shares. (2017). *Major DDoS Attacks Hit Bitcoin.Com*. [Online]. Available: https://news.bitcoin.com/ddos-attacks-bitcoin-com-uncensored-information/

[54] E. Arazi. (2018). *Choosing the Right DDoS Solution (Part 4): Hybrid Protection*. [Online]. Available: https://blog.radware.com/security/2018/04/choosing-the-right-ddos-solution-hybrid-protection/

[55] S. D. Lerner. (2019). *New DoS Vuln by Forcing Continuous Hard Disk Seek/Read Activity (Fixed in 0.8.0)*. [Online]. Available: https://bitcointalk.org/index.php?topic=144122.0

[56] BitcoinWiki. *Weaknesses*. Accessed: Jul. 24, 2018. [Online]. Available: https://en.bitcoin.it/wiki/Weaknesses#Denial_of_Service_.28DoS.29_attacks

[57] A. Biryukov and I. Pustogarov, "Bitcoin over tor isn't a good idea," in *Proc. IEEE Symp. Security Privacy (SP)*, San Jose, CA, USA, May 2015, pp. 122–134. [Online]. Available: https://doi.org/10.1109/SP.2015.15

[58] A. Miller *et al.*, *Discovering Bitcoin's Public Topology and Influential Nodes*, Univ. Maryland, College Park, MD, USA, 2015.

[59] G. Caffyn. (2015). *Chainalysis CEO Denies Sybil Attack on Bitcoin Network*. [Online]. Available: https://www.coindesk.com/chainalysis-ceo-denies-launching-sybil-attack-on-bitcoin-network

[60] A. Miller, A. Kosba, J. Katz, and E. Shi, "Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions," in *Proc. ACM CCS*, 2015, pp. 680–691.

[61] V. Buterin and V. Griffith, "Casper the friendly finality gadget," 2017. [Online]. Available: arxiv.abs/1710.09437.

[62] X. Yang, Y. Chen, and X. Chen, "Effective scheme against 51% attack on proof-of-work blockchain with history weighted information," in *Proc. IEEE Int. Conf. Blockchain (Blockchain)*, 2019, pp. 261–265.

[63] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of BFT protocols," in *Proc. ACM CCS*, 2016, pp. 31–42.

[64] A. Boverman. (2011). *Timejacking & Bitcoin*. [Online]. Available: http://culubas.blogspot.com/2011/05/timejacking-bitcoin_802.html

[65] P. Szalachowski, "(Short paper) towards more reliable bitcoin timestamps," in *Proc. IEEE CVCBT*, 2018, pp. 1–5.

[66] S. Dexter. (2018). *1% Shard Attack Explained—Ethereum Sharding (Contd.)*. [Online]. Available: https://www.mangoresearch.co/1-shard-attack-explained-ethereum-sharding-contd/

[67] M. Bjelic. *On the Probability of 1% Attack*. Accessed: Aug. 16, 2019. [Online]. Available: https://ethresear.ch/t/on-the-probability-of-1-attack/4009/6

[68] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 17–30.

[69] E. Syta *et al.*, "Scalable bias-resistant distributed randomness," in *Proc. IEEE Symp. Security Privacy (SP)*, 2017, pp. 444–460.

[70] A. Sonnino, S. Bano, M. Al-Bassam, and G. Danezis, "Replay attacks and defenses against cross-shard consensus in sharded distributed ledgers," 2019. [Online]. Available: arxiv.abs/1901.11218.

[71] J. Long and R. Wei, "Scalable BFT consensus mechanism through aggregated signature Gossip," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC)*, 2019, pp. 360–367.

[72] F. Silva, A. Alonso, J. Pereira, and R. Oliveira, "A comparison of message exchange patterns in BFT protocols," in *Proc. IFIP Int. Conf. Distrib. Appl. Interoperable Syst.*, 2020, pp. 104–120.

[73] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak, "Proofs of space," in *Proc. CRYPTO*, 2015, pp. 1–25.

[74] S. Park, K. Pietrzak, J. Alwen, G. Fuchsbauer, and P. Gazi. (2015). *SpaceCoin: A Cryptocurrency Based on Proofs of Space*. [Online]. Available: https://eprint.iacr.org/2015/528.pdf

[75] T. Hønsi, "Spacemint—A cryptocurrency based on proofs of space," M.S. thesis, Dept. Comput. Sci., NTNU, Trondheim, Norway, 2017.

[76] K. Karantias, A. Kiayias, and D. Zindros. (2019). *Proof-of-Burn*. [Online]. Available: https://eprint.iacr.org/2019/1096.pdf

[77] P4Titan. (2014). *SlimCoin: A Peer-to-Peer Crypto-Currency With Proof-of-Burn*. [Online]. Available: ttps://github.com/slimcoin-project/slimcoin-project.github.io/blob/master/whitepaperSLM.pdf

[78] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," in *Proc. IEEE S&P*, 2014, pp. 475–490.

[79] "FileCoin: A decentralized storage network," Protocol Labs, San Francisco, CA, USA, Rep., 2017. [Online]. Available: https://filecoin.io/filecoin.pdf

[80] S. King and S. Nadal. (2012). *PPcoin: Peer-to-Peer Crypto-Currency With Proof-of-Stake*. [Online]. Available: https://peercoin.net/assets/paper/peercoin-paper.pdf

[81] R. Zhang and B. Preneel, "Lay down the common metrics: Evaluating proof-of-work consensus protocols' security," in *Proc. IEEE S&P*, 2019, pp. 175–192.

[82] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," *Commun. ACM*, vol. 61, no. 7, pp. 95–102, 2018.

[83] J. Brown-Cohen, A. Narayanan, A. Psomas, and S. M. Weinberg, "Formal barriers to longest-chain proof-of-stake protocols," in *Proc. ACM Conf. Econ. Comput.*, 2019, pp. 459–473.

[84] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal selfish mining strategies in Bitcoin," in *Proc. FC*, 2016, pp. 515–532.

[85] K. Nayak, S. Kumar, A. Miller, and E. Shi, "Stubborn mining: Generalizing selfish mining and combining with an Eclipse attack," in *Proc. IEEE EuroSP*, 2016, pp. 305–320.

[86] R. Pass and E. Shi, "FruitChains: A fair blockchain," in *Proc. PODC*, 2017, pp. 315–324.

[87] R. Zhang and B. Preneel, "Publish or perish: A backward-compatible defense against selfish mining in bitcoin," in *Proc. CT-RSA*, 2017, pp. 277–292.

[88] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *Proc. USENIX Security*, 2016, pp. 279–296.

[89] A. Miller. (2013). *Feather-Forks: Enforcing a Blacklist With Sub—50% Hash Power*. [Online]. Available: https://bitcointalk.org/index.php?topic=312668.0

[90] P. R. Rizun, "Subchains: A technique to scale Bitcoin and improve the user experience," *Ledger*, to be published.

[91] J. Bonneau, E. W. Felten, S. Goldfeder, J. A. Kroll, and A. Narayanan, "Why buy when you can rent? Bribery attacks on bitcoin consensus," in *Proc. 3rd Workshop Bitcoin Blockchain Res. (BITCOIN)*, 2016, pp. 1–8.

[92] P. Daian *et al.*, "Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability," in *Proc. IEEE Symp. Security Privacy (SP)*, 2020, pp. 566–583.

[93] M. Rosenfeld. (2011). *Analysis of Bitcoin Pooled Mining Reward Systems*. [Online]. Available: http://arxiv.org/abs/1112.4980.

[94] Raulo. (2011). *Optimal Pool Abuse Strategy*. [Online]. Available: http://bitcoin.atspace.com/poolcheating.pdf

[95] O. Schrijvers, J. Bonneau, D. Boneh, and T. Roughgarden, "Incentive compatibility of bitcoin mining pool reward functions," in *Proc. Financial Crypto*, 2016, pp. 477–498.

[96] S. Bag, S. Ruj, and K. Sakurai, "Bitcoin block withholding attack: Analysis and mitigation," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 8, pp. 1967–1978, Aug. 2017.

[97] S. Bag and K. Sakurai, "Yet another note on block withholding attack on bitcoin mining pools," in *Proc. Int. Conf. Inf. Security*, 2016, pp. 167–170.

[98] P2Pool.org. (2017). *P2Pool—Decentralized Bitcoin Mining Pool*. [Online]. Available: http://p2pool.org/

[99] A. Bessani, J. Sousa, and E. E. Alchieri, "State machine replication for the masses with BFT-smart," in *Proc. IEEE/IFIP DSN*, 2014, pp. 355–362.

[100] C. Cachin, "Yet another visit to paxos," IBM Res., Zürich, Switzerland, Rep. RZ3754, 2009.

[101] C. Cachin and M. Vukolić, "Blockchain consensus protocols in the wild," in *Proc. DISC*, 2017, pp. 1–16.

[102] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: BFT consensus with linearity and responsiveness," in *Proc. ACM Symp. Principles Distrib. Comput.*, 2019, pp. 347–356.

[103] M. Franklin, "A survey of key evolving cryptosystems," *Int. J. Security Netw.*, vol. 1, nos. 1–2, pp. 46–53, 2006.

[104] M. Bellare and S. K. Miner, "A forward-secure digital signature scheme," in *Proc. CRYPTO*, 1999, pp. 116–129.

[105] P. Gaži, A. Kiayias, and A. Russell, "Stake-bleeding attacks on proof-of-stake blockchains," in *Proc. IEEE CVCBT*, 2018, pp. 85–92.

[106] M. Drijvers, S. Gorbunov, G. Neven, and H. Wee, "Pixel: Multi-signatures for consensus," in *Proc. 29th {USENIX} Security Symp. ({USENIX} Security)*, 2020, pp. 2093–2110.

[107] QuantumMechanic. (2011). *Proof of Stake Instead of Proof of Work*. [Online]. Available: https://bitcointalk.org/index.php?topic=27787.0

[108] I. Bentov, A. Gabizon, and A. Mizrahi, "Cryptocurrencies without proof of work," in *Proc. FC*, 2016, pp. 142–157.

[109] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, "Proof of activity: Extending bitcoin's proof of work via proof of stake," in *Proc. ACM SIGMETRICS*, 2014, p. 452.

[110] I. Bentov, R. Pass, and E. Shi, "Snow white: Provably secure proofs of stake," in *Proc. IACR Cryptol. ePrint Arch.*, 2016, p. 919.

[111] D. Karakostas and A. Kiayias, "Securing proof-of-work ledgers via checkpointing." in *Proc. IACR Cryptol. ePrint Archive*, vol. 2020, 2020, p. 173.

[112] W. Li, S. Andreina, J.-M. Bohli, and G. Karame, "Securing proof-of-stake blockchain protocols," in *Proc. DPM*, 2017, pp. 297–315.

[113] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Proc. EUROCRYPT*, 2018, pp. 66–98.

[114] V. Buterin, *Long-Range Attacks: The Serious Problem With Adaptive Proof of Work*, Ethereum, Zug, Switzerland, May 2014.

[115] S. Bano *et al.*, "SoK: Consensus in the age of blockchains," in *Proc. 1st ACM Conf. Adv. Financ. Technol. (AFT)*, Zurich, Switzerland, Oct. 2019, pp. 183–198.

[116] Q. Feng, D. He, S. Zeadally, M. K. Khan, and N. Kumar, "A survey on privacy protection in blockchain system," *J. Netw. Comput. Appl.*, vol. 126, pp. 45–58, Jan. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804518303485

[117] A. Biryukov, D. Khovratovich, and I. Pustogarov, "Deanonymisation of clients in bitcoin P2P network," in *Proc. ACM CCS*, 2014, pp. 15–29.

[118] I. Pustogarov, "Deanonymisation techniques for Tor and bitcoin," Ph.D. dissertation, Comput. Sci., Univ. Luxembourg, Luxembourg City, Luxembourg, 2015.

[119] G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn, "An empirical analysis of anonymity in zcash," in *Proc. USENIX Security*, 2018, pp. 463–477.

[120] M. Möser *et al.*, "An empirical analysis of traceability in the Monero blockchain," in *Proc. PETS*, vol. 2018, 2018, p. 3.

[121] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, and E. W. Felten, "MixCoin: Anonymity for bitcoin with accountable mixes," in *Proc. FC*, 2014, pp. 486–504.

[122] L. Valenta and B. Rowan, "BlindCoin: Blinded, accountable mixes for bitcoin," in *Financial Crypto*, M. Brenner, N. Christin, B. Johnson, and K. Rohloff, Eds. Berlin, Germany: Springer, 2015.

[123] G. Maxwell, "CoinJoin: Bitcoin privacy for the real world," in *Proc. Bitcoin Forum*, 2013, p. 9.

[124] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "CoinShuffle: Practical decentralized coin mixing for bitcoin," in *Proc. ESORICS*, 2014, pp. 345–364.

[125] J. H. Ziegeldorf, R. Matzutt, M. Henze, F. Grossmann, and K. Wehrle, "Secure and anonymous decentralized bitcoin mixing," *Future Gener.*

Comput. Syst., vol. 80, pp. 448–466, Mar. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X16301297

[126] S. Noether, "Ring signature confidential transactions for monero," in *Proc. IACR Cryptol. ePrint Arch.*, 2015, p. 1098.

[127] I. Miers, C. Garman, M. Green, and A. D. Rubin, "ZeroCoin: Anonymous distributed e-cash from bitcoin," in *Proc. IEEE Symp. Security Privacy*, 2013, pp. 397–411.

[128] E. B. Sasson *et al.*, "ZeroCash: Decentralized anonymous payments from bitcoin," in *Proc. IEEE S&P*, 2014, pp. 459–474.

[129] E. Heilman, F. Baldimtsi, and S. Goldberg, "Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions," in *Proc. FC*, 2016, pp. 43–60.

[130] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "BulletProofs: Short proofs for confidential transactions and more," in *Proc. IEEE Symp. Security Privacy (SP)*, 2018, pp. 315–334.

[131] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. IEEE S&P*, 2016, p. 10.

[132] R. Cheng *et al.*, "EKIDEN: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *Proc. IEEE Eur. Symp. Security Privacy (EuroS&P)*, 2019, pp. 185–200.

[133] G. Zyskind, O. Nathan, and A. Pentland, "ENIGMA: Decentralized computation platform with guaranteed privacy," 2015. [Online]. Available: arxiv/abs/1506.03471.

[134] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "ZETHER: Towards privacy in a smart contract world," in *Proc. IACR Cryptol. ePrint Archive*, vol. 2019, 2019, p. 191. [Online]. Available: https://eprint.iacr.org/2019/191

[135] S.-F. Sun, M. H. Au, J. K. Liu, and T. H. Yuen, "RingCT 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero," in *Proc. Eur. Symp. Res. Comput. Security*, 2017, pp. 456–474.

[136] Zeppelin Solutions. (2017). *Serpent Compiler Audit*. [Online]. Available: https://docs.google.com/document/d/1_PqXuAkvgUAOG3jbBvaUvqN6W90eJ3N4IdTLNMRAijo/edit#heading=h.pe41jxc4c6xs

[137] F. Schrans, S. Eisenbach, and S. Drossopoulou, "Writing safe smart contracts in Flint," in *Proc. Conf. Companion 2nd Int. Art Sci. Eng. Program.*, Nice, France, Apr. 2018, pp. 218–219. [Online]. Available: https://doi.org/10.1145/3191697.3213790

[138] L. Valenta and B. Rowan, "Blindcoin: Blinded, accountable mixes for bitcoin," in *Proc. Financial Cryptogr. Data Security (FC) Int. Workshops BITCOIN WAHC Wearable* (Lecture Notes in Computer Science), vol. 8976. San Juan, Puerto Rico, Jan. 2015, pp. 112–126.

[139] A. Manning. (2018). *Solidity Security: Comprehensive List of Known Attack Vectors and Common Anti-Patterns*. [Online]. Available: shttps://blog.sigmaprime.io/solidity-security.html

[140] Aion Foundation. *List of Historical Vulnerabilities*. Accessed: Aug. 16, 2019. [Online]. Available: https://learn.aion.network/docs/list-of-historical-big-vulnerabilities

[141] SmartContractSecurity. (2019). *Smart Contract Weakness Classification Registry*. [Online]. Available: https://github.com/SmartContractSecurity/SWC-registry/

[142] Trailofbits. *Awesome Ethereum Security*. Accessed: Aug. 11, 2018. [Online]. Available: https://github.com/trailofbits/awesome-ethereum-security

[143] R. M. Parizi, A. Dehghantanha, K.-K. R. Choo, and A. Singh, "Empirical vulnerability analysis of automated smart contracts security testing on blockchains," in *Proc. 28th Annu. Int. Conf. Comput. Sci. Softw. Eng.*, 2018, pp. 103–113.

[144] Consensys. (2016). *Security Tools*. [Online]. Available: https://consensys.github.io/smart-contract-best-practices/security_tools/

[145] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "SmartCheck: Static analysis of ethereum smart contracts," in *Proc. IEEE/ACM 1st Int. Workshop Emerg. Trends Softw. Eng. Blockchain (WETSEB)*, 2018, pp. 9–16.

[146] R. Dua. (Feb. 11, 2019). *Solium Documentation Release 1.0.0*. [Online]. Available: https://media.readthedocs.org/pdf/solium/latest/solium.pdf

[147] J. Chang, B. Gao, H. Xiao, J. Sun, Y. Cai, and Z. Yang, "sCompile: Critical path identification and analysis for smart contracts," in *Proc. Formal Methods Softw. Eng. 21th Int. Conf. Formal Eng. Methods (ICFEM)*, Shenzhen, China, Nov. 2019, pp. 286–304.

[148] P. Hartel and M. van Staalduinen, "Truffle tests for free – replaying Ethereum smart contracts for transparency," iTRust, Singapore Univ. Technol. Design, Singapore, Rep. 1907.09208, Jul. 2019. [Online]. Available: https://arxiv.org/abs/1907.09208.

[149] M. Sutton, A. Greene, and P. Amini, *Fuzzing: Brute Force Vulnerability Discovery*. Upper Saddle River, NJ, USA: Pearson Educ., 2007.

[150] B. Jiang, Y. Liu, and W. Chan, "ContractFuzzer: Fuzzing smart contracts for vulnerability detection," in *Proc. 33rd ACM/IEEE Int. Conf. Autom. Softw. Eng.*, 2018, pp. 259–269.

[151] V. Wüstholz and M. Christakis. (2019). *Harvey: A Greybox Fuzzer for Smart contracts*. [Online]. Available: http://arxiv.org/abs/1905.06944.

[152] J. C. King, "Symbolic execution and program testing," *Commun. ACM*, vol. 19, no. 7, pp. 385–394, 1976. [Online]. Available: https://doi.org/10.1145/360248.360252

[153] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proc. ACM CCS*, 2018, pp. 67–82.

[154] Trailofbits. (2019). *Manticore Documentation Release 0.1.0*. [Online]. Available: https://media.readthedocs.org/pdf/manticore/latest/manticore.pdf

[155] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proc. ACM CCS*, 2016, pp. 254–269.

[156] C. F. Torres, J. Schütte, and R. State, "OSIRIS: Hunting for integer bugs in Ethereum smart contracts," in *Proc. 34th Annu. Comput. Security Appl. Conf. (ACSAC)* San Juan, PR, USA, Dec. 2018, pp. 664–676. [Online]. Available: https://doi.org/10.1145/3274694.3274737

[157] A. Mavridou and A. Laszka, "Designing secure ethereum smart contracts: A finite state machine based approach," in *Proc. Financ. Cryptography Data Security 22nd Int. Conf. (FC)*, Nieuwpoort, Curaçao, Feb./Mar. 2018, pp. 523–540. [Online]. Available: https://doi.org/10.1007/978-3-662-58387-6_28

[158] E. Hildenbrandt *et al.*, "KEVM: A complete semantics of the Ethereum virtual machine," in *Proc. CSF*, 2018, pp. 204–217.

[159] T. Abdellatif and K. Brousmiche, "Formal verification of smart contracts based on users and blockchain behaviors models," in *Proc. 9th IFIP Int. Conf. New Technol. Mobility Security (NTMS)*, Paris, France, Feb. 2018, pp. 1–5. [Online]. Available: https://doi.org/10.1109/NTMS.2018.8328737

[160] Y. Murray and D. A. Anisi, "Survey of formal verification methods for smart contracts on blockchain," in *Proc. 10th IFIP Int. Conf. New Technol. Mobility Security (NTMS)*, Jun. 2019, pp. 1–6. [Online]. Available: https://doi.org/10.1109/NTMS.2019.8763832

[161] X. Bai, Z. Cheng, Z. Duan, and K. Hu, "Formal modeling and verification of smart contracts," in *Proc. 7th Int. Conf. Softw. Comput. Appl. (ICSCA)*, Feb. 2018, pp. 322–326. [Online]. Available: https://doi.org/10.1145/3185089.3185138

[162] Z. Nehai, P. Piriou, and F. F. Daumas, "Model-checking of smart contracts," in *Proc. IEEE Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber Phys. Soc. Comput. (CPSCom) IEEE Smart Data (SmartData)*, Jul./Aug. 2018, pp. 980–987. [Online]. Available: https://doi.org/10.1109/Cybermatics_2018.2018.00185

[163] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "ZEUS: Analyzing safety of smart contracts," in *Proc. 25th Annu. Netw. Distrib. Syst. Security Symp. (NDSS)*, San Diego, CA, USA, Feb. 2018, pp. 1–9. [Online]. Available: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_09–1_Kalra_paper.pdf

[164] I. Grishchenko, M. Maffei, and C. Schneidewind, "A semantic framework for the security analysis of Ethereum smart contracts," in *Proc. Principles Security Trust 7th Int. Conf. (POST) Eur. Joint Conf. Theory Practice Softw. (ETAPS)*, Thessaloniki, Greece, Apr. 2018, pp. 243–269.

[165] Y. Zhou, D. Kumar, S. Bakshi, J. Mason, A. Miller, and M. Bailey, "ERAYs: Reverse engineering Ethereum's opaque smart contracts," in *Proc. USENIX Security*, 2018, pp. 1371–1385.

[166] M. Suiche, "Porosity: A decompiler for blockchain-based smart contracts bytecode," in *Proc. DEF CON*, vol. 25, 2017, p. 11.

[167] I. Nikolić, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, "Finding the greedy, prodigal, and suicidal contracts at scale," in *Proc. 34th Annu. Comput. Security Appl. Conf. (ACSAC)*, San Juan, PR, USA, Dec. 2018, pp. 653–663.

[168] N. Grech *et al.*, "MADMAX: surviving out-of-gas conditions in Ethereum smart contracts," in *Proc. ACM Program. Lang.*, vol. 2, 2018, pp. 1–27. [Online]. Available: https://doi.org/10.1145/3276486

[169] L. Brent *et al.* (2018). *Vandal: A Scalable Security Analysis Framework for Smart Contracts*. [Online]. Available: http://arxiv.org/abs/1809.03981.

[170] J. Krupp and C. Rossow, "TEETHER: Gnawing at Ethereum to automatically exploit smart contracts," in *Proc. USENIX Security*, 2018, p. 9.

[171] S. Popejoy. (2016). *The Pact Smart Contract Language*. [Online]. Available: http://kadena.io/docs/Kadena-PactWhitepaper.pdf

[172] I. Sergey, V. Nagaraj, J. Johannsen, A. Kumar, A. Trunov, and K. C. G. Hao, "Safer smart contract programming with scilla," *Proc. ACM Program. Lang.*, vol. 3, 2019, pp. 1–30. [Online]. Available: https://doi.org/10.1145/3360611

[173] R. Klomp and A. Bracciali, "On symbolic verification of Bitcoin's script language," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology (ESORICS)* (Lecture Notes in Computer Science), J. Garcia-Alfaro, J. Herrera-JoancomartÃ■, G. Livraga, R. Rios, Eds. Cham, Switzerland: Springer, Sep. 2018, pp. 38–56.

[174] R. O'Connor, "Simplicity: A new language for blockchains," in *Proc. Workshop Program. Lang. Anal. Security (PLAS@CCS)*, Dallas, TX, USA, Oct. 2017, pp. 107–120. [Online]. Available: https://doi.org/10.1145/3139337.3139340

[175] MME. (2018). *Conceptual Framework for Legal and Risk Assessment of Crypto Tokens*. [Online]. Available: https://www.mme.ch/fileadmin/files/documents/180501_BCP_Framework_for_Assessment_of_Crypto_Tokens_-_Block_2.pdf

[176] S. Eskandari, J. Clark, D. Barrera, and E. Stobert, "A first look at the usability of bitcoin key management," 2018. [Online]. Available: arxiv.abs/1802.04351.

[177] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "SoK: Research perspectives and challenges for bitcoin and cryptocurrencies," in *Proc. IEEE S&P*, 2015, pp. 104–121.

[178] I. Homoliak, D. Breitenbacher, O. Hujnak, P. Hartel, A. Binder, and P. Szalachowski, "SmartOTPs: An air-gapped 2-factor authentication for smart-contract wallets," in *Proc. AFT*, 2020, pp. 145–162.

[179] Wolfie Zhao. (2018). *Bithumb $31 Million Crypto Exchange Hack: What We Know (And Don't)*. [Online]. Available: https://www.coindesk.com/bithumb-exchanges-31-million-hack-know-dont-know/

[180] Rachel Abrams and Nathaniel Popper. (2014). *Trading Site Failure Stirs Ire and Hope for Bitcoin*. [Online]. Available: https://dealbook.nytimes.com/2014/02/25/trading-site-failure-stirs-ire-and-hope-for-bitcoin/

[181] Reuters. (2016). *Bitcoin Worth $72M was Stolen in Bitfinex Exchange Hack in Hong Kong*. [Online]. Available: http://fortune.com/2016/08/03/bitcoin-stolen-bitfinex-hack-hong-kong/

[182] Dell SecureWorks. (2015). *Cryptocurrency-Stealing Malware Landscape Dell SecureWorks*. [Online]. Available: http://www.opensource.im/cryptocurrency/cryptocurrency-stealing-malware-landscape-dell-secureworks.php

[183] A. Peyton. (2017). *Cyren Sounds Siren over Bitcoin Siphon Scam FinTech Futureless*. [Online]. Available: https://www.bankingtech.com/2017/01/cyren-sounds-siren-over-bitcoin-siphon-scam/

[184] S. Goldfeder *et al.* (2015). *Securing Bitcoin Wallets Via a New DSA/ECDSA Threshold Signature Scheme*. 2015. [Online]. Available: http://stevengoldfeder.com/papers/threshold_sigs.pdf

[185] M. Westerkamp and J. Eberhardt, "zkRelay: Facilitating sidechains using zkSNARK-based chain-relays," *Contract*, vol. 1, no. 2, p. 3, 2020.

[186] M. Herlihy, "Atomic cross-chain swaps," in *Proc. ACM Symp. Principles Distrib. Comput.*, 2018, pp. 245–254.

[187] W. Warren and A. Bandeali. (2017). *0x: An Open Protocol for Decentralized Exchange on the Ethereum Blockchain*. [Online]. Available: https://github.com/0xProject/whitepaper

[188] H. Berg *et al.*, "Hanson's automated market maker," *J. Prediction Markets*, vol. 3, no. 1, pp. 45–59, 2009.

[189] N. Johnson. (2016). *EULER: The Simplest Exchange and Token Currency*. [Online]. Available: https://www.reddit.com/r/ethereum/comments/54l32y/euler_the_simplest_exchange_and_currency/

[190] E. Hertzog, G. Benartzi, and G. Benartzi. (2017). *Bancor Protocol: A Hierarchical Monetary System and the Foundation of a Global Decentralized Autonomous Exchange*. [Online]. Available: https://www.regulus.com/wp-content/uploads/2018/11/BancorProtocolWhitepaperPublicDraftv0.7.pdf

[191] A. Zamyatin *et al.*, "SoK: Communication across distributed ledgers," in *Proc. IACR Cryptol. ePrint Archive*, vol. 2019, 2019, p. 1128.

[192] K. Oosthoek and C. Doerr, "From hodl to heist: Analysis of cyber security threats to bitcoin exchanges," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC)*, 2020, pp. 1–9.

[193] I. Bentov *et al.*, "TesserAct: Real-time cryptocurrency exchange using trusted hardware," in *Proc. IACR Cryptol. ePrint Archive*, vol. 2017, 2017, p. 1153.

[194] J. Poon and T. Dryja. (2016). *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*. [Online]. Available: https://coinrivet.com/research/papers/the-bitcoin-lightning-network-scalable-off-chain-instant-payments/.

[195] G. Avarikioti, F. Laufenberg, J. Sliwinski, Y. Wang, and R. Wattenhofer, "Towards secure and efficient payment channels," 2018. [Online]. Available: arxiv.abs/1811.12740.

[196] P. McCorry, S. Bakshi, I. Bentov, A. Miller, and S. Meiklejohn. (2018). *PISA: Arbitration Outsourcing for State Channels*. [Online]. Available: https://eprint.iacr.org/2018/582

[197] B. Liu, P. Szalachowski, and S. Sun, "Fail-safe watchtowers and short-lived assertions for payment channels," in *Proc. 15th ACM ASIA Conf. Comput. Commun. Security (AsiaCCS)*, 2020, p. 9.

[198] P. Szalachowski, "PADVA: A blockchain-based TLS notary service," in *Proc. 25th IEEE Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Tianjin, China, Dec. 2019, pp. 836–843. [Online]. Available: https://doi.org/10.1109/ICPADS47876.2019.00124

[199] J. Guarnizo and P. Szalachowski, "PDFS: Practical data feed service for smart contracts," in *Proc. 24th Eur. Symp. Res. Comput. Security Comput. Security (ESORICS)*, Sep. 2019, pp. 767–789.

[200] S. Ellis, A. Juels, and S. Nazarov. (2017). *ChainLink: A Decentralized Oracle Network*. [Online]. Available: https://link.smartcontract.com/whitepaper

[201] J. Peterson and J. Krug, "AUGUR: A decentralized, open-source platform for prediction markets," 2015. [Online]. Available: arxiv.abs/1501.01042.

[202] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town Crier: An authenticated data feed for smart contracts," in *Proc. ACM CCS*, 2016, pp. 1–8.

[203] A. S. de Pedro, D. Levi, and L. I. Cuende, "WITNET: A decentralized oracle network protocol," 2017. [Online]. Available: arXiv:1711.09756.

[204] S. A. Crosby and D. S. Wallach, "Efficient data structures for tamper-evident logging." in *Proc. USENIX Security Symp.*, 2009, pp. 317–334.

[205] Proovable Team. (2019). *The Provable Blockchain Oracle for Modern DApps*. [Online]. Available: https://www.oraclize.it/

[206] S. Sen and J. Wang. (Apr. 2004). *Analyzing Peer-to-Peer Traffic Across Large Networks*. [Online]. Available: https://doi.org/10.1109/TNET.2004.826277

[207] H. Kopp, C. Bösch, and F. Kargl, "KopperCoin—A distributed file storage with financial incentives," in *Proc. Int. Conf. Inf. Security Pract. Exp.*, 2016, pp. 79–93.

[208] J. Benet, D. Dalrymple, and N. Greco, *Proof of Replication*, Protocol Labs, San Francisco, CA, USA, Jul. 2017.

[209] C. Fromknecht, D. Velicanu, and S. Yakoubov, "A decentralized public key infrastructure with identity retention," in *Proc. IACR Cryptol. ePrint Archive*, vol. 2014, 2014, p. 803.

[210] Z. Wilcox-O'Hearn. (2003). *Names: Decentralized, Secure, Human-Meaningful: Choose Two*. [Online]. Available: https://web.archive.org/web/20011020191610/http://zooko.com/distnames.html

[211] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, "Bootstrapping trust in distributed systems with blockchains," *USENIX Mag.*, vol. 41, no. 3, pp. 1–7, 2016.

[212] C. Lundkvist, R. Heck, J. Torstensson, Z. Mitton, and M. Sena. (2016). *uPort: A Platform for Self-Sovereign Identity*. [Online]. Available: http://blockchainlab.com/pdf/uPort_whitepaper_DRAFT20161020.pdf

[213] Shocard Inc. (2017). *Shocard: Identity Management Verified Using the Blockchain*. [Online]. Available: http://shocard.com/wp-content/uploads/2018/01/ShoCard-Whitepaper-Dec13–2.pdf

[214] A. Tobin and D. Reed. (2016). *The Inevitable Rise of Self-Sovereign Identity*. [Online]. Available: https://sovrin.org/wp-content/uploads/2018/03/The-Inevitable-Rise-of-Self-Sovereign-Identity.pdf

[215] W3C Community. (2019). *Decentralized Identifiers (DIDs)*. [Online]. Available: https://w3c-ccg.github.io/did-spec/

[216] H. A. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. Narayanan, "An empirical study of Namecoin and lessons for decentralized namespace design," in *Proc. WEIS*, 2015, pp. 1–21.

[217] J. Clark and A. Essex, "CommitCoin: Carbon dating commitments with bitcoin," in *Proc. Int. Conf. Financ. Cryptography Data Security*, 2012, pp. 390–398.

[218] B. Gipp, N. Meuschke, and A. Gernandt, "Decentralized trusted timestamping using the crypto currency bitcoin," 2015. [Online]. Available: arxiv.abs/1502.04015.

[219] A. Kiayias and M. Yung, "Self-tallying elections and perfect ballot secrecy," in *Proc. 5th Int. Workshop Practice Theory Public Key Cryptosyst. (PKC)*, vol. 2274. Paris, France, Feb. 2002, pp. 141–158.

[220] J. Groth, "Efficient maximal privacy in boardroom voting and anonymous broadcast," in *Proc. 8th Int. Conf. Financial Cryptogr. (FC)*, vol. 3110. Key West, FL, USA, Feb. 2004. pp. 90–104.

[221] D. L. Chaum, “Untraceable electronic mail, return addresses, and digital pseudonyms,” *Commun. ACM*, vol. 24, no. 2, pp. 84–90, Feb. 1981. [Online]. Available: http://doi.acm.org/10.1145/358549.358563

[222] J. D. Cohen and M. J. Fischer, “A robust and verifiable cryptographically secure election scheme,” in *Proc. 26th Annu. Symp. Found. Comput. Sci. (SFCS)*, 1985, pp. 372–382. [Online]. Available: https://doi.org/10.1109/SFCS.1985.2

[223] P. Boucher, *What If Blockchain Technology Revolutionised Voting*, Eur. Parliament, Brussels, Belgium, 2016.

[224] P. McCorry, S. F. Shahandashti, and F. Hao, “A smart contract for boardroom voting with maximum voter privacy,” in *Proc. 21st Int. Conf. Financial Cryptography Data Security (FC)*, Sliema, Malta, Apr. 2017, pp. 357–375.

[225] W. Zhang *et al.*, “A privacy-preserving voting protocol on blockchain,” in *Proc. 11th IEEE Int. Conf. Cloud Comput. (CLOUD)*, San Francisco, CA, USA, Jul. 2018, pp. 401–408.

[226] S. Venugopalan, I. Homoliak, Z. Li, and P. Szalachowski, “BBB-voting: 1-out-of-*k*blockchain-based boardroom voting,” 2020. [Online]. Available: arxiv.abs/2010.09112.

[227] Y. Li, W. Susilo, G. Yang, Y. Yu, D. Liu, and M. Guizani, “A blockchain-based self-tallying voting scheme in decentralized IoT,” 2019. [Online]. Available: arxiv.abs/1902.03710.

[228] J. Benaloh and D. Tuinstra, “Receipt-free secret-ballot elections (extended abstract),” in *Proc. 26th Annu. ACM Symp. Theory Comput. (STOC)*, 1994, pp. 544–553. [Online]. Available: http://doi.acm.org/10.1145/195058.195407

[229] O. Baudron, P.-A. Fouque, D. Pointcheval, J. Stern, and G. Poupard, “Practical multi-candidate election system,” in *Proc. 20th Annu. ACM Symp. Principles Distrib. Comput. (PODC)*, 2001, pp. 274–283.

[230] K. Sako and J. Kilian, “Receipt-free mix-type voting scheme—A practical solution to the implementation of a voting booth,” in *Proc. Int. Conf. Theory Appl. Cryptogr. Techn. Adv. Cryptol. (EUROCRYPT)*, vol. 921. Saint-Malo, France, May 1995, pp. 393–403.

[231] R. Canetti and R. Gennaro, “Incoercible multiparty computation,” in *Proc. 37th Conf. Found. Comput. Sci.*, Oct. 1996, pp. 504–513.

[232] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky, “Deniable encryption,” in *Proc. 17th Annu. Int. Cryptol. Conf. Adv. Cryptol. (CRYPTO)*, vol. 1294. Santa Barbara, CA, USA, Aug. 1997, pp. 90–104.

[233] D. Khader, B. Smyth, P. Y. A. Ryan, and F. Hao, “A fair and robust voting system by broadcast,” in *Proc. 5th Int. Conf. Electron. Voting (EVOTE)*, Jul. 2012, pp. 285–299. [Online]. Available: https://dl.gi.de/20.500.12116/18220.

[234] A. Schaub, R. Bazin, O. Hasan, and L. Brunie, “A trustless privacy-preserving reputation system,” in *Proc. IFIP Int. Conf. ICT Syst. Security Privacy Protect.*, 2016, pp. 398–411.

[235] D. Carboni. (2015). *Feedback Based Reputation on Top of the Bitcoin Blockchain*. [Online]. Available: http://arxiv.org/abs/1502.01504

[236] K. Zhao, S. Tang, B. Zhao, and Y. Wu, “Dynamic and privacy-preserving reputation management for blockchain-based mobile crowdsensing,” *IEEE Access*, vol. 7, pp. 74694–74710, 2019.

[237] A. Grech and A. F. Camilleri, “Blockchain in education,” Publ. Office Eur. Union, Rep., 2017.

[238] P. Buneman, S. Khanna, and T. Wang-Chiew, “Why and where: A characterization of data provenance,” in *Proc. Int. Conf. Database Theory*, 2001, pp. 316–330.

[239] N. Kshetri, “1 blockchain’s roles in meeting key supply chain management objectives,” *Int. J. Inf. Manag.*, vol. 39, pp. 80–89, Apr. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0268401217305248

[240] Deloitte. (2017). *Continuous Interconnected Supply Chain Using Blockchain and Internet-of-Things in Supply Chain Traceability*. [Online]. Available: https://www2.deloitte.com/content/dam/Deloitte/lu/Documents/technology/lu-blockchain-internet-things-supply-chain-traceability.pdf

[241] H. M. Kim and M. Laskowski, “Toward an ontology-driven blockchain design for supply-chain provenance,” *Intell. Syst. Account. Finance Manag.*, vol. 25, no. 1, pp. 18–27, 2018.

[242] S. Shetty, V. Red, C. Kamhoua, K. Kwiat, and L. Njilla, “Data provenance assurance in the cloud using blockchain,” in *Proc. SPIE Defense Security Conf.*, 2017, p. 102060I.

[243] C. Jaag and C. Bach, “Blockchain technology and cryptocurrencies: Opportunities for postal financial services,” in *The Changing Postal and Delivery Sector*, M. Crew, P. L. Parcu, and T. Brennan, Eds. Heidelberg, Germany: Springer, 2017, pp. 205–221.

[244] T. Hardjono and N. Smith, “Cloud-based commissioning of constrained devices using permissioned blockchains,” in *Proc. 2nd ACM Int. Workshop IoT Privacy Trust Security*, 2016, pp. 29–36.

[245] B. Liu, X. L. Yu, S. Chen, X. Xu, and L. Zhu, “Blockchain based data integrity service framework for IoT data,” in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2017, pp. 468–475.

[246] D. Bhowmik and T. Feng, “The multimedia blockchain: A distributed and tamper-proof media transaction framework,” in *Proc. 22nd Int. Conf. Digit. Signal Process. (DSP)*, Aug. 2017, pp. 1–5.

[247] A. Tomescu and S. Devadas, “CATENA: Efficient non-equivocation via bitcoin,” in *Proc. IEEE Symp. Security Privacy (SP)*, May 2017, pp. 393–409.

[248] M. Al-Bassam and S. Meiklejohn, “CONTOUR: A practical system for binary transparency,” in *Proc. Data Privacy Manag. Cryptocurrencies Blockchain Technol.*, 2018, pp. 94–110.

[249] D. Breitenbacher, I. Homoliak, Y. L. Aung, N. O. Tippenhauer, and Y. Elovici, “HADES-IoT: A practical host-based anomaly detection system for IoT devices,” in *Proc. ACM Asia Conf. Comput. Commun. Security (AsiaCCS)*, Auckland, New Zealand, Jul. 2019, pp. 479–484.

[250] I. Homoliak and P. Szalachowski, “AQUAREUM: A centralized ledger enhanced with blockchain and trusted computing,” 2020. [Online]. Available: arxiv.abs/2005.13339.

[251] R. Paccagnella *et al.*, “CUSTOS: Practical tamper-evident auditing of operating systems using trusted execution,” in *Proc. Symp. Netw. Distrib. Syst. Security (NDSS)*, 2020, pp. 1–9.

[252] National Notary Association, CA. (2019). *What Is Notarization?* [Online]. Available: https://www.nationalnotary.org/knowledge-center/about-notaries/what-is-notarization

[253] Heleness. (2018). *Blockchain-Based Notarization Platform on Ethereum*. [Online]. Available: https://ethresear.ch/t/blockchain-based-notarization-platform-on-ethereum/5326

[254] ERC-725 Alliance. (2018). *ERC-725: Ethereum Identity Standard*. [Online]. Available: https://erc725alliance.org/

[255] K. Rantos, G. Drosatos, K. Demertzis, C. Ilioudis, and A. Papanikolaou, “Blockchain-based consents management for personal data processing in the IoT ecosystem.” in *Proc. ICETE*, 2018, pp. 738–743.

[256] A. Mizrahi. (2015). *A Blockchain-Based Property Ownership Recording System*. [Online]. Available: https://static1.squarespace.com/static/5e26f18cd5824c7138a9118b/t/5e3c1c7d6d5ff061da34da80/1580997757765/A-blockchain-based-property-registry.pdf

[257] B. Notheisen, J. B. Cholewa, and A. P. Shanmugam, “Trading real-world assets on blockchain,” *Bus. Inf. Syst. Eng.*, vol. 59, no. 6, pp. 425–440, 2017.

[258] G. Andresen and M. Hearn. (2016). *BIP-70*. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0070.mediawiki

[259] P. McCorry, S. F. Shahandashti, and F. Hao, “Refund attacks on bitcoin’s payment protocol,” in *Proc. Int. Conf. Financ. Cryptography Data Security*, 2016, pp. 581–599.

[260] S. Goldfeder, J. Bonneau, R. Gennaro, and A. Narayanan, “Escrow protocols for cryptocurrencies: How to buy physical goods using bitcoin,” in *Proc. Int. Conf. Financ. Cryptography Data Security*, 2017, pp. 321–339.

[261] O. R. Kabi and V. N. Franqueira, “Blockchain-based distributed marketplace,” in *Proc. Int. Conf. Bus. Inf. Syst.*, 2018, pp. 197–210.

[262] N. Christin, “Traveling the silk road: A measurement analysis of a large anonymous online marketplace,” in *Proc. ACM 22nd Int. Conf. World Wide Web*, 2013, pp. 213–224.

[263] H. S. Galal and A. M. Youssef, “Verifiable sealed-bid auction on the Ethereum blockchain,” in *Proc. Int. Conf. Financ. Cryptography Data Security*, 2018, pp. 265–278.

[264] H. Galal and A. Youssef, “Succinctly verifiable sealed-bid auction smart contract,” in *Proc. Data Privacy Manag. Cryptocurrencies Blockchain Technol.*, 2018, pp. 3–19.

[265] H. S. Galal and A. M. Youssef, “Trustee: Full privacy preserving vickrey auction on top of ethereum,” in *Proc. Financial Cryptography Data Security (FC) Int. Workshops*, Feb. 2019, pp. 190–207.

[266] E.-O. Blass and F. Kerschbaum, “STRAIN: A secure auction for blockchains,” in *Proc. Eur. Symp. Res. Comput. Security*, 2018, pp. 87–110.

[267] J. Ma, B. Qi, and K. Lv, “Fully private auctions for the highest bid,” in *Proc. ACM Turing Celebration Conf. China*, 2019, p. 64.

[268] C. Esposito, A. De Santis, G. Tortora, H. Chang, and K.-K. R. Choo, “Blockchain: A panacea for healthcare cloud-based data security and privacy?” *IEEE Cloud Comput.*, vol. 5, no. 1, pp. 31–37, Mar. 2018.

[269] G. Liang, S. R. Weller, F. Luo, J. Zhao, and Z. Y. Dong, “Distributed blockchain-based data protection framework for modern power systems against cyber attacks,” *IEEE Trans. Smart Grid*, vol. 10, no. 3, pp. 3162–3173, Mar. 2018.

[270] P. K. Sharma, S. Singh, Y.-S. Jeong, and J. H. Park, "DistBlockNet: A distributed blockchains-based secure SDN architecture for IoT networks," *IEEE Commun. Mag.*, vol. 55, no. 9, pp. 78–85, Sep. 2017.

[271] E. Gaetani, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, "Blockchain-based database to ensure data integrity in cloud computing environments," in *Proc. 1st Ital. Conf. Cybersecurity (ITASEC)*, Venice, Italy, Jan. 2017, pp. 146–155.

[272] A. Dorri, S. S. Kanhere, and R. Jurdak, "Towards an optimized blockchain for IoT," in *Proc. ACM 2nd Int. Conf. Internet Things Design Implement.*, 2017, pp. 173–178.

[273] M. Carlsten, H. A. Kalodner, S. M. Weinberg, and A. Narayanan, "On the instability of bitcoin without the block reward," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 154–167. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978408

[274] S. Leonardos, D. Reijsbergen, and G. Piliouras, "PREStO: A systematic framework for blockchain consensus protocols," *IEEE Trans. Eng. Manag.*, vol. 67, no. 4, pp. 1028–1044, Nov. 2020.

[275] K. Wüst and A. Gervais, "Do you need a blockchain?" in *Proc. Crypto Valley Conf. Blockchain Technol. (CVCBT)*, 2018, pp. 45–54.

[276] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's inside the cloud? An architectural map of the cloud landscape," in *Proc. ICSE Workshop Softw. Eng. Challenges Cloud Comput.*, 2009, pp. 23–31.

[277] J. Verschuren, R. Govaerts, and J. Vandewalle, "ISO-OSI security architecture," in *Computer Security and Industrial Cryptography*. Vancouver, BC, Canada: Springer, 1993, pp. 179–192.

[278] S. M. Bellovin, "Security problems in the TCP/IP protocol suite," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 2, pp. 32–48, 1989.

[279] P. Pritzker and W. E. May, *Secure Hash Standard (SHS)*, Nat. Inst. Stand. Technol., Gaithersburg, MD, USA, 2015.

[280] R. Cavanagh, "Digital signature standard; request for comments on the NIST-recommended elliptic curves," Nat. Inst. Stand. Technol., Gaithersburg, MD, USA, Rep. 186-4, 2015.

[281] A. Narayanan. (2018). *Analyzing the 2013 Bitcoin Fork: Centralized Decision-Making Saved the Day*. [Online]. Available: https://freedom-to-tinker.com/2015/07/28/analyzing-the-2013-bitcoin-fork-centralized-decision-making-saved-the-day/

[282] A. Hertig. (2018). *Bitcoin Bug Exploited on Crypto Fork as Attacker Prints 235 Million Pigeoncoins*. [Online]. Available: https://www.coindesk.com/bitcoin-bug-exploited-on-crypto-fork-as-attacker-prints-235-million-pigeoncoins

[283] Bitcoin Wiki. (2016). *Value Overflow Incident*. [Online]. Available: https://en.bitcoin.it/wiki/Value_overflow_incident

[284] Ocminer. (2018). *Network Attack on XVG/VERGE*. Accessed: Jun. 9, 2019. [Online]. Available: https://bitcointalk.org/index.php?topic=3256693.0

[285] F. Tschorsch and B. Scheuermann, "Bitcoin and beyond: A technical survey on decentralized digital currencies," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 3, pp. 2084–2123, 3rd Quart., 2016.

[286] J. Yli-Huumo, D. Ko, S. Choi, S. Park, and K. Smolander, "Where is current research on blockchain technology? A systematic review," *PLoS ONE*, vol. 11, no. 10, 2016, Art. no. e0163477.

[287] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, "A survey on the security of blockchain systems," *Future Gener. Comput. Syst.*, vol. 107, pp. 841–853, Jun. 2020.

[288] M. Conti, E. S. Kumar, C. Lal, and S. Ruj, "A survey on security and privacy issues of bitcoin," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3416–3452, 4th Quart., 2018.

[289] M. Saad *et al.*, "Exploring the attack surface of blockchain: A systematic overview," 2019. [Online]. Available: arxiv.abs/1904.03487.

[290] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A survey on ethereum systems security: Vulnerabilities, attacks, and defenses," *ACM Comput. Surveys*, vol. 53, no. 3, pp. 1–43, 2020.

[291] D. Floyd. (2018). *150K Stolen From MyEtherWallet Users in DNS Server Hijacking*. [Online]. Available: https://www.coindesk.com/150k-stolen-myetherwallet-users-dns-server-hijacking

[292] L. Poinsignon. (2018). *BGP Leaks and Cryptocurrencies*. [Online]. Available: https://blog.cloudflare.com/bgp-leaks-and-crypto-currencies/

[293] R. Zhang, R. Xue, and L. Liu, "Security and privacy on blockchain," *ACM Comput. Surveys*, vol. 52, no. 3, p. 51, Jul. 2019. [Online]. Available: https://doi.org/10.1145/3316481

[294] A. Alkhalifah, A. Ng, A. Kayes, J. Chowdhury, M. Alazab, and P. Watters. (2019). *A Taxonomy of Blockchain Threats and Vulnerabilities*. [Online]. Available: https://doi.org/10.20944/preprints201909.0117.v1.

[295] L. Zhu *et al.*, "Research on the security of blockchain data: A survey," 2018. [Online]. Available: arxiv.abs/1812.02009.

[296] C. Natoli, J. Yu, V. Gramoli, and P. Veríssimo, "Deconstructing blockchains: A comprehensive survey on consensus, membership and structure," 2019. [Online]. Available: arxiv.abs/1908.08316.

[297] T. Neudecker and H. Hartenstein, "Network layer aspects of permissionless blockchains," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 838–857, 2nd Quart., 2018.

[298] D. Harz and W. Knottenbelt, "Towards safer smart contracts: A survey of languages and verification methods," 2018. [Online]. Available: arxiv.abs/1809.09805.

[299] M. Di Angelo and G. Salzer, "A survey of tools for analyzing Ethereum smart contracts," in *Proc. IEEE Int. Conf. Decentralized Appl. Infrastruct. (DAPPCON)*, 2019, pp. 69–78.

[300] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A survey of distributed consensus protocols for blockchain networks," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 1432–1465, 2nd Quart., 2020.

[301] J. A. Garay and A. Kiayias, "SoK: A consensus taxonomy in the blockchain era," in *Proc. IACR Cryptol. ePrint Archive*, vol. 2018, 2018, p. 754.

[302] A. Shahaab, B. Lidgey, C. Hewage, and I. Khan, "Applicability and appropriateness of distributed ledgers consensus protocols in public and private sectors: A systematic review," *IEEE Access*, vol. 7, pp. 43622–43636, 2019.

[303] S. Azouvi and A. Hicks, "SoK: Tools for game theoretic models of security for cryptocurrencies," 2019. [Online]. Available: arxiv.abs/1905.08595.

[304] J. Huang *et al.*, "Survey on blockchain incentive mechanism," in *Proc. Int. Conf. Pioneering Comput. Sci. Eng. Educ.*, 2019, pp. 386–395.

[305] A. Panarello, N. Tapas, G. Merlino, F. Longo, and A. Puliafito, "Blockchain and IoT integration: A systematic survey," *Sensors*, vol. 18, no. 8, p. 2575, 2018.

[306] M. S. Ali, M. Vecchio, M. Pincheira, K. Dolui, F. Antonelli, and M. H. Rehmani, "Applications of blockchains in the Internet of Things: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1676–1717, 3rd Quart., 2018.

[307] F. Restuccia, S. D'Oro, S. S. Kanhere, T. Melodia, and S. K. Das, "Blockchain for the Internet of Things: Present and future," 2019. [Online]. Available: arxiv.abs/1903.07448.

[308] M. A. Ferrag, M. Derdour, M. Mukherjee, A. Derhab, L. Maglaras, and H. Janicke, "Blockchain Technologies for the Internet of Things: Research Issues and Challenges," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2188–2204, Apr. 2019.

[309] Team Rocket. *Snowflake to Avalanche: A Novel Metastable Consensus Protocol Family for Cryptocurrencies*. 2018. [Online]. Available: https://www.semanticscholar.org/paper/Snowflake-to-Avalanche-%3A-A-Novel-Metastable-Family/85ec19594046bbcfe12137c7c2e3744677129820?p2df

[310] S. Popov. (2016). *The Tangle*. [Online]. Available: http://tanglereport.com/wp-content/uploads/2018/01/IOTA_Whitepaper.pdf

[311] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Security*, 2001, pp. 253–280.

[312] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *Proc. USENIX Security*, 2014, pp. 781–796.

[313] S. Micali, "Simple and fast optimistic protocols for fair electronic exchange," in *Proc. ACM 22nd Annu. Symp. Principles Distrib. Comput.*, 2003, pp. 12–19.

[314] Bitcoin Wiki. (2018). *Atomic Swap*. [Online]. Available: https://en.bitcoinwiki.org/wiki/Atomic_Swap

[315] D. Shares. (2016). *Gatecoin Official Statement: Hot Wallet Breach Losses Estimated to Be $2m USD*. [Online]. Available: https://news.bitcoin.com/gatecoin-official-statement-hot-wallet-breach-losses-estimated-2m-usd/

[316] P. Delaney. (2014). *Major Attack Reported on Dogecoin Vault Leads to Shutdown*. [Online]. Available: https://www.ccn.com/major-attack-reported-dogecoin-vault-leads-shutdown/

[317] S. Khandelwal. (2013). *Danish Bitcoin Exchange Bips Hacked and 1,295 Bitcoins Worth $1 Million Stolen*. [Online]. Available: https://thehackernews.com/2013/11/danish-bitcoin-exchange-bips-hacked-and_25.html

[318] S. Higgins. (2016). *The Bitfinex Bitcoin Hack: What We Know (and Don't Know)*. [Online]. Available: https://www.coindesk.com/bitfinex-bitcoin-hack-know-dont-know

[319] D. Bradbury. (2013). *Hackers Hit Bitcoin Central Exchange*. [Online]. Available: https://www.coindesk.com/hackers-hit-bitcoin-central-exchange

[320] C. Jentzsch. (2016). *The History of the DAO and Lessons Learned*. [Online]. Available: https://blog.slock.it/the-history-of-the-dao-and-lessons-learned-d06740f8cfa5/

[321] K. Elby. (2016). *King of the Ether Throne: Post Mortem Investigation*. [Online]. Available: https://www.kingoftheether.com/postmortem.html

[322] S. C. Bürgel. (2016). *Rock Paper Scissors on Ethereum*. [Online]. Available: https://github.com/SCBuergel/ethereum-rps

[323] H. Official. (2018). *Unit Underflows and Overflows—Ethereum Solidity Vulnerability*. [Online]. Available: https://medium.com/haloblock/unit-underflows-and-overflows-ethereum-solidity-vulnerability-39a39355c422

[324] B. Mahendar. (2019). *Unexpected Ether*. [Online]. Available: https://www.nvestlabs.com/2019/04/23/unexpected-ether/

[325] P. Technologies. (2017). *The Multi-Sig Hack: A Postmortem* [Online]. Available: https://www.parity.io/the-multi-sig-hack-a-postmortem/

[326] E. Ward. (2018). *Breaking Randomness in the Ethereum Universe*. [Online]. Available: https://blog.gdssecurity.com/labs/2018/6/1/breaking-randomness-in-the-ethereum-universe-part-1.html

[327] F. Hildebrand. (2018). *The State of Ponzi Schemes*. [Online]. Available: https://medium.com/solidified/the-state-of-ponzi-schemes-98dde4518431

[328] J. Wilcke. (2016). *The Ethereum Network Is Currently Undergoing a DoS Attack*. [Online]. Available: https://blog.ethereum.org/2016/09/22/ethereum-network-currently-undergoing-dos-attack/

[329] GuardRails. *Write to Arbitrary Storage Locations*. Accessed: Aug. 16, 2019. [Online]. Available: https://www.guardrails.io/docs/en/vulnerabilities/solidity/write_to_arbitrary_storage_location

[330] A. Akentiev. (2017). *Parity Multisig Hacked. Again*. [Online]. Available: https://medium.com/chain-cloud-company-blog/parity-multisig-hack-again-b46771eaa838

[331] A. Butler. (2019). *Ethereum Classic Attacked! How Does the 51% Attack Occur?*. [Online]. Available: https://hackernoon.com/ethereum-classic-attacked-how-does-the-51-attack-occur-a5f3fa5d852e

[332] D. Gutteridge. (2018). *Japanese Cryptocurrency Monacoin Hit by Selfish Mining Attack*. [Online]. Available: https://www.ccn.com/japanese-cryptocurrency-monacoin-hit-by-selfish-mining-attack/

[333] G. Milton. (2018). *Hackers Launch 'Double-Spend' Attack on Bitcoin Gold to Steal Over $18 Million*. [Online]. Available: https://cyware.com/news/hackers-launch-double-spend-attack-on-bitcoin-gold-to-steal-over-18-million-011a29e2

[334] Bitcoin Exchange Guide News Team. (2018). *Litecoin Cash (LCC) Experiencs 51% Attack, Cryptocurrency Dies on Spot*. [Online]. Available: https://bitcoinexchangeguide.com/litecoin-cash-lcc-experiencs-51-attack-cryptocurrency-dies-on-spot/

[335] ZenCash. (2018). *Zencash Statement on Double Spend Attack*. [Online]. Available: https://blog.zencash.com/zencash-statement-on-double-spend-attack/

[336] K. Sedgwick. (2018). *Verge Is Forced to Fork After Suffering a 51% Attack*. [Online]. Available: https://news.bitcoin.com/verge-is-forced-to-fork-after-suffering-a-51-attack/

[337] A. Judmayer *et al.*, "Pay-to-win: Incentive attacks on proof-of-work cryptocurrencies," *Proc. IACR Cryptol. ePrint Archive*, 2019.

[338] A. Greenberg. (2014). *Hacker Redirects Traffic From 19 Internet Providers to Steal Bitcoins*. [Online]. Available: https://www.wired.com/2014/08/isp-bitcoin-theft/

[339] D. Canellis. (2019). *Bitcoin Wallet Electrum Hit by DoS Attack From 140,000-Strong Botnet*. [Online]. Available: https://thenextweb.com/hardfork/2019/04/08/bitcoin-wallet-electrum-dos-attack-botnet-phishing/

[340] M. Apostolaki *et al.* (2017). *Blockchain Meets Internet Routing*. [Online]. Available: https://btc-hijack.ethz.ch/

[341] M. Saad, L. Njilla, C. Kamhoua, J. Kim, D. Nyang, and A. Mohaisen, "Mempool optimization for defending against ddos attacks in pow-based blockchain systems," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC)*, 2019, pp. 285–292.

[342] J. Young. (2017). *Analyst: Suspicious Bitcoin Mempool Activity, Transaction Fees Spike to $16*. [Online]. Available: https://cointelegraph.com/news/analyst-suspicious-bitcoin-mempool-activity-transaction-fees-spike-to-16

StrongChain: Transparent and Collaborative Proof-of-Work Consensus

Pawel Szalachowski, Daniël Reijsbergen, and Ivan Homoliak, *Singapore University of Technology and Design (SUTD);* Siwei Sun, *Institute of Information Engineering and DCS Center, Chinese Academy of Sciences*

https://www.usenix.org/conference/usenixsecurity19/presentation/szalachowski

This paper is included in the Proceedings of the
28th USENIX Security Symposium.
August 14–16, 2019 • Santa Clara, CA, USA
978-1-939133-06-9

StrongChain: Transparent and Collaborative Proof-of-Work Consensus

Pawel Szalachowski[1] Daniël Reijsbergen[1] Ivan Homoliak[1] Siwei Sun[2,*]
[1]*Singapore University of Technology and Design (SUTD)*
[2]*Institute of Information Engineering and DCS Center, Chinese Academy of Sciences*

Abstract

Bitcoin is the most successful cryptocurrency so far. This is mainly due to its novel consensus algorithm, which is based on proof-of-work combined with a cryptographically-protected data structure and a rewarding scheme that incentivizes nodes to participate. However, despite its unprecedented success Bitcoin suffers from many inefficiencies. For instance, Bitcoin's consensus mechanism has been proved to be incentive-incompatible, its high reward variance causes centralization, and its hardcoded deflation raises questions about its long-term sustainability.

In this work, we revise the Bitcoin consensus mechanism by proposing StrongChain, a scheme that introduces transparency and incentivizes participants to collaborate rather than to compete. The core design of our protocol is to reflect and utilize the computing power aggregated on the blockchain which is invisible and "wasted" in Bitcoin today. Introducing relatively easy, although important changes to Bitcoin's design enables us to improve many crucial aspects of Bitcoin-like cryptocurrencies making it more secure, efficient, and profitable for participants. We thoroughly analyze our approach and we present an implementation of StrongChain. The obtained results confirm its efficiency, security, and deployability.

1 Introduction

One of the main novelties of Bitcoin [28] is Nakamoto consensus. This mechanism enabled the development of a permissionless, anonymous, and Internet-scale consensus protocol, and combined with incentive mechanisms allowed Bitcoin to emerge as the first decentralized cryptocurrency. Bitcoin is successful beyond all expectations, has inspired many other projects, and has started new research directions. Nakamoto consensus is based on proof-of-work (PoW) [8] in order to mitigate Sybil attacks [6]. To prevent modifications, a cryptographically-protected append-only list [2] is introduced. This list consists of transactions grouped into blocks and is usually referred to as a *blockchain*. Every active protocol participant (called a *miner*) collects transactions sent by users and tries to solve a computationally-hard puzzle in order to be able to write to the blockchain (the process of solving the puzzle is called *mining*). When a valid solution is found, it is disseminated along with the transactions that the miner wishes to append. Other miners verify this data and, if valid, append it to their replicated blockchains. The miner that has found a solution is awarded by a) the system, via a rewarding scheme programmed into the protocol, and b) fees paid by transaction senders. All monetary transfers in Bitcoin are expressed in its native currency (called bitcoin, abbreviated as BTC) whose supply is limited by the protocol.

Bitcoin has started an advent of decentralized cryptocurrency systems and as the first proposed and deployed system in this class is surprisingly robust. However, there are multiple drawbacks of Bitcoin that undermine its security promises and raise questions about its future. Bitcoin has been proved to be incentive-incompatible [9,11,39,47]. Namely, in some circumstances, the miners' best strategy is to not announce their found solutions immediately, but instead withhold them for some time period. Another issue is that the increasing popularity of the system tends towards its centralization. Strong competition between miners resulted in a high reward variance, thus to stabilize their revenue miners started grouping their computing power by forming *mining pools*. Over time, mining pools have come to dominate the computing power of the system, and although they are beneficial for miners, large mining pools are risky for the system as they have multiple ways of abusing the protocol [9, 11, 18, 39]. Recently, researchers rigorously analyzed one of the impacts of Bitcoin's deflation [4, 27, 47]. Their results indicate that Bitcoin may be unsustainable in the long term, mainly due to decreasing miners' rewards that will eventually stop completely. Besides that, unusually for a transaction system, Bitcoin is designed to favor availability over consistency. This choice was motivated by its open and

[*] This work was done while the author was at SUTD.

permissionless spirit, but in the case of inconsistencies (i.e., *forks* in the blockchain) the system can be slow to converge.

Motivated by these drawbacks, we propose StrongChain, a simple yet powerful revision of the Bitcoin consensus mechanism. Our main intuition is to design a system such that the mining process is more transparent and collaborative, i.e., miners get better knowledge about the mining power of the system and they are incentivized to solve puzzles together rather than compete. In order to achieve it, in the heart of the StrongChain's design we employ *weak solutions*, i.e., puzzle solutions with a PoW that is significant yet insufficient for a standard solution. We design our system, such that a) weak solutions are part of the consensus protocol, b) their finders are rewarded independently, and c) miners have incentives to announce own solutions and append solutions of others immediately. We thoroughly analyze our approach and show that with these changes, the mining process is becoming more transparent, collaborative, secure, efficient, and decentralized. Surprisingly, we also show how our approach can improve the freshness properties offered by Bitcoin. We present an implementation and evaluation of our scheme.

2 Background and Problem Definition

2.1 Nakamoto Consensus and Bitcoin

The Nakamoto consensus protocol allows decentralized and distributed network comprised of mutually distrusting participants to reach an agreement on the state of the global distributed ledger [28]. The distributed ledger can be regarded as a linked list of blocks, referred to as the *blockchain*, which serializes and confirms "transactions". To resolve any *forks* of the blockchain the protocol specifies to always accept the longest chain as the current one. Bitcoin is a peer-to-peer cryptocurrency that deploys Nakamoto consensus as its core mechanism to avoid double-spending. Transactions spending bitcoins are announced to the Bitcoin network, where miners validate, serialize all non-included transactions, and try to create (mine) a block of transactions with a PoW embedded into the block header. A valid block must fulfill the condition that for a cryptographic hash function H, the hash value of the block header is less than the target T.

Brute-forcing the nonce (together with some other changeable data fields) is virtually the only way to produce the PoW, which costs computational resources of the miners. To incentivize miners, the Bitcoin protocol allows the miner who finds a block to insert a special transaction (see below) minting a specified amount of new bitcoins and collecting transaction fees offered by the included transactions, which are transferred to an account chosen by the miner. Currently, every block mints 12.5 new bitcoins. This amount is halved every four years, upper-bounding the number of bitcoins that will be created to a fixed total of 21 million coins. It implies that after around the year 2140, no new coins will be created, and the transaction fees will be the only source of reward for miners. Because of its design, Bitcoin is a deflationary currency.

The overall hash rate of the Bitcoin network and the difficulty of the PoW determine how long it takes to generate a new block for the whole network (the block interval). To stabilize the block interval at about 10 minutes for the constantly changing total mining power, the Bitcoin network adjusts the target T every 2016 blocks (about two weeks, i.e., a *difficulty window*) according to the following formula

$$T_{new} = T_{old} \cdot \frac{\textit{Time of the last 2016 blocks}}{\textit{2016} \cdot \textit{10 minutes}}. \tag{1}$$

In simple terms, the difficulty increases if the network is finding blocks faster than every 10 minutes, and decrease otherwise. With dynamic difficulty, Nakamoto's longest chain rule was considered as a bug,[1] as it is trivial to produce long chains that have low difficulty. The rule was replaced by the strongest-PoW chain rule where competing chains are measured in terms of PoW they aggregated. As long as there is one chain with the highest PoW, this chain is chosen as the current one.

Bitcoin introduced and uses the *unspent transaction output* model. The validity of a Bitcoin transaction is verified by executing a script proving that the transaction sender is authorized to redeem unspent coins. The only exception is the first transaction in the transaction list of a block, which implements how the newly minted bitcoins and transaction fees are distributed. It is called a *coinbase transaction* and it contains the amount of bitcoins (the sum of newly minted coins and the fees derived from all the transactions) and the beneficiary (typically the creator of the block). Also, the Bitcoin scripting language offers a mechanism (`OP_RETURN`) for recording data on the blockchain, which facilitates third-party applications built-on Bitcoin.

Bitcoin proposes the simplified payment verification (SPV) protocol, that allows resource-limited clients to verify that a transaction is indeed included in a block provided only with the block header and a short transaction's inclusion proof. The key advantage of the protocol is that SPV clients can verify the existence of a transaction without downloading or storing the whole block. SPV clients are provided only with block headers and on-demand request from the network inclusion proofs of the transactions they are interested in.

In the original white paper, Nakamoto heuristically argues that the consensus protocol remains secure as long as a majority ($>$ 50%) of the participants' computing power honestly follow the rule specified by the protocol, which is compatible with their own economic incentives.

[1] https://goo.gl/thhusi

2.2 Bitcoin Mining Issues

Despite its popularity, Nakamoto consensus and Bitcoin suffer from multiple issues. Bitcoin mining is not always incentive-compatible. By deviating from the protocol and strategically withholding found blocks, a miner in possession of a proportion α of the total computational power may occupy more than α portion of the blocks on the blockchain, and therefore gain disproportionally higher payoffs with respect to her share [1, 11, 39]. More specifically, an attacker tries to create a private chain by keeping found blocks secret as long as the chain is in an advantageous position with one or more blocks more than the public branch. She releases her private chain only when the public chain has almost caught up, hence invalidating the public branch and all the efforts made by the honest miners. This kind of attack, called *selfish mining*, can be more efficient when a well-connected selfish miner's computational power exceeds a certain threshold (around more than 30%). Thus, selfish mining does not pay off if the mining power is sufficiently decentralized.

Unfortunately, the miners have an impulse to centralize their computing resources due to Bitcoin's rewarding scheme. In Bitcoin, rewarding is a zero-sum game and only the lucky miner who manages to get her block accepted receives the reward, while others who indeed contributed computational resources to produce the PoW are completely invisible and ignored. Increasing mining competition leads to an extremely high variance of the payoffs of a miner with a limited computational power. A solo miner may need to wait months or years to receive any reward at all. As a consequence, miners are motivated to group their resources and form mining pools, that divide work among pool participants and share the rewards according to their contributions. As of November 2018, only five largest pools account for more than 65% of the mining power of the whole Bitcoin network.[2] Such mining pools not only undermine the decentralization property of the system but also raise various in-pool or cross-pool security issues [5, 9, 22, 37].

Another seemingly harmless characteristic of Bitcoin is its finite monetary supply. However, researchers in their recent work [4, 27, 47] investigate the system dynamics when incentives coming from transaction fees are non-negligible compared with block rewards (in one extreme case the incentives come only from fees). They provide analysis and evidence, indicating an undesired system degradation due to the rational and self-interested participants. Firstly, such a system incentivizes large miner coalitions, increasing the system centralization even more. Secondly, it leads to a mining gap where miners would avoid mining when the available fees are insufficient. Even worse, rational miners tend to mine on chains that do not include available transactions (and their fees), rather than following the block selection rule specified by the protocol, resulting in a backlog of transactions. Finally, in the sole transaction fee regime, selfish mining attacks are efficient for miners with arbitrarily low mining power, regardless of their network connection qualities. These results suggest that making the block reward permanent and accepting the monetary inflation may be a wise design choice to ensure the stability of the cryptocurrency in the long run.

Moreover, the chain selection rule (i.e., the strongest chain is accepted), together with the network delay, occasionally lead to forks, where two or more blocks pointing to the same block are created around the same time, causing the participants to have different views of the current system state. Such conflicting views will eventually be resolved since with a high probability one branch will finally beat the others (then the blocks from the "losing" chain become *stale blocks*). The process of fork resolution is quite slow, as blocks have the same PoW weight and they arrive in 10-minutes intervals (on average).

Finally, the freshness properties provided by Bitcoin are questionable. By design, the Bitcoin blockchain preserves the order of blocks and transactions, however, the accurate estimation of time of these events is challenging [43], despite the fact that each block has an associated timestamp. A block's timestamp is accepted if a) it is greater than the median timestamp of the previous eleven blocks, and b) it is less than the network time plus two hours.[3] This gives significant room for manipulation — in theory, a timestamp can differ in hours from the actual time since it is largely determined by a single block creator. In fact, as time cannot be accurately determined from the timestamps, the capabilities of the Bitcoin protocol as a timestamping service are limited, which may lead to severe attacks by itself [3, 17].

2.3 Requirements

For the purpose of revising a consensus protocol of PoW blockchains in a secure, well-incentivized, and seamless way, we define the following respective requirements:

- **Security** – the scheme should improve the security of Nakamoto consensus by mitigating known attack vectors and preventing new ones. In essence, the scheme should be incentive-compatible, such that miners benefit from following the consensus rules and have no gain from violating them.
- **Reward Variance** – another objective is to minimize the variance in rewards. This requirement is crucial for decentralization since a high reward variance is the main motivation of individual miners to join centralized mining pools. Centralization is undesirable as large-enough mining pools can attack the Bitcoin protocol.
- **Chain Quality** – the scheme should provide a high chain quality, which usually is described using the two following properties.

[2] `https://btc.com/stats/pool?pool_mode=month`

[3] `https://en.bitcoin.it/wiki/Block_timestamp`

- **Mining Power Utilization** – the ratio between the mining power on the main chain and the mining power of the entire blockchain network. This property describes the performance of mining and its ideal value is 1, which denotes that all mining power of the system contributes to the "official" or "canonical" chain. A high mining power utilization implies a low stale block rate.
- **Fairness** – the protocol should be fair, i.e., a miner should earn rewards proportionally to the resources invested by her in mining. We denote a miner with α of the global mining power as an *α-strong miner*.

- **Efficiency and Practicality** – the scheme should not introduce any significant computational, storage, or bandwidth overheads. This is especially important since Bitcoin works as a replicated state machine, therefore all full nodes replicate data and the validation process. In particular, the block validation time, its size, and overheads of SPV clients should be at least similar as today. Moreover, the protocol should not introduce any assumptions that would be misaligned with Bitcoin's spirit and perceived as unacceptable by the community. In particular, the scheme should not introduce any trusted parties and should not assume strong synchronization of nodes (like global and reliable timestamps).

3 High-level Overview

3.1 Design Rationale

Our first observation is that Bitcoin mining is not transparent. It is difficult to quickly estimate the computing power of the different participants, because the only indicator is the found blocks. After all, blocks arrive with a low frequency, and each block is equal in terms of its implied computational power. Consequently, the only way of resolving forks is to wait for a stronger chain to emerge, which can be a time-consuming process. A related issue is block-withholding-like attacks (e.g., selfish mining) which are based on the observation that sometimes it is profitable for an attacker to deviate from the protocol by postponing the announcement of new solutions. We see transparency as a helpful property also in this context. Ideally, non-visible (hidden) solutions should be penalized, however, in practice it is challenging to detect and prove that a solution was hidden. We observe that an alternative way of mitigating these attacks would be to promote visible solutions, such that with more computing power aggregated around them they get stronger. This would incentivize miners to publish their solutions immediately, since keeping it secret may be too risky as other miners could strengthen a competing potential (future) solution over time. Finally, supported by recent research results [4, 11, 27, 39, 47], we envision that redesigning the Bitcoin reward scheme is unavoidable to keep the system sustainable and more secure. Beside the deflation issues (see Section 2.2), the reward scheme in Bitcoin is a zero-sum game rewarding only lucky miners and ignoring all effort of other participants. That causes fierce competition between miners and a high reward variance, which stimulates miners to collaborate, but within mining pools, introducing more risk to the system. We aim to design a system where miners can benefit from collaboration but without introducing centralization risks.

3.2 Overview

Motivated by these observations, we see weak puzzle solutions, currently invisible and "wasted" in Bitcoin, as a promising direction. Miners exchanging them could make the protocol more transparent as announcing them could reflect the current distribution of computational efforts on the network. Furthermore, if included in consensus rules, they could give blocks a better granularity in terms of PoW, and incentivize miners to collaborate. In our scheme, miners solve a puzzle as today but in addition to publishing solutions, they exchange weak solutions too (i.e., almost-solved puzzles). The lucky miner publishes her solution that embeds gathered weak solutions (pointing to the same previous block) of other miners. Such a published block better reflects the aggregated PoW of a block, which in the case of a fork can indicate that more mining power is focused on a given branch (i.e., actually it proves that more computing power "believes" that the given branch is correct). Another crucial change is to redesign the Bitcoin reward system, such that the finders of weak solutions are also rewarded. Following lessons learned from mining pool attacks, instead of sharing rewards among miners, our scheme rewards weak solutions proportionally to their PoW contributed to a given block and all rewards are independent of other solutions of the block. (Note, that this change requires a Bitcoin *hard fork*.)

There are a few intuitions behind these design choices. First, a selfish miner finding a new block takes a high risk by keeping this block secret. This is because blocks have a better granularity due to honest miners exchanging partial solutions and strengthening their prospective block, which in the case of a fork would be stronger than the older block kept secret (i.e., the block of the selfish miner). Secondly, miners are actually incentivized to collaborate by a) exchanging their weak solutions, and b) by appending weak solutions submitted by other miners. For the former case, miners are rewarded whenever their solutions are appended, hence keeping them secret can be unprofitable for them. For the latter case, a miner appending weak solutions of others only increases the strength of her potential block, and moreover, appending these solutions does not negatively influence the miner's potential reward. Finally, our approach comes with another benefit. Proportional rewarding of weak solutions decreases the reward variance, thus miners do not have to join large mining pools in order to stabilize their revenue. This could lead to a higher decentralization of mining power on the network.

In the following sections, we describe details of our system, show its analysis, and report on its implementation.

4 StrongChain Details

4.1 Mining

As in Bitcoin, in StrongChain miners authenticate transactions by collecting them into blocks whose headers are protected by a certain amount of PoW. A simplified description of a block mining procedure in StrongChain is presented as the *mineBlock()* function in Algorithm 1. Namely, every miner tries to solve a PoW puzzle by computing the hash function over a newly created header. The header is constantly being changed by modifying its nonce field,[4] until a valid hash value is found. Whenever a miner finds a header *hdr* whose hash value $h = H(hdr)$ is smaller than the *strong target* T_s, i.e., a h that satisfies the following:

$$h < T_s,$$

then the corresponding block is announced to the network and becomes, with all its transactions and metadata, part of the blockchain. We refer to headers of included blocks as *strong headers*.

One of the main differences with Bitcoin is that our mining protocol handles also headers whose hash values do not meet the strong target T_s, but still are low enough to prove a significant PoW. We call such a header a *weak header* and its hash value h has to satisfy the following:

$$T_s \leq h < T_w, \qquad (2)$$

where $T_w > T_s$ and T_w is called the *weak target*.

Whenever a miner finds such a block header, she adds it to her local list of weak headers (i.e., *weakHdrsTmp*) and she propagates the header among all miners. Then every miner that receives this information first validates it (see *onRecvWeakHdr()*) by checking whether

- the header points to the last strong header,
- its other fields are correct (see Section 4.2),
- and Equation 2 is satisfied.

Afterward, miners append the header to their lists of weak headers. We do not limit the number of weak headers appended, although this number is correlated with the T_w/T_s ratio (see Section 5).

Finally, miners continue the mining process in order to find a strong header. In this process, a miner keeps creating candidate headers by computing hash values and checking whether the strong target is met. Every candidate header

Algorithm 1: Pseudocode of StrongChain functions.

```
function mineBlock()
    weakHdrsTmp ← ∅;
    for nonce ∈ {0,1,2,...} do
        hdr ← createHeader(nonce);
        /* check if the header meets the strong target */
        h_tmp ← H(hdr);
        if h_tmp < T_s then
            B ← createBlock(hdr, weakHdrsTmp, Txs);
            broadcast(B);
            return; /* signal to mine with the new block */
        /* check if the header meets the weak target */
        if h_tmp < T_w then
            weakHdrsTmp.add(hdr);
            broadcast(hdr);

function onRecvWeakHdr(hdr)
    h_w ← H(hdr);
    assert(T_s ≤ h_w < T_w and validHeader(hdr));
    assert(hdr.PrevHash == H(lastBlock.hdr)) ;
    weakHdrsTmp.add(hdr);

function rewardBlock(B)
    /* reward block finder with R */
    reward(B.hdr.Coinbase, R + B.TxFees);
    w ← γ ∗ T_s/T_w; /* reward weak headers proportionally */
    for hdr ∈ B.weakHdrSet do
        reward(hdr.Coinbase, w ∗ c ∗ R);

function validateBlock(B)
    assert(H(B.hdr) < T_s and validHeader(B.hdr));
    assert(B.hdr.PrevHash == H(lastBlock.hdr)) ;
    assert(validTransactions(B));
    for hdr ∈ B.weakHdrSet do
        assert(T_s ≤ H(hdr) < T_w and validHeader(hdr));
        assert(hdr.PrevHash == H(lastBlock.hdr));

function chainPoW(chain)
    sum ← 0;
    for B ∈ chain do
        /* for each block compute its aggregated PoW */
        T_s ← B.hdr.Target;
        sum ← sum + T_max/T_s;
        for hdr ∈ B.weakHdrSet do
            sum ← sum + T_max/T_w;
    return sum;

function getTimestamp(B)
    sumT ← B.hdr.Timestamp;
    sumW ← 1.0;
    /* average timestamp by the aggregated PoW */
    w ← T_s/T_w;
    for hdr ∈ B.weakHdrSet do
        sumT ← sumT + w ∗ hdr.Timestamp;
        sumW ← sumW + w;
    return sumT/sumW;
```

[4] In fact, other fields can be modified too if needed.

"protects" all collected weak headers (note that all of these weak headers point to the same previous strong header).

In order to keep the number of found weak headers close to a constant value, StrongChain adjusts the difficulty T_w of weak headers every 2016 blocks immediately following the adjustment of the difficulty T_s of the strong headers according to Equation 1, such that the ratio T_w/T_s is kept at a constant (we discuss its value in Section 5).

4.2 Block Layout and Validation

A block in our scheme consists of transactions, a list of weak headers, and a strong header that authenticates these transactions and weak headers. Strong and weak headers in our system inherit the fields from Bitcoin headers and additionally enrich it by a new field. A block header consists of the following fields:

PrevHash: is a hash of the previous block header,

Target: is the value encoding the current target defining the difficulty of finding new blocks,

Nonce: is a nonce, used to generate PoW,

Timestamp: is a Unix timestamp,

TxRoot: is the root of the Merkle tree [24] aggregating all transactions of the block, and

Coinbase: represents an address of the miner that will receive a reward.

As our protocol rewards finders of weak headers (see details in Section 4.4), every weak header has to be accompanied with the information necessary to identify its finder. Otherwise, a finder of a strong block could maliciously claim that some (or all) weak headers were found by her and get rewards for them. For this purpose and for efficiency, we introduced a new 20B-long header field named *Coinbase*. With the introduction of this field, StrongChain headers are 100B long. But on the other hand, there is no longer any need for Bitcoin coinbase transactions (see Section 2.1), as all rewards are determined from headers.

In our scheme, weak headers are exchanged among nodes as part of a block, hence it is necessary to protect the integrity of all weak headers associated with the block. To realize it, we introduce a special transaction, called a *binding transaction*, which contains a hash value computed over the weak headers. This transaction is the first transaction of each block and it protects the collected weak headers. Whenever a strong header is found, it is announced together with all its transactions and collected weak headers, therefore, this field protects all associated weak headers. To encode this field we utilize the `OP_RETURN` operation as follows:

$$\texttt{OP\_RETURN} \quad H(hdr_0 \| hdr_1 \| ... \| hdr_n), \tag{3}$$

where hdr_i is a weak header pointing to the previous strong header. Since weak headers have redundant fields (the *PrevHash*, *Target*, and *Version* fields have the same values as the strong header), we propose to save bandwidth and storage by not including these fields into the data of a block. This modification reduces the size of a weak header from 100B to 60B only, which is especially important for SPV clients who keep downloading new block headers.

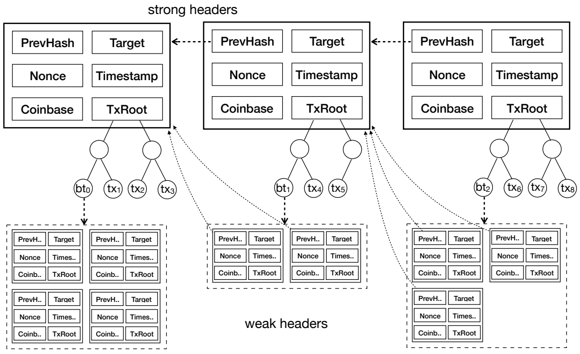


Figure 1: An example of a blockchain fragment with strong headers, weak headers, and binding and regular transactions.

With our approach, a newly mined and announced block can encompass multiple weak headers. Weak headers, in contrast to strong headers, are not used to authenticate transactions, and they are even stored and exchanged *without* their corresponding transactions. Instead, the main purpose of including weak headers it to contribute and reflect the aggregated mining power concentrated on a given branch of the blockchain. We present a fragment of a blockchain of StrongChain in Figure 1. As depicted in the figure, each block contains a single strong header, transactions, and a set of weak headers aggregated via a binding transaction.

On receiving a new block, miners validate the block by checking the following (see *validateBlock()* in Algorithm 1):

1. The strong header is protected by the PoW and points to the previous strong header.
2. Header fields have correct values (i.e., the version, target, and timestamp are set correctly).
3. All included transactions are correct and protected by the strong header. This check also includes checking that all weak headers collected are protected by a binding transaction included in the block.
4. All included weak headers are correct: a) they meet the targets as specified in Equation 2, b) their *PrevHash* fields point to the previous strong header, and c) their version, targets, and timestamps have correct values.

If the validation is successful, the block is accepted as part of the blockchain.

4.3 Forks

One of the main advantages of our approach is that blocks reflect their aggregated mining power more precisely. Each block beside its strong header contains multiple weak head-

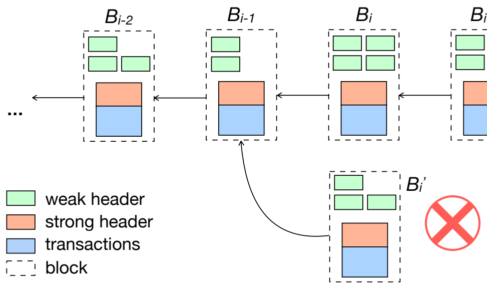


Figure 2: An example of a forked blockchain in StrongChain.

ers that contribute to the block's PoW. In the case of a fork, our scheme relies on the strongest chain rule, however, the PoW is computed differently than in Bitcoin. For every chain its PoW is calculated as presented by the *chainPoW()* procedure in Algorithm 1. Every chain is parsed and for each of its blocks the PoW is calculated by adding:

1. the PoW of the strong header, computed as T_{max}/T_s, where T_{max} is the maximum target value, and
2. the accumulated PoW of all associated weak headers, counting each weak header equally as T_{max}/T_w.

Then the chain's PoW is expressed as just the sum of all its blocks' PoW. Such an aggregated chain's PoW is compared with the competing chain(s). The chain with the largest aggregated PoW is determined as the current one. As difficulty in our protocol changes over time, the strong target T_s and PoW of weak headers are relative to the maximum target value T_{max}. We assume that nodes of the network check whether every difficulty window is computed correctly (we skipped this check in our algorithms for easy description).

Including and empowering weak headers in our protocol moves away from Bitcoin's "binary" granularity and gives blocks better expression of the PoW they convey. An example is presented in Figure 2. For instance, nodes having the blocks B_i and B'_i can immediately decide to follow the block B_i as it has more weak headers associated, thus it has accumulated more PoW than the block B'_i.

An exception to this rule is when miners solve conflicts. Namely, on receiving a new block, miners run the algorithm as presented, however, they also take into consideration PoW contributions of known weak headers that point to the last blocks. For instance, for a one-block-long fork within the same difficulty window, if a block B includes l weak headers and a miner knows of k weak headers pointing to B, then that miner will select B over any competing block B' that includes l' weak and has k' known weak headers pointing to it if $l+k > l'+k'$. Note that this rule incentivizes miners to propagate their solutions as quickly as possible as competing blocks become "stronger" over time.

4.4 Rewarding Scheme

The rewards distribution is another crucial aspect of StrongChain and it is presented by the *rewardBlock()* procedure from Algorithm 1. The miner that found the strong header receives the full reward R. Moreover, in contrast to Bitcoin, where only the "lucky" miner is paid the full reward, in our scheme all miners that have contributed to the block's PoW (i.e., whose weak headers are included) are paid by commensurate rewards to the provided PoW. A weak header finder receive a fraction of R, i.e., $\gamma * c * R * T_s/T_w$, as a reward for its corresponding solution contributing to the total PoW of a particular branch, where the γ parameter influences the relative impact of weak header rewards and c is just a scaling constant (we discuss their potential values and implications in Section 5). Moreover, we do not limit weak header rewards and miners can get multiple rewards for their weak headers within a single block. Similar reward mechanisms are present in today's mining pools (see Section 8), but unlike them, weak header rewards in StrongChain are independent of each other. Therefore, the reward scheme is not a zero-sum game and miners cannot increase their own rewards by dropping weak headers of others (actually, as we discuss in Section 5, they can only lose since their potential solutions would have less PoW without others' weak headers). Furthermore, weak header rewards decrease significantly the mining variance as miners can get steady revenue, making the system more decentralized and collaborative.

As mentioned before, the number of weak headers of a block is unlimited, they are rewarded independently (i.e., do not share any reward), and all block rewards in our system are proportional to the PoW contributed. In such a setting, a mechanism incentivizing miners to terminate a block creation is needed (without such a mechanism, miners could keep creating huge blocks with weak headers only). In order to achieve this, StrongChain always attributes block transaction fees ($B.TxFees$) to the finder of the strong header (who also receives the full reward R).

Note that in our rewarding scheme, the amount of newly minted coins is always at least R, and consequently, unlike Bitcoin or Ethereum [48], the total supply of the currency in our protocol is not upper-bounded. This design decision is made in accordance with recent results on the long-term instability of deflationary cryptocurrencies [4, 27, 47].

4.5 Timestamps

In StrongChain, we follow the Bitcoin rules on constraining timestamps (see Section 2.1), however, we redefine how block timestamps are interpreted. Instead of solely relying on a timestamp put by the miner who mined the block, block timestamps in our system are derived from the strong header and all weak headers included in the corresponding block. The algorithm to derive a block's timestamp is presented as *getTimestamp()* in Algorithm 1. A block's timestamp is determined as a weighted average timestamp over the strong header's timestamp and all timestamps of the weak headers included in the block. The strong header's timestamp has a weight of 1, while weights of weak header timestamps are determined as their PoW contributed (namely, a weak header's timestamp has a weight of the ratio between the strong target and the weak target). Therefore, the timestamp value is adjusted proportionally to the mining power associated with a given block. That change reflects an average time of the block creation and mitigates miners that intentionally or misconfigured put incorrect timestamps into the blockchain. We show the effectiveness of this approach in Section 5.5.

4.6 SPV Clients

Our protocol supports light SPV clients. With every new block, an SPV client is updated with the following information:

$$hdr, hdr_0, hdr_1, ..., hdr_n, BTproof, \quad (4)$$

where *hdr* is a strong header, hdr_i are associated weak headers, and *BTproof* is an inclusion proof of a binding transaction that contains a hash over the weak headers (see Equation 3). Note that headers contain redundant fields, thus as described in Section 4.2, they can be provided to SPV clients efficiently.

With this data, the client verifies fields of all headers, computes the PoW of the block (analogous, as in *chainPoW()* from Algorithm 1), and validates the *BTproof* proof to check whether all weak headers are correct, and whether the transaction is part of the blockchain (the proof is validated against *TxRoot* of *hdr*). Afterward, the client saves the strong header *hdr* and its computed PoW, while other messages (the weak headers and the proof) can be dropped.

5 Analysis

In this section, we evaluate the requirements discussed in Section 2.3. We start with analyzing StrongChain's efficiency and practicality. Next, we study how our design helps with reward variance, chain quality, and security.

5.1 Efficiency and Practicality

For the efficiency, it is important to consider the main source of additional load on the bandwidth, storage, and processing power of the nodes: the weak headers. Hence, in the following section we analyze the probability distribution of the number of weak headers. Next, we discuss the value of the impact of the parametrization on the average block rewards.

5.1.1 Number of Weak Headers

In Bitcoin, we assume that hashes are drawn randomly between 0 and $T_{max} = 2^{256} - 1$. Hence, a single hash being smaller than T_w is a *Bernoulli trial* with parameter $p_w = T_w/2^{256}$. The number of hashes tried until a weak header is found is therefore *geometrically* distributed, and the time in seconds between two weak headers is approximately *exponentially* distributed with rate ηp_w, where η is the total hash rate per second and p_w is chosen such that $\eta p_w \approx 1/600$. When a weak header is found, it is also a strong block with probability p_s/p_w (where $p_s = T_s/2^{256}$), which is again a Bernoulli trial. Hence, the probability distribution of the number of weak headers found between two strong blocks is that of the number of trials before the first successful trial — as such, it also follows a geometric distribution, but with mean $p_w/p_s - 1$.[5] For example, for $T_w/T_s = 2^{10}$ this means that the average number of weak headers per block equals 1023. With 60 bytes per weak header (see Section 4.2) and 1MB per Bitcoin block, this would mean that the load increases by little over 6% on average with a small computational overhead introduced (see details in Section 7). The probability of having more than 16667 headers (or 1MB) in a block would equal.[6]

$$\left(1 - \frac{p_s}{p_w}\right)^{16668} = \left(1 - 2^{-10}\right)^{16668} \approx 8.4603 \cdot 10^{-8}.$$

Since around 51,000 Bitcoin blocks are found per year, this is expected to happen roughly once every 230 years.

5.1.2 Total Rewards

To ease the comparison to the Bitcoin protocol, we can enforce the same average mining reward per block (currently 12.5 BTC). Let R denote Bitcoin's mining reward. Since we reward weak headers as well as strong blocks, we need to scale all mining rewards by a constant c to ensure that the total reward remains unchanged — this is done in the *rewardBlock* function in Algorithm 1. As argued previously, we reward all weak headers equally by $\gamma R T_s/T_w$. Since the average number of weak headers per strong block is $T_w/T_s - 1$, this means that the expected total reward per block (i.e., strong block and weak header rewards) equals $cR + cR\gamma T_s/T_w \cdot (T_w/T_s - 1)$. Hence, we find that

$$c = \frac{1}{1 + \gamma(T_w/T_s - 1)T_s/T_w},$$

[5] Another way to reach this conclusion is as follows: the number of weak headers found in a fixed time interval is Poisson distributed, and it can be shown that the number of Poisson arrivals in an interval with exponentially distributed length is geometrically distributed.

[6] For an actual block implementation, we advice to introduce separate spaces for weak headers and transactions. With such a design, miners do not have incentives and trade-offs between including more transactions instead of weak headers.

which for large values of T_w/T_s is close to $1/(1+\gamma)$. This means that if $\gamma = 1$, the strong block and weak header rewards contribute almost equally to a miner's total reward.

5.2 Reward Variance of Solo Mining

The tendency towards centralization in Bitcoin caused by powerful mining pools can largely be attributed to the high reward variance of solo mining [15, 37]. Therefore, keeping the reward variance of a solo miner at a low level is a central design goal.

Let $\mathbf{R}^{BC}$ and $\mathbf{R}^{SC}$ be the random variables representing the per-block rewards for an α-strong solo miner in Bitcoin and in StrongChain, respectively. For any given strong block in both protocols, we define the random variable $\mathbf{I}$ as follows:

$$\mathbf{I} = \begin{cases} 1 & \text{the block is mined by the solo miner,} \\ 0 & \text{otherwise.} \end{cases}$$

By definition, $\mathbf{I}$ has a Bernoulli distribution, which means that $\mathbb{E}(\mathbf{I}) = \alpha$ and $\mathrm{Var}(\mathbf{I}) = \alpha(1-\alpha)$, where $\mathbb{E}$ and Var are the mean and variance of a random variable respectively. The following technical lemma will aid our analysis of the reward variances of solo miners:

Lemma 1. *Let $X_1, X_2, \ldots$ be independent and identically distributed random variables. Let N be defined on $\{0,1,\ldots\}$ and independent of $X_1, X_2, \ldots$. Let N and all X_i have finite mean and variance. Then*

$$\mathrm{Var}\left(\sum_{i=1}^{N} X_i\right) = \mathbb{E}(N)\mathrm{Var}(X) + \mathrm{Var}(N)(\mathbb{E}(X))^2.$$

Proof. See [7]. □

Reward Variance of Solo Mining in Bitcoin. Bitcoin rewards the miner of a block creator with the fixed block reward R and the variable (total) mining fees, which we denote by the random variable $\mathbf{F}$. Therefore, we have

$$\mathbf{R}^{BC} = \mathbf{I}(R + \mathbf{F}),$$

which implies that

$$\mathrm{Var}(\mathbf{R}^{BC}) = R^2\mathrm{Var}(\mathbf{I}) + \mathrm{Var}(\mathbf{IF}). \tag{5}$$

Since $\mathbf{IF} = \sum_{i=0}^{\mathbf{I}} \mathbf{F}$, we can use Lemma 1 (substituting $\mathbf{I}$ for N and $\mathbf{F}$ for X) to obtain

$$\mathrm{Var}(\mathbf{IF}) = \mathbb{E}(\mathbf{I})\mathrm{Var}(\mathbf{F}) + \mathrm{Var}(\mathbf{I})\mathbb{E}^2(\mathbf{F}). \tag{6}$$

Combining (5) and (6) gives

$$\begin{aligned} \mathrm{Var}(\mathbf{R}^{BC}) &= \mathbb{E}(\mathbf{I})\mathrm{Var}(\mathbf{F}) + \mathrm{Var}(\mathbf{I})\left(\mathbb{E}^2(\mathbf{F}) + R^2\right) \\ &= \alpha\mathrm{Var}(\mathbf{F}) + \alpha(1-\alpha)\left(\mathbb{E}^2(\mathbf{F}) + R^2\right). \end{aligned} \tag{7}$$

When the fees are small compared to the mining reward, this simplifies to $\alpha(1-\alpha)R^2$. By comparison, in [37] the variance of the block rewards (without fees) earned by a solo miner across a time period of t seconds is studied, and found to equal $\alpha R^2 t/600$.[7] The same quantity can be obtained by using (7), Lemma 1, and the total number of strong blocks found (by any miner) after t seconds of mining (which has a Poisson distribution with mean $t/600$).

Reward Variance of Solo Mining in StrongChain. For $\mathbf{R}^{SC}$, we assume that the solo miner has $\mathbf{N}$ weak headers included in the strong block, and that she obtains $c\gamma RT_s/T_w$ reward per weak header. Then the variance equals

$$\mathbf{R}^{SC} = \mathbf{I}(cR + \mathbf{F}) + c\gamma RT_s/T_w\mathbf{N},$$

where c is the scaling constant derived in Section 5.1.2. Hence, by applying Lemma 1, we compute the variance of $\mathbf{R}^{SC}$ as

$$\begin{aligned} \mathrm{Var}(\mathbf{R}^{SC}) &= (cR)^2\mathrm{Var}(\mathbf{I}) + \mathrm{Var}(\mathbf{IF}) \\ &\quad + (c\gamma RT_s/T_w)^2\mathrm{Var}(\mathbf{N}). \end{aligned} \tag{8}$$

The first term, which represents the variance of the strong block rewards, is similar to Bitcoin but multiplied by c^2. If we choose $T_w/T_s = 1024$ and $\gamma = 10$ (this choice is motivated later in this section), c^2 roughly equals 0.0083, which is quite small. Hence, the strong block rewards have a much smaller impact on the reward variance in our setting than in Bitcoin. The second term, which represents the variance of the fees, is precisely the same as for Bitcoin. The third term represents the variance of the weak header rewards, which in turn completely depends on $\mathrm{Var}(\mathbf{N})$.

To evaluate $\mathrm{Var}(\mathbf{N})$, we again use Lemma 1: let, for any weak header, $\mathbf{J}$ equal 1 if it is found by the solo miner, and 0 otherwise. Also, let $\mathbf{L}$ be the total number of weak headers found in the block, so including those not found by the solo miner. Then $\mathbf{N}$ is the sum of $\mathbf{L}$ instances of $\mathbf{J}$, where $\mathbf{J}$ has a Bernoulli distribution with success probability α (and therefore $\mathbb{E}(\mathbf{J}) = \alpha$ and $\mathrm{Var}(\mathbf{J}) = \alpha(1-\alpha)$), and $\mathbf{L}$ has a geometric distribution with success probability T_s/T_w (and therefore $\mathbb{E}(\mathbf{L}) = T_w/T_s - 1$ and $\mathrm{Var}(\mathbf{L}) = (T_w/T_s)^2 - T_w/T_s$. By substituting this into (8), we obtain:

$$\begin{aligned} \mathrm{Var}(\mathbf{N}) &= \mathbb{E}(\mathbf{L})\mathrm{Var}(\mathbf{J}) + \mathrm{Var}(\mathbf{L})(\mathbb{E}(\mathbf{J}))^2 \\ &= (T_w/T_s - 1)\alpha(1-\alpha) \\ &\quad + ((T_w/T_s)^2 - T_w/T_s)\alpha^2 \end{aligned} \tag{9}$$

Substituting (9) for $\mathrm{Var}(\mathbf{N})$ and $\alpha(1-\alpha)$ for $\mathrm{Var}(\mathbf{I})$ into (8) then yields an expression that can be evaluated for different values of T_w/T_s, γ, and α, as we discuss in the following.

[7] In particular, it is found to be $htR^2/(2^{32}D)$, where $h = \alpha\eta$ and $\eta/(2^{32}D) \approx 1/600$.

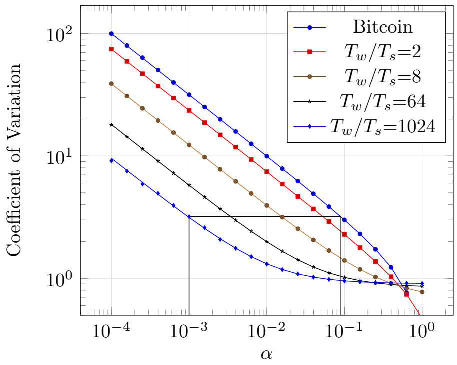


Figure 3: *Coefficients of variation* for the total rewards of α-strong miners for different strong/weak header difficulty ratios ($T_w/T_s = 1$ corresponds to Bitcoin). The lines indicate the exact results obtained using our analysis, whereas the markers indicate simulation results. We used $\gamma = \log_2(T_w/T_s)$. The black lines indicate that for $T_w/T_s = 1024$, a 0.1%-strong miner has a coefficient of variation that is comparable to a 9%-strong miner's in Bitcoin.

Comparison The difference between between (7) and (8) in practice is illustrated in Figure 3. This is done by comparing for a range of different values of α the block rewards' *coefficient of variation*, which is the ratio of the square root of the variance to the mean.

To empirically validate the results, we have also implemented a simulator in Java that can evaluate Bitcoin as well as StrongChain. We use two nodes, one of which controls a share α of the hash rate, and another controls a share $1-\alpha$. The nodes can broadcast information about blocks, although we abstract away from most of the other network behavior. We do not consider transactions (i.e., we mine empty blocks), and we use a simplified model for the propagation delays: delays are drawn from a Weibull distribution with shape parameter 0.6 [31], although for Figure 3 the mean was chosen to be negligible (more realistic values are chosen for Table 1).

The black lines in Figure 3 demonstrate that when $T_w/T_s = 1024$, a miner with share 0.1% of the mining power has the same coefficient of reward variation as a miner with stake 9% in Bitcoin. Also note that for $T_w/T_s = 1024$ and $\alpha \geq 1\%$, the coefficient of variation does not substantially decrease anymore, because nearly all of the reward variance is due to the number of weak headers. Hence, there would be fewer reasons for miners in our system to join large and cooperative mining pools, which has a positive effect on the decentralization of the system.

5.3 Chain Quality

One measure for the 'quality' of a blockchain is the stale rate of blocks [16], i.e., the percentage of blocks that appear during forks and do not make it onto the main chain. This is closely related to the notion of mining power utilization [10], which is the fraction of mining power used for non-stale blocks. In StrongChain, the stale rate of strong blocks may increase due to high latency. After all, while a new block is being propagated through the network, weak headers that strengthen the previous block that are found will be included by miners in their PoW calculation. As a result, some miners may refuse to switch to the new block when it arrives. However, the probability of this happening is very low: because each weak header only contributes T_s/T_w to the difficulty of a block, it would take on average 10 minutes to find enough weak headers to outweigh a block. As we can see in Table 1, the effect on the stale rate is negligible even for very high network latencies (i.e., 53 seconds). We also emphasize that the strong block stale rate is less important in our setting, as the losing miner still would benefit from her weak headers appended to the winning block.

Regarding the fairness, defined as the ratio between the observed share of the rewards (we simulate using one 10%-strong miner and a 90%-strong one) and the share of the mining power, we see that StrongChain does slightly worse than Bitcoin for high network latencies. The most likely cause is that due to the delay in the network, the 10%-strong miner keeps mining on a chain that has already been extended for longer than necessary. This gives the miner a slight disadvantage compared to the 90%-strong miner.

5.4 Security

One of the main advantages of StrongChain is the added robustness to selfish mining strategies akin to those discussed in [11] and [39]. In selfish mining, attackers aim to increase their share of the earned rewards by tricking other nodes into mining on top of a block that is unlikely to make it onto the main chain, thus wasting their mining power. This may come at a short-term cost, as the chance of the attacker's blocks going stale is increased — however, the difficulty rescale that occurs every 2016 blocks means that if the losses to the honest nodes are structural, the difficulty will go down and the gains of the attacker will increase.

In the following, we will consider the selfish mining strategy of [11],[8] described as follows:

- The attacker does not propagate a newly found block until she finds at least a second block on top of it, and then only if the difference in difficulty between her chain and the strongest known alternative chain is between zero and R.
- The attacker adopts the strongest known alternative chain if its difficulty is at least greater than her own by R.

[8] The 'stubborn mining' strategy of [39] offers mild improvements over [11] for powerful miners, but the comparison with StrongChain is similar. We have also modeled StrongChain using a Markov decision process, in a way that is similar to the recently proposed framework of [51]. Due to the state space explosion problem, we could only investigate the protocol with a small number of expected weak headers, but we have not found any strategies noticeably that are better than those presented.

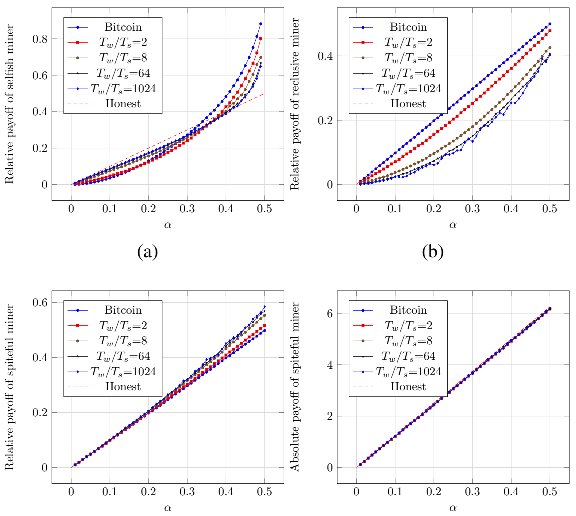
			StrongChain						
	Latency	**Bitcoin**	$\mathbf{T_w/T_s = 2}$	$\mathbf{T_w/T_s = 64}$			$\mathbf{T_w/T_s = 1024}$		
			$\gamma = 1$	$\gamma = 1$	$\gamma = 7$	$\gamma = 63$	$\gamma = 1$	$\gamma = 10$	$\gamma = 1023$
strong stale rate	low	.0023	.0025	.0021	.0026	.0028	.0023	.0025	.0019
	medium	.0073	.0082	.0087	.0077	.0078	.0084	.0067	.0081
	high	.0243	.0297	.0242	.0263	.0247	.0274	.0249	.0263
weak stale rate	low	—	.0043	.0047	.0049	.0046	.0049	.0047	.0047
	medium	—	.0142	.0151	.0154	.0149	.0145	.0147	.0149
	high	—	.0400	.0459	.0474	.0452	.0469	.0455	.0463
fairness	low	.9966	.9814	.9749	.9747	.9838	.9645	.9809	.9812
	medium	.9276	.9384	.9570	.9360	.9364	.9329	.9400	.9385
	high	.7951	.7640	.7978	.7820	.7757	.7756	.7766	.7775

Table 1: For several different protocols, the strong block stale rate, weak header rate, and the 'fairness' for an α-strong honest miner with $\alpha = 0.1$. Here, fairness is defined as the ratio between the observed share of the reward and the 'fair' share of the rewards (i.e, 0.1). 'Low', 'medium', and 'high' latencies refer to the mean of the delay distribution in the simulator; these are roughly 0.53 seconds, 5.3 seconds, and 53 seconds respectively. The simulations are based on a time period corresponding to roughly 20 000 blocks.

In Figure 4a, we have depicted the profitability of this selfish mining strategy for different choices of T_w/T_s. As we can see, for $T_w/T_s = 1024$ the probability of being 'ahead' after two strong blocks is so low that the strategy only begins to pay off when the attackers' mining power share is close to 45% — this is an improvement over Bitcoin, where the threshold is closer to 33%.

StrongChain does introduce new adversarial strategies based on the mining of new weak headers. Some examples include not broadcasting any newly found weak blocks ("reclusive" mining), refusing to include the weak headers of other miners ("spiteful" mining), and postponing the publication of a new strong block and wasting the weak headers found by other miners in the meantime. In the former case, the attacker risks losing their weak blocks, whereas in both of the latter two cases, the attacker risks their strong block going stale as other blocks and weak headers are found. Hence, these are not cost-free strategies. Furthermore, because the number of weak headers does not affect the difficulty rescale, the attacker's motive for increasing the stale rate of other miners' weak headers is less obvious (although in the long run, an adversarial miner could push other miners out of the market entirely, thus affecting the difficulty rescale).

In Figure 4b, we have displayed the relative payout (with respect to the total rewards) of a reclusive α-strong miner — this strategy does not pay for any $\alpha < 0.5$. In Figure 4c, we have depicted the relative payoff of a spiteful mine who does not include other miners' weak blocks unless necessary (i.e., unless others' weak blocks together contribute more than R to the difficulty, which would mean that any single block found by the spiteful miner would always go stale). For low latencies (the graphs were generated with an average latency of 0.53 seconds), the strategy is almost risk-free, and the attacker does manage to hurt other miners more than herself, leading to an increased relative payout. However, as displayed in Figure 4d, there are no absolute gains, even mild losses. As mentioned earlier, the weak headers do not affect the difficulty rescale so there is no short-term incentive to engage in this behavior — additionally there is little gain in computational overhead as the attacker still needs to process her own weak headers. In Section 6.1 we will discuss protocol updates that can mitigate these strategies regardless.



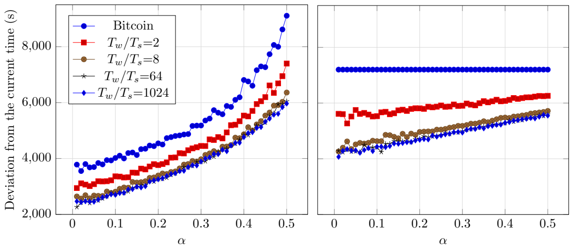
(a)

(b)

(c)

(d)

Figure 4: Payoffs of an α-strong adversarial miner for different strategies. Figure (a): relative payoff of a *selfish* miner following the strategy of [11], compared to an $(1-\alpha)$-strong honest miner. Figure (b): relative payoff of a *reclusive* miner who does not broadcast her weak blocks. Figure (c): *relative* payoff (with respect to the rewards of all miners combined) of a *spiteful* miner, who does not include other miners' weak blocks unless necessary. Figure (d): *absolute* payoff of a *spiteful* miner, with 12.5 BTC on average awarded per block. We consider Bitcoin and StrongChain with different choices of T_w/T_s, with $\gamma = \log_2(T_w/T_s)$.



(a) (b)

Figure 5: The deviation from the network time that an α-strong adversary can introduce for its mined blocks by slowing (the left graph) and accelerating (the right graph) timestamps.

5.5 More Reliable Timestamps

Finally, we conducted a series of simulations to investigate how the introduced redefinition of timestamps interpretation (see *getTimestamp()* in Algorithm 1 and Section 4.5) influences the timestamp reliability in an adversarial setting. We assume that an adversary wants to deviate blockchain timestamps by as much as possible. There are two strategies for such an attack, i.e., an adversary can either "slow down" timestamps or "accelerate" them. In the former attack, the best adversary's strategy is to use the minimum acceptable timestamp in every header created by the adversary. Namely, the adversary sets its timestamps to the median value of the last eleven blocks (a header with a lower timestamp would not be accepted by the network – see Section 2.2). As for the latter attack, the adversary can analogously bias timestamps towards the future by putting the maximum acceptable value in all her created headers. The maximum timestamp value accepted by network nodes is two hours in the future with respect to the nodes' internal clocks (any header with a higher timestamp would be rejected).

In our study, we assume that honest nodes maintain the network time which the adversary tries to deviate from. We consider the worst-case scenario, which is when the adversary, who also biases all her header timestamps, mines the strong block. We measure (over 10000 runs) how such a malicious timestamp can be mitigated by our redefinition of the block timestamps interpretation. We present the obtained results in Figure 5, and as shown in the slow-down case our protocol achieves much more precise timestamps than Bitcoin (the difference is around 2000 seconds). Similarly, when the adversary accelerates timestamps, our protocol can mitigate it effectively, adjusting the adversarial timestamps by 2000-3500 seconds towards the correct time. This effect is achieved due to the block's timestamp calculation as a weighted average of all block headers. The adversary could try to remove honest participants' weak headers in order to give a stronger weight to its malicious timestamps, but in Section 6.1 we discuss ways to mitigate it.

6 Discussion

6.1 Impact of the Parameter Choice

The results presented in Section 5 required several parameters to be fixed. First of all, we had to choose γ, which determines the relative contribution of the weak headers to the total mining rewards. Second, there is the contribution of the weak blocks to the chain difficulty, which in the *chainPoW()* function in Algorithm 1 was set to be only T_{max}/T_w. This means that the PoW of a weak header relative to a strong block's PoW — we call this the *difficulty factor* — is fixed to be T_s/T_w. In the following, we first discuss the relevant trade-offs and then motivate our choice.

When both γ and the difficulty factor are low, the impact on the reward variance of the miners (as per Figure 3) will be mild as the strong block rewards still constitute about 50% of the mining rewards. This reliance on the block rewards also means that 'spiteful' mining as discussed in Section 5.4 is disincentivized as the risk of strong blocks going stale still has a considerable impact on total rewards. However, selfish mining as proposed in [11] relies on several blocks in a row being mined in secret, and even for a low difficulty factor it becomes much harder for the attacker's chain to stay 'ahead' of the honest chain, as the latter accumulates strength from the weak headers at a faster rate. Hence, in this setting we only gain protection against selfish mining.

When γ is high but the difficulty factor is not (which is the setting of Section 5), then in addition to disincentivizing selfish mining, the reward variances become much less dependent on the irregular strong block rewards. This benefits small miners and reduces centralization, as we also discuss in Section 6.2. However, spiteful mining will have more of an impact as the possible downside (i.e., a latency-dependent increase in the strong block stale rate) will have less of an effect on the total rewards.

When both γ and the difficulty factor are high, the impact of spiteful mining is mitigated. The reason is that blocks quickly accumulate enough weak headers to outweigh a strong block, and in this case spiteful miners need to adopt the other weak blocks or risk their strong block becoming stale with certainty. The downside in this setting is that the system-wide block stale rate is increased. For example, if each weak header contributes $\gamma T_s/T_w$ to the difficulty and $\gamma = 10$, then after (on average) one minute enough weak headers are found to outweigh a strong block, and if propagation of the block takes longer than one minute then some miners will not adopt the block, increasing the likelihood of a fork.

In this paper, we have chosen the second of the three approaches — a moderately high γ, yet a minor difficulty factor. The reason is that the only downside (spiteful mining) was considered less of a concern than the other downsides (namely a low impact on reward variances and a higher block stale rate respectively) for two reasons: a) because spiteful mining does not lead to clear gains for the attacker, and b) because it only has a large impact on other miners' profits if the attacker controls a large share of the mining power, whereas the emergence of large mining pools is exactly what StrongChain discourages. The specific value of $\gamma = 10$ for $T_w/T_s = 1024$ (or $\gamma = \log_2(T_w/T_s)$ in general) was chosen to sufficiently reduce mining reward variances, yet leaving some incentive to discourage spiteful mining.

The protocol can be further extended to disincentivize spiteful mining, e.g., by additionally awarding strong block finders a reward that is proportional to the number of weak headers included. This would make StrongChain more similar to Ethereum, where stale block ('uncle') rewards are paid both to the miner of a stale block and the miner of the successful block that included it (see Section 8 for additional discussion of Ethereum's protocol). However, we leave such modifications and their consequences as future work.

6.2 StrongChain and Centralized Mining

Decentralized mining pools aim to reduce variance while providing benefits for the system (i.e., trust minimization for pools, and a higher number of validating nodes). However, mining in Bitcoin is in fact dominated by centralized mining pools whose value proposition, over decentralized pools, is an easy setup and participation. Therefore, rational miners motivated by their own benefit, instead of joining decentralized pools prefer centralized "plug-and-play" mining. It is still debatable whether centralized mining pools are beneficial or harmful to the system. However, it has been proved multiple times, that the concentration of significant computing power caused by centralized mining is risky and should be avoided, as such a strong pool has multiple ways of misbehaving and becomes a single point of failure in the system. One example is the pool GHash.IO, which in 2014 achieved more than 51% of the mining power. This undermined trust in the Bitcoin network to the extent that the pool was forced to actively ask miners to join other pools [12].

In order to follow incentives of rational miners, StrongChain does not require any radical changes from them and is compatible with centralized mining pools; however, it is specifically designed to mitigate their main security risk (i.e., power centralization). In StrongChain such pools could be much smaller than in Bitcoin (due to minimized variance) and to support this argument we conducted a study. We listed the largest Bitcoin mining pools and their shares in the global mining power (according to `https://www.blockchain.com/en/pools` as for the time of writing). Then for each pool, we calculated what would be the pool size in StrongChain to offer the miner the same payout variance experience, and the variance reduction factor in that case. As shown in Table 2, for the Bitcoin largest mining pool with 18.1% of the global hash rate, an equivalent pool in StrongChain (to provide miners the same reward experience) could be as small as 0.245% of the hash rate – around 74 times smaller. Even better reduction factors are achieved for smaller pools. Therefore, our study indicates that StrongChain makes the size of a pool almost an irrelevant factor for miners' benefits (i.e., there is no objective advantage of joining a large pool over a medium or a small one). Therefore we envision that with StrongChain, centralized mining pools will naturally be much more distributed.

Mining Pool	Pool Size		Size
	Bitcoin	StrongChain	Reduction
BTC.com	18.1%	0.245%	74×
F2Pool	14.1%	0.172%	82×
AntPool	11.7%	0.135%	87×
SlushPool	9.1%	0.099%	92×
ViaBTC	7.5%	0.079%	95×
BTC.TOP	7.1%	0.074%	96×
BitClub	3.1%	0.030%	103×
DPOOL	2.6%	0.025%	104×
Bitcoin.com	1.9%	0.018%	106×
BitFury	1.7%	0.016%	106×

Table 2: Largest Bitcoin mining pools and the corresponding pool sizes in StrongChain offering the same relative reward variance ($T_w/T_s = 1024$ and $\gamma = 10$).

Limitations

As discussed, it is beneficial for the system if as many participants as possible independently run full nodes; however, miners join large centralized pools not only due to high reward variance. Other potential reasons include the minimization of operational expenses as running a full node is a large overhead, higher efficiency since large pools may use high-performance hardware and network, better ability to earn extra income from merge mining [29], better protection against various attacks, anonymity benefits, etc. This work focuses on removing the reward variance reason. Although we believe that StrongChain would produce a larger number of small pools in a natural way, it does not eliminate the other reasons, so some large centralized pools may still remain. Luckily, our system is orthogonal to multiple concurrent solutions. For instance, StrongChain could be easily combined with non-outsourceable puzzle schemes (see Section 8) to increase the number of full nodes by explicitly disincentivizing miners from outsourcing their computing power. We leave such a combination as interesting future work.

7 Realization in Practice

We implemented our system in order to investigate its feasibility and confirm the stated properties. We implemented a StrongChain full node with interactive client in Python, and our implementation includes the complete logic from Algorithm 1 and all functionalities required to have a fully operational system (communication modules, message types, validation logic, etc...).[9] As described before, the main changes in our implementation to the Bitcoin's block layout are:

- a new (20B-long) *Coinbase* header field,
- a new binding transaction protecting all weak headers of the block,
- removed original coinbase transaction,

where a binding transaction has a single (32B-long) output as presented in Equation 3.[10]

Weak headers introduced by our system impact the bandwidth and storage overhead (when compared with Bitcoin). Due to compressing them (see Section 4.2), the size of a single weak header in a block is 60B. For example, with an average number of weak headers equal 1024, the storage and bandwidth overhead increases by about 61.5KB per block (e.g., with 64 weak headers, the overhead is only 3.8KB). Taking into account the average Bitcoin block size of about 1MB (the average between 15 Oct and 15 Nov 2018[11]), 1024 weak headers constitute around 6.1% of today's blocks, while 64 headers only 0.4%. The same overhead is introduced to SPV clients, that besides a strong header need to obtain weak headers and a proof for their corresponding binding transaction. Thus, an SPV update (every 10 minutes) would be 61.5KB or 3.8KB on average for 1024 or 64 weak headers, respectively. However, since only strong headers authenticate transactions, SPV clients do not need to store weak headers and after they are validated, they can remove them (they need to just calculate and associate their aggregated PoW with the strong header). Such an approach would not introduce any noticeable storage overhead on SPV clients.

Nodes validate all incoming weak headers; however, this overhead is a single hash computation and simple sanity checks per header. Even with our unoptimized implementation running on a commodity PC the total validation of a single weak header takes around 50μs on average (i.e., 51ms per 1024 headers on a single core). Given that we do not believe this overhead can lead to more serious denial-of-service attacks than ones already known and existing in Bitcoin (e.g., spamming with large invalid blocks). Additionally, StrongChain can adopt prevention techniques present in Bitcoin, like blacklisting misbehaving peers.

8 Related work

Employing weak solutions (and their variations) in Bitcoin is an idea [36,38] circulating on Bitcoin forums for many years. Initial proposals leverage weak solutions (i.e., *weak blocks*) for faster transaction confirmations [45,46], for signaling the current working branch of particular miners [13,14,30]. Unfortunately, most of these proposals come without necessary details or lack rigorous analysis. Below, we discuss the most related attempts that have been made to utilize weak or stale blocks in PoW-based decentralized consensus protocols. We compare these systems in Table 3 according to their reward and PoW calculation schemes.

Subchains. Rizun proposes Subchains [35], where a chain of weak blocks (a so-called subchain) bridging each pair of subsequent strong blocks is created. The design of Subchain puts a special focus on increasing the transaction throughput and the double-spend security for unconfirmed transactions. Rizun argues that since the (weak) block interval of subchains is much smaller than the strong block interval, it allows for faster (weak) transaction confirmations. Another claimed advantage of such an approach is that during the process of building subchains, the miners can detect forks earlier, and take actions accordingly to avoid wasting computational power. However, the design of Subchain sidesteps a concrete security analysis at the subchain level. In detail, by using a chaining data structure where one weak header referencing the previous weak header in a subchain, it introduces high stale rate on a subchain. More importantly, due to applying a Bitcoin-like subchain selection policy in case of conflicts, this approach is vulnerable to the selfish mining attack launched on a subchain.

Flux. Based on similar ideas as Subchain, Zamyatin et al. propose Flux [49]. In contrast to Subchain, Flux shares rewards (from newly minted coins and transaction fees) evenly among the finders of weak and strong blocks according to the computational resources they invested. This approach reduces the reward variance of miners, and therefore mitigates the need for large mining pools, which is beneficial for the system's decentralization. In addition, simulation experiments show that Flux renders selfish mining on the main chain less profitable. However, alike Subchains, Flux employs a chain structure for weak blocks, which inevitably introduces race conditions, increasing the stale rate of weak blocks and making it more susceptible to selfish mining attacks at the subchain level. The designers of Flux let both of these issues open and discuss the potential application of GHOST [41] to subchains. Another limitation of this work is that the authors do not analyze the requirements on space consumption when putting possibly a high number of overlapping transactions into Flux subchains, which could negatively influence network, storage, and processing resources.

Remarks on Subchain and Flux. One important difference between our approach and the above two designs is that we

[9] Our implementation is available at `https://github.com/ivan-homoliak-sutd/strongchain-demo/`.

[10] An alternative choice is to store a hash of weak headers in a header itself. Although simpler, that option would incur a higher overhead if the number of weak headers is greater than several.

[11] `https://www.blockchain.com/en/charts/avg-block-size`

	Bitcoin v0.1	Bitcoin	Fruitchains	Flux	StrongChain
Reward (strong)	$R+F$	$R+F$	0	$(R+F)/(E+1)$	$cR+F$
Reward (weak)	0	0	$(R+F)/E$	$(R+F)/(E+1)$	$c\gamma RT_s/T_w$
Chain weight contrib. (strong)	1	T_{max}/T_s	T_{max}/T_s	T_{max}/T_s	T_{max}/T_s
Chain weight contrib. (weak)	0	0	0	0	T_{max}/T_w

Table 3: The comparison of reward and PoW computation schemes of StrongChain and the related systems. (F: block transaction fees, E: expected number of weak headers per block. The entries for Flux are approximations for the PPLNS scheme in P2Pool, on which it is based.)

adopt a flat hierarchy for the weak blocks, which not only eliminates the possibility of selfish mining in a set of weak solutions, but also avoids the issue of stale rate of weak blocks. In contrast, both Subchain and Flux employ a chain structure for weak blocks, inevitably making them more susceptible to selfish mining attacks at the subchain level. Moreover, in our approach rewards are not shared, therefore miners can only benefit from appending received weak solutions. In addition, none of Subchain and Flux provide a concrete implementation demonstrating their applicability.

FruitChains. Another approach to address the mining variance and selfish mining issues is the FruitChains protocol proposed by Pass and Shi [32]. In FruitChains, instead of directly storing the records inside blocks, the records or transactions are put inside "fruits" with relatively low mining difficulties. Fruits then are appended to a blockchain via blocks which are mined with a higher difficulty. Mined fruits and blocks yield rewards, hence, miners can be paid more often and there is no need to form a mining pool.

However, some practical and technical details of FruitChains lead to undesired side effects. First, the scheme allows fruits with small difficulties to be announced and accepted by other miners. With too small difficulty it could render high transmission overheads or even potential denial-of-service attacks and its effects on the network are not investigated. On the other hand, too high fruit difficulty could result in a low transaction throughput and a high reward variance. Second, duplicate fruits are discarded, even though they might be found by different miners – this naturally implies some stale fruit rate (uninvestigated in the paper). Similarly, it is unclear would a block finder have an incentive to treat all fruits equally and to not prioritize her mined fruits (especially when fruits are associated with transaction fees). Moreover, fruits that are not appended to the blockchain quickly enough have to be mined and broadcast again, rendering additional overheads. Finally, the description of FruitChains lacks important details (e.g., the size of the fruits or the overheads introduced) as well as an actual implementation.

GHOST and Ethereum. An alternative approach for decreasing a high reward variance of miners is to shorten the block creation rate to the extent that does not hurt the overall system security – such an approach increases transaction throughput as well. The Greedy Heaviest-Observed Sub-Tree (GHOST) chain selection rule [41] makes use of stale blocks to increase the weight of their ancestors, which achieves a 600 fold speedup for the block generation compared to Bitcoin, while preserving its security strength. Despite the inclusion of stale blocks in the blockchain, only the miners of the main chain get rewards for the inclusion of the stale blocks.

In contrast to the original GHOST, the white and yellow papers of Ethereum [44, 48] propose to reward also miners of stale blocks in order to further increase the security – not wasting with the consumed resources for mining of stale blocks. However, Ritz and Zugenmaier shows that rewarding so called "uncle blocks" lowers the threshold at which selfish mining is profitable [34] – a selfish miner can built-up the "heaviest" chain, as she can reference blocks previously not broadcast to the honest network. Likewise, the inclusive blockchain protocol [20], which increases the transaction throughput, but leaves the selfish mining issue unsolved.

DAG-based Protocols. SPECTRE [40] is an example of the protocols family that leverages a directed acyclic graph (DAG). This family proposed more radical design changes motivated by the observation that one essential throughput limitation of Nakamoto consensus is the data structure leveraged which can be maintained only sequentially. SPECTRE generalizes the Nakamoto's blockchain into a DAG of blocks, while allowing miners to add blocks concurrently with a high frequency, just basing on their individual current views of the DAG. Such a design provides multiple advantages over chain-based protocols including StrongChain. Frequently published blocks increase transaction throughput and provide fast confirmation times while relaxed consistency requirements allow to tolerate propagation delays. Like StrongChain, SPECTRE also aims to decrease mining reward variance, but achieves it again via frequent blocks. However, frequent blocks have a side effect of transaction redundancy which negatively impacts the storage and transmission overheads, and this aspect was not investigated. Moreover, SPECTRE provides a weaker property than chain-based consensus protocols as simultaneously added transactions cannot be ordered. This and schemes following a similar design are payments oriented and to support order-specific applications, like smart contracts, they need to be enhanced with an additional ordering logic.

More recently, Sompolinsky and Zohar [42] proposed two DAG-based protocols improving the prior work. PHANTOM introduces and uses a greedy algorithm (called the

GHOSTDAG protocol) to determine the order of transactions. This eliminates the applicability issues of SPECTRE, but for the cost of slowing down transaction confirmation times. Combining advantages of PHANTOM and SPECTRE into a full system was left by the authors as a future work.

Decentralization-oriented Schemes. Mining decentralization was a goal of multiple previous proposals. One direction is to design mining such that miners do not have incentive to outsource resources or forming coalitions. Permacoin [25] is an early attempt to achieve it where miners instead of proving their work prove that their store (fragments of) a globally-agreed file. Permacoin is designed such that: a) payment private keys are bound to puzzle solutions – outsourcing private keys is risky for miners, b) sequential and random storage access is critical for the mining efficiency, thus it disincentives miners from outsourcing data. If the file is valuable, then a side-effect of Permacoin is its usefulness, as miners replicate the file.

The notion of non-outsourceable mining was further extended and other schemes were proposed [26, 50]. Miller et al. [26] introduces "strongly non-outsourceable puzzles" that aim to disincentivize pool creation by requiring all pool participants to remain honest. In short, with these puzzles any pool participant can steal the pool reward without revealing its identity. The scheme relies on zero knowledge proofs requiring a trusted setup and introducing significant computational overheads. The scheme is orthogonal to StrongChain and could be integrated with easily integrated with StrongChain, however, after a few years of their introduction no system of this class was actually deployed at scale; thus, we do not have any empirical results confirming their promised benefits.

SmartPool is a different approach that was proposed by Luu et al. [23]. In SmartPool, the functionality of mining pools was implemented as a smart contract. Such an approach runs natively only on smart-contract platforms but it allows to eliminate actual mining pools and their managers (note that SmartPool still imposes fees for running smart contracts), while preserving most benefits of pool mining.

Rewarding Schemes for Mining Pools. Mining pools divide the block reward (together with the transaction fees) in such a way that each miner joining the pool is paid his fair share in proportion to his contribution. Typically, the contribution of an individual miner in the pool is witnessed by showing weak solutions called *shares*.

There are various rewarding schemes that mining pools employ. The simplest and most natural method is the proportional scheme where the reward of a strong block is divided in proportion to the numbers of shares submitted by the miners. However, this scheme leads to pool hopping attacks [33]. To avoid this security issue, many other rewarding systems are developed, including the Pay-per-last-N-shares (PPLNS) scheme and its variants. We refer the reader to [37] for a systematic analysis of different pool rewarding systems.

The reward mechanisms in StrongChain can be seen conceptually as a mining pool built-in into the protocol. However, there are important differences between our design and the mining pools. The most contrasting one is that in StrongChain rewarding is not a zero-sum game and miners do not share rewards. In mining pools, all rewards are shared and this characteristic causes multiple in- and cross-pool attacks that cannot be launched against our scheme. Furthermore, the miner collaboration within Bitcoin mining pools is a "necessary evil", while in StrongChain the collaboration is beneficial for miners and the overall system. We discuss StrongChain and mining pools further in Section 6.2.

9 Conclusions

In this paper, we proposed a transparent and collaborative proof-of-work protocol. Our approach is based on Nakamoto consensus and Bitcoin, however, we modified their core designs. In particular, in contrast to them, we take advantage of weak solutions, which although they do not finalize a block creation positively contribute to the blockchain properties. We also proposed a rewarding scheme such that miners can benefit from exchanging and appending weak solutions. These modifications lead to a more secure, fair, and efficient system. Surprisingly, we show that our approach helps with seemingly unrelated issues like the freshness property. Finally, our implementation indicates the efficiency and deployability of our approach.

Incentives-oriented analysis of consensus protocols is a relatively new topic and in the future we would like to extend our work by modeling our protocol with novel frameworks and tools. Another topic worth investigating in future is how to combine StrongChain with systems solving other drawbacks of Nakamoto consensus [10, 19, 21], or how to mimic the protocol in the proof-of-stake setting.

Acknowledgment

We thank the anonymous reviewers and our shepherd Joseph Bonneau for their valuable comments and suggestions.

This work was supported in part by the National Research Foundation (NRF), Prime Minister's Office, Singapore, under its National Cybersecurity R&D Programme (Award No. NRF2016NCR-NCR002-028) and administered by the National Cybersecurity R&D Directorate, and by ST Electronics and NRF under Corporate Laboratory @ University Scheme (Programme Title: STEE Infosec - SUTD Corporate Laboratory).

References

[1] L. Bahack. Theoretical Bitcoin attacks with less than half of the computational power (draft). *arXiv preprint*

arXiv:1312.7013, 2013.

[2] D. Bayer, S. Haber, and W. S. Stornetta. Improving the efficiency and reliability of digital time-stamping. In *Sequences II*. Springer, 1993.

[3] A. Boverman. Timejacking & Bitcoin. `https://culubas.blogspot.sg/2011/05/timejacking-bitcoin_802.html`, 2011.

[4] M. Carlsten, H. A. Kalodner, S. M. Weinberg, and A. Narayanan. On the instability of Bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[5] N. T. Courtois and L. Bahack. On subversive miner strategies and block withholding attack in Bitcoin digital currency. *arXiv preprint arXiv:1402.1718*, 2014.

[6] J. R. Douceur. The Sybil attack. In *International workshop on peer-to-peer systems*. Springer, 2002.

[7] S. Dunbar. Random sums of random variables. `http://www.math.unl.edu/~sdunbar1/ProbabilityTheory/Lessons/Conditionals/RandomSums/randsum.shtml`.

[8] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*. Springer, 1992.

[9] I. Eyal. The miner's dilemma. In *2015 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2015.

[10] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse. Bitcoin-NG: A scalable blockchain protocol. In *Proceedings of NSDI*, 2016.

[11] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*. Springer, 2014.

[12] C. Farivar. Bitcoin pool GHash.io commits to 40% hashrate limit after its 51% breach. `https://arstechnica.com/information-technology/2014/07/bitcoin-pool-ghash-io-commits-to-40-hashrate-limit-after-its-51-breach/`, 2014.

[13] Gavin Andresen. Faster blocks vs bigger blocks. `https://bitcointalk.org/index.php?topic=673415.msg7658481#msg7658481`, 2014.

[14] Gavin Andresen. Weak block thoughts. `https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2015-September/011157.html`, 2015.

[15] A. E. Gencer, S. Basu, I. Eyal, R. van Renesse, and E. G. Sirer. Decentralization in Bitcoin and Ethereum networks. *arXiv preprint arXiv:1801.03998*, 2018.

[16] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016.

[17] A. Gervais, H. Ritzdorf, G. O. Karame, and S. Capkun. Tampering with the delivery of blocks and transactions in Bitcoin. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015.

[18] G. O. Karame, E. Androulaki, and S. Capkun. Double-spending fast payments in Bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012.

[19] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, 2016.

[20] Y. Lewenberg, Y. Sompolinsky, and A. Zohar. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security*. Springer, 2015.

[21] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016.

[22] L. Luu, R. Saha, I. Parameshwaran, P. Saxena, and A. Hobor. On power splitting games in distributed computation: The case of Bitcoin pooled mining. In *Computer Security Foundations Symposium (CSF), 2015 IEEE 28th*. IEEE, 2015.

[23] L. Luu, Y. Velner, J. Teutsch, and P. Saxena. Smartpool: Practical decentralized pooled mining. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, 2017.

[24] R. C. Merkle. A digital signature based on a conventional encryption function. In *Proceedings of Advances in Cryptology*, 1988.

[25] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz. Permacoin: Repurposing Bitcoin work for data preservation. In *2014 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2014.

[26] A. Miller, A. Kosba, J. Katz, and E. Shi. Nonoutsourceable scratch-off puzzles to discourage Bitcoin mining coalitions. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015.

[27] M. Möser and R. Böhme. Trends, tips, tolls: A longitudinal study of bitcoin transaction fees. In *Financial Cryptography Workshops*, 2015.

[28] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[29] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder. *Bitcoin and cryptocurrency technologies: A comprehensive introduction*. Princeton University Press, 2016.

[30] T. Nolan. Distributing low POW headers. `https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-July/002976.html`, 2013.

[31] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, and C. Diot. Measurement and analysis of single-hop delay on an IP backbone network. *IEEE Journal on Selected Areas in Communications*, 21(6), 2003.

[32] R. Pass and E. Shi. Fruitchains: A fair blockchain. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*. ACM, 2017.

[33] Raulo. Optimal pool abuse strategy. `http://bitcoin.atspace.com/poolcheating.pdf`, 2011.

[34] F. Ritz and A. Zugenmaier. The impact of uncle rewards on selfish mining in Ethereum. *arXiv preprint arXiv:1805.08832*, 2018.

[35] P. R. Rizun. Subchains: A technique to scale Bitcoin and improve the user experience. *Ledger*, 1, 2016.

[36] K. Rosenbaum. Weak Blocks – The Good And The Bad. `http://popeller.io/index.php/2016/01/19/weak-blocks-the-good-and-the-bad/`, 2016.

[37] M. Rosenfeld. Analysis of Bitcoin pooled mining reward systems. *arXiv preprint arXiv:1112.4980*, 2011.

[38] R. Russell. Weak block simulator for Bitcoin. `https://bitcointalk.org/index.php?topic=179598.0`, 2017.

[39] A. Sapirshtein, Y. Sompolinsky, and A. Zohar. Optimal selfish mining strategies in Bitcoin. In *International Conference on Financial Cryptography and Data Security*. Springer, 2016.

[40] Y. Sompolinsky, Y. Lewenberg, and A. Zohar. SPECTRE: Serialization of proof-of-work events: confirming transactions via recursive elections, 2016.

[41] Y. Sompolinsky and A. Zohar. Accelerating Bitcoin's transaction processing. *Fast Money Grows on Trees, Not Chains*, 2013.

[42] Y. Sompolinsky and A. Zohar. PHANTOM, GHOSTDAG: Two scalable BlockDAG protocols. Cryptology ePrint Archive, Report 2018/104, 2018. `https://eprint.iacr.org/2018/104`.

[43] P. Szalachowski. (short paper) towards more reliable Bitcoin timestamps. In *Proceedings of the Crypto Valley Conference on Blockchain Technology (CVCBT)*, 2018.

[44] E. team. A Next-Generation Smart Contract and Decentralized Application Platform. `https://github.com/ethereum/wiki/wiki/White-Paper#modified-ghost-implementation`, 2018.

[45] TierNolan (Pseudonymous). Decoupling Transactions and PoW. `https://bitcointalk.org/index.php?topic=179598.0`, 2013.

[46] P. Todd. Near-block broadcasts for proof of tx propagation. `https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-September/003275.html`, 2013.

[47] I. Tsabary and I. Eyal. The gap game. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018.

[48] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151, 2014.

[49] A. Zamyatin, N. Stifter, P. Schindler, E. Weippl, and W. J. Knottenbelt. Flux: Revisiting near blocks for proof-of-work blockchains. Cryptology ePrint Archive, Report 2018/415, 2018. `https://eprint.iacr.org/2018/415/20180529:172206`.

[50] G. Zeng, S. M. Yiu, J. Zhang, H. Kuzuno, and M. H. Au. A nonoutsourceable puzzle under GHOST rule. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 2017.

[51] R. Zhang and B. Preneel. Lay down the common metrics: Evaluating proof-of-work consensus protocols' security. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019.

Incentive Attacks on DAG-Based Blockchains with Random Transaction Selection

Martin Perešíni*✉ Federico Matteo Benčić[†] Martin Hrubý* Kamil Malinka* Ivan Homoliak*✉
iperesini@fit.vut.cz federico-matteo.bencic@fer.hr hrubym@fit.vut.cz malinka@fit.vut.cz ihomoliak@fit.vut.cz

*Brno University of Technology,
Faculty of Information Technology

[†]University of Zagreb,
Faculty of Electrical Engineering and Computing

***Abstract*—Several blockchain consensus protocols proposed to use of Directed Acyclic Graphs (DAGs) to solve the limited processing throughput of traditional single-chain Proof-of-Work (PoW) blockchains. Many such protocols utilize a random transaction selection (*RTS*) strategy (e.g., PHANTOM, GHOSTDAG, SPECTRE, Inclusive, and Prism) to avoid transaction duplicates across parallel blocks in DAG and thus maximize the network throughput. However, previous research has not rigorously examined incentive-oriented greedy behaviors when transaction selection deviates from the protocol. In this work, we first perform a generic game-theoretic analysis abstracting several DAG-based blockchain protocols that use the RTS strategy, and we prove that such a strategy does not constitute a Nash equilibrium. Next, to validate the conclusions from game theoretic analysis, we perform simulations, confirming that greedy actors who do not follow the RTS strategy can profit more than honest miners and harm the processing throughput of the protocol. Finally, we show that greedy miners are incentivized to form a shared mining pool to increase their profits. This undermines the decentralization and degrades the design of the protocols in question.**

I. Introduction

Blockchains inherently suffer from the processing throughput bottleneck, as consensus must be reached for each block within the chain. One approach to solve this problem is to increase the block creation rate. However, such an approach has drawbacks. If blocks are not propagated through the network before a new block is created, a *soft fork* might occur, in which two concurrent blocks reference the same parent block. A soft fork is resolved in a short time by a fork-choice rule, and thus only one block is eventually accepted. All transactions in an *orphaned* (a.k.a., stale) block are discarded. As a result, consensus nodes that created orphaned blocks wasted their resources and did not get rewarded.

As a response to the above issue, several proposals (e.g., Inclusive [17], PHANTOM [34], GHOSTDAG [34], SPECTRE [33]) have substituted a single chaining data structure for (unstructured) Directed Acyclic Graphs (DAGs) (see Fig. 1), while another proposal in this direction employed structured DAG (i.e., Prism [5]). Such a structure can maintain multiple interconnected chains and thus theoretically increase processing throughput. The assumption of concerned DAG-oriented solutions is to abandon transaction selection purely based on the highest fees since this approach intuitively increases the probability that the same transaction is included in more than one block (hereafter *transaction collision*). Instead, these approaches use the random transaction selection (i.e., RTS) [1] strategy as part of the consensus protocol to avoid transaction collisions. Although the consequences of deviating from such a strategy might seem intuitive, no one has yet thoroughly analyzed the performance and robustness of concerned DAG-oriented approaches within an empirical study investigating incentive attacks on transaction selection.

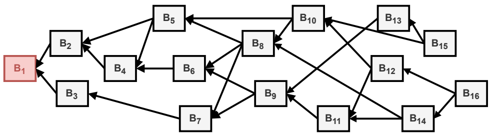


Fig. 1: A structure of DAG-oriented blockchain.

In this work, we focus on the impact of **greedy**[2] actors in several DAG-oriented designs of consensus protocols. In particular, we study the situation where a greedy attacker (or attackers) deviates from the protocol by not following the RTS strategy that is assumed by a few DAG-oriented approaches [17], [34], [34], [33], [5]. Out of these approaches, PHANTOM [34], GHOSTDAG, [34], and SPECTRE [33] utilize RTS that was introduced in Inclusive [17] – whose game theoretic analysis (and missing assumption about creating a mining pool) we contradict in this work. In contrast, Prism [5] does not provide any incentive-oriented analysis and thus did not show that it is resistant to any incentive attacks based on transaction selection. Nevertheless, both lines of works employ RTS and thus enable us to abstract their details and focus on modeling and analysis of this aspect.

We make a hypothesis stating that the attacker deviating from RTS strategy might earn greater rewards as compared to honest participants, and moreover, such an attacker harms transaction throughput since transaction collision is increased. We verify and prove our hypothesis in a game theoretical analysis and show that RTS does not constitute Nash equilib-

This work was supported by HungryEcoCities project, funded from the EU's Horizon Europe research and innovation programme (101069990), the internal project of BUT (FIT-S-23-8151), Croatian Science Foundation (IP-2019-04-1986), and Cybersecurity Innovation Hub, Digital Europe Programme (101083932). Resources were provided by the e-INFRA CZ (ID:90254).

[1] Note that RTS involves a certain randomness in transaction selection but does not necessarily equals to uniform random transaction selection (to be in line with the works utilizing Inclusive [17], such as PHANTOM, GHOSTDAG [34], SPECTRE [33], as well as the implementation of GHOSTDAG called Kaspa [32]).

[2] Greedy actors deviate from the protocol to increase their profits.

rium.[3] Next, we substantiate conclusions from game theoretical analysis by a few simulation experiments, where we focus on an abstracted DAG-PROTOCOL, inspired by existing designs.

Contributions. The contributions of this work are as follows:

1.) We hypothesize that not following the RTS strategy in concerned DAG-based protocols negatively affects the relative profit of honest miners and the throughput of the network.
2.) We validate the hypothesis by a game theory approach, concluding that the RTS does not constitute Nash equilibrium.
3.) We adapted open-source simulation tools to consider multiple chains and various incentive schemes, and thus enable us to investigate properties of concerned DAG-based protocols.
4.) We execute experiments on an abstracted DAG-PROTOCOL, which confirm that a greedy actor who selects transactions based on the highest fee profits significantly more than honest miners following the RTS.
5.) We demonstrate that multiple greedy actors can significantly reduce the effective transaction throughput by increasing the transaction collision rate across parallel chains of DAGs.
6.) We show that greedy actors have an incentive to form a mining pool to increase their relative profits, which degrades the decentralization of the concerned DAG-oriented designs.

II. BACKGROUND

We assume that the reader is familiar with the basic terminology related to Nakamoto consensus [23], its rewarding scheme, PoW puzzle, and mempool. We refer unfamiliar readers to the extended version of our paper [28].

Block Creation Time. In Bitcoin, there is a default block creation time λ set to create a new block every 10 minutes on average. This parameter is derived directly from the network difficulty, which changes over time, and it is adjusted every 2016 blocks to fit the target value of 10 minutes. According to Gervais et al., [10], the stale block rate of Bitcoin is 0.41%. Other sources [7], [12] state the values $0.5-1\%$, which is considered negligible. We assume that the mathematical model corresponding to λ of Bitcoin is an exponentially distributed random variable with the time between two consecutive blocks:

$$f^{\mathbb{T}}(t) = \Lambda e^{-\Lambda t}, \tag{1}$$

where $\Lambda = \frac{1}{\lambda}$ [6], [13] and t is time in seconds. Thus, we model the creation of blocks by a Poisson process with λ.

III. PROBLEM DEFINITION

Let there be a PoW blockchain network that uses the Nakamoto consensus (NC) and consists of honest and greedy miners, with the greedy miners holding a fraction κ of the total mining power (i.e., adversarial mining power). Then, we denote the network propagation delay in seconds as τ and the block creation time in seconds as λ. We assume that the minimum value of λ is constrained by τ of the blockchain network. It is well-known that Nakamoto-style blockchains generate stale blocks (a.k.a., orphan blocks). As a result, a fraction of the mining power is wasted. The rate at which stale blocks are generated increases when λ is decreased, which is one of the reasons why Bitcoin maintains a high λ of 600s.

DAG-Oriented Designs. Many DAG-oriented designs were proposed to allow a decrease of λ while utilizing stale blocks in parallel chains, which should increase the transaction throughput. Although there are some DAG-oriented designs that do not address the problem of increasing transaction throughput (e.g., IoTA [29], Nano [16], Byteball [3]), we focus on the specific group of solutions addressing this problem, such as Inclusive [17], GHOSTDAG, PHANTOM [34], SPECTRE [33], and Prism [5]. We are targeting the RTS strategy, which is a common property of this group of protocols. In the RTS, the miners do not take into account transaction fees of all included transactions; instead, they select transactions to blocks randomly – although not necessarily uniformly at random (e.g., [32]). In this way, these designs aim to eliminate transaction collision within parallel blocks of the DAG structure. Nevertheless, the interpretation of randomness in RTS is not enforced/verified by these designs, and miners are trusted to ignore fees of all (or the majority (e.g., [32]) of) transactions for the common "well-being" of the protocol. Contrary, miners of blockchains such as Bitcoin use a well-known transaction selection mechanism that maximizes profit by selecting transactions of the block based on the highest fees – we refer to this strategy as the *greedy strategy* in this work.

A. Assumptions

We assume a generic DAG-oriented consensus protocol using the RTS strategy (denoted as DAG-PROTOCOL). Then, we assume that the incentive scheme of DAG-PROTOCOL relies on transaction fees (but additionally might also rely on block rewards),[4] and transactions are of the same size.[5] Let us assume that the greedy miners may only choose a different transaction selection strategy to make more profit than honest miners. Then, we assume that DAG-PROTOCOL uses rewarding where the miner of the block A gets rewarded for all unique not-yet-mined transactions in A (while she is not rewarded for transaction duplicates mined before).

B. Identified Problems – Incentive Attacks

Although the assumptions stated above might seem intuitive, there is no related work studying the impact of greedy miners deviating from the RTS strategy on any of the considered DAG-PROTOCOLS ([34],[33],[17],[5]) and the effect it might have on the throughput of these protocols as well as a fair distribution of earned rewards. Note that we assume GHOSTDAG, PHANTOM, and SPECTRE are utilizing the RTS strategy that was proposed in the Inclusive protocol [33], as recommended by the (partially overlapping) authors of these works – this is further substantiated by the practical implementation of GHOSTDAG/PHANTOM called Kaspa [32], which utilizes a variant of RTS strategy (see Sec. IV) that selects a majority portion of transactions in a block uniformly random, while a small portion of the block capacity is seized by the transaction selected based on the highest fees. Nevertheless, besides potentially increased transaction collision rate, even

[3] Note that we empirically validated a similar hypothesis on PHANTOM (and its extension GHOSTDAG) only in an earlier version of this paper [26].

[4] Note that block rewards would not change the applicability of our incentive attacks, and the constraints defined in the game theoretic model (see Sec. V-B) would remain met even with them.

[5] Note that this assumption serves only for simplification of the follow-up sections. Transactions of different sizes would require normalizing fees by the sizes of transactions to obtain an equivalent setup (i.e., a fee per Byte).

such an approach enables more greedy behavior. We make a hypothesis for our incentive attacks:

Hypothesis 1. *A greedy transaction selection strategy will decrease the relative profit of honest miners as well as transaction throughput in the* DAG-PROTOCOL.[6]

IV. DAG-ORIENTED SOLUTIONS (DAG-PROTOCOLS)

Inclusive Protocol. Lewenberg et al. [17] proposed a new way to structure the chain that can operate at much faster rates than Bitcoin. The authors utilize the DAG to form blocks in a structure called the *blockDAG*. This structure is created by allowing blocks to reference multiple previous blocks, enabling less strict transaction inclusion rules that can potentially store conflicting transactions in parallel blocks due to allowing $\lambda < \tau$. This means that the system can process larger blocks faster than is possible to gossip within the bounds of τ, increasing transaction throughput. The authors propose the protocol as a building block for other DAG-oriented protocols, and they claim that they reduce the advantage of highly connected miners in single-chain protocols since even stale blocks (of a single-chain) are included. Further, the authors present the key concept of *randomly selecting* transactions (i.e., RTS) to avoid collisions. The authors theoretically analyze this assumption by modeling the protocol and its transaction selection as a game, in which rational miners opt to avoid collisions. According to the authors, the game's outcome is a sequential equilibrium, where the growing fraction of greedy miners causes a decrease in their profits, which should make such a strategy less attractive (we show this phenomenon in Fig. 3a). However, the authors do not assume that the miners can create a mining pool, in which they can achieve significantly higher profits than honest miners (we demonstrate it in Fig. 3b).

PHANTOM. The PHANTOM protocol [34] is a generalization of the NC's longest-chain protocol. While in NC each block contains a hash of the previous block in the chain it extends, PHANTOM organizes blocks in a DAG. As a result, each block may contain multiple hash references to predecessors, like in Inclusive [17]. The key contribution of PHANTOM is that it totally orders all blocks by solving *the maximum k-cluster SubDAG problem*, which utilizes the concept of the main chain and the distance from it. Unlike NC which discards the blocks out of the main chain (i.e., orphan blocks), PHANTOM includes these blocks in a DAG, except for the attacker-created blocks that would be weakly connected. PHANTOM uses the RTS strategy proposed by the (partially overlapping) authors of the Inclusive protocol. The incentive scheme of PHANTOM revolves around rewarding all miners who include a transaction within a new block A, while assuming that transactions in the parallel blocks are unique and due to a DAG will not be discarded as in single-chain blockchains. If there are some duplicate transactions, PHANTOM rewards them only once – in the first block that includes them, which is evaluated after establishing the total ordering. However, such an incentive scheme must be constructed with care, as sidechain blocks might also be the result of an attack. Therefore, the reward a miner receives for publishing A is indirectly proportional to the discretized delay at which A was referenced by the main chain. For this reason, the protocol defines a measure of the delay in publishing A w.r.t. the main chain, called the *gap parameter* c. The value by which the reward is "decayed" is determined by the discount function γ, where $\gamma(c(A)) \in [0,1]$ and γ is weakly decreasing.[7] Finally, the miner is rewarded for including transactions in A using the *payoff function*. In detail, the miner gets rewarded for all non-duplicate transactions contained in A, and after γ was applied to the respective transaction fees.

GHOSTDAG. PHANTOM is considered impractical for efficient use [34], because it requires the solution of an NP-hard problem (the maximum k-cluster SubDAG problem). Therefore, the authors of PHANTOM have developed a greedy (heuristic) algorithm to find block clusters, obtaining the GHOSTDAG protocol. This protocol uses greedy ordering of the DAG, which has practical advantages.

Kaspa. The RTS strategy is utilized even in the already running blockchain Kaspa [32], which is the implementation of the GHOSTDAG protocol. Kaspa selects transactions using a variant of the RTS strategy, in which a small fraction of a block is dedicated to prioritized transactions with higher fees and remaining part of a block serves for transactions selected uniformly at random. We argue that even this approach is vulnerable to our incentive attacks since the part of the block relying on uniformly random selection cannot be enforced/verified, and thus miners might still prioritize transactions with higher fees, which can consequently result in throughput problems and incentive attacks. Nevertheless, the current Kaspa mainnet is not saturated, and its blocks usually contain only 1 to 5 transactions,[8] not fully utilizing the concept of DAG for increased throughput.

Prism. Prism [5] is a protocol that aims to achieve a total ordering of transactions with consistency and liveness guarantees while achieving high throughput and low latency. Prism differs from traditional single-chain blockchains since it involves a few parallel chains rather than a single chain. It decouples transaction confirmation, validation, and proposal. Prism replaces traditional blocks with (1) transaction blocks (i.e., blocks that contain transactions), (2) voter blocks (i.e., blocks that vote for proposer blocks), and (3) proposer blocks (i.e., blocks that reference transaction blocks). The authors of Prism recognize that blocks mined in parallel chains might contain duplicate transactions. To cope with it, they propose to randomly divide unprocessed transactions of the local mempool into multiple queues and then create blocks using transactions only from one randomly selected queue, which is a variant of RTS strategy and thus enables incentive attacks based on greedy strategy. However, the authors do not provide any analysis related to such incentive attacks.

V. GAME THEORETICAL ANALYSIS

In this section, we model a DAG-PROTOCOL[9] as a two-player game, in which the honest player/phenotype (P_{hon}) uses

[6] Note that the greedy transaction selection strategy deviates from the DAG-PROTOCOL and thus is considered adversarial.

[7] I.e., later inclusion of the side-chain block imposes lower reward.

[8] https://explorer.kaspa.org/

[9] Note that we consider DAG-based designs (described in Sec. IV) under this generic term of DAG-PROTOCOLS to simplify the description but not to claim that all DAG-PROTOCOLS (with RTS) can be modeled as we do.

the RTS strategy and the greedy player/phenotype (P_{grd}) uses the greedy transaction selection strategy. We assume that the fees of transactions vary – the particular variance of fees is agnostic to this analysis. We present the game theoretical approach widely used to analyze interactions of players (i.e., consensus nodes) in the blockchain. Several works attempted to study the outcomes of different scenarios in blockchain networks (e.g., [19], [38], [30]) but none of them addressed the case of DAG-PROTOCOLS and their transaction selection. In game theoretic terms, we examine the following hypothesis:

Hypothesis 2. *So-called (honest) H-behavior with RTS is a Subgame Perfect Nash Equilibrium (SPNE) in an infinitely repeated* DAG-PROTOCOL *game. This was presented in Inclusive [17] and we will contradict it.*

A. *Model of the* DAG-PROTOCOL

Players in DAG-PROTOCOL receive transaction fees after a delay. To simplify analysis, we can divide the flow of transactions into rounds of the game. This allows us to study player behavior within defined time. In each round, players make decisions and receive payoffs. Since no round is explicitly marked as the last one, this game is repeated infinitely.

We model DAG-PROTOCOL in the form of *an infinitely repeated two players game with a base game*

$$\Gamma = (\{P_{hon}, P_{grd}\}; \{H, G\}; U_{hon}, U_{mal}), \tag{2}$$

where P_{hon} is the player's determination to play H strategy and P_{grd} the player's determination to the G-behavior. Pure strategy H is interpreted as the RTS, while G strategy represents picking the transactions with the highest fees. Payoff functions are depicted in Tab. I, where the profits in the strategic profiles (H,H) and (G,G) are uniformly distributed between players. In the following, we analyze the model in five possible scenarios with generic levels a,b,c,d of the payoffs.

B. *Analysis of the Model*

For purposes of our analysis, lets start with the assumption that G-behavior is more attractive and profitable than H-behavior. Otherwise, there would be no reason to investigate Hypothesis 2. Thus, let us consider $c > a$ as the basic constraint. We also assume $c > b$, meaning that H-behavior loses against G-behavior in the cases of (H,G) and (G,H) profiles. These basic constraints yield the following scenarios:

- **Scenario 1**: $d > c > a > b$,
- **Scenario 2**: $c > d > a > b$,
- **Scenario 3**: $c > a > d > b$,
- **Scenario 4**: $c > a > b > d$,
- **Scenario 5**: where $a = d$ and $c > a$, $c > b$.

Note that we do not assume the case $a = b$ since the presence of P_{grd} will drain all high-fee transactions that P_{hon} would originally obtain.

The following provides a high-level summary of the scenarios. For a more comprehensive analysis, we refer the refer to the full version of our paper [28].

• Scenarios 1 and 2 are covered just for a sake of completeness. If the transaction fees were to cause such game outcomes, there would be no need to trust in H-behavior, and the system would settle in the unique (G,G) Pure Nash Equilibrium (*PNE*).

P_{hon}/P_{grd}	H	G
H	(a,a)	(b,c)
G	(c,b)	(d,d)

Tab. I: The utility functions U_{hon}, U_{mal} in the *base game.*

• Scenario 3 **(A) Purely Non-Cooperative Interpretation**. In Scenario 3, both players (P_{hon} and P_{grd}) are incentivized to choose the greedy G strategy, even though this leads to a worse overall outcome for both of them. This is because each player can do better by betraying the other player than by cooperating. This situation is known as a *Prisoner's dilemma* [24].

Proof: (Informal) Strategy G strictly dominates H and thus (G,G) is the unique PNE. ■

Corollary 1. *If P_{hon} is willing to follow the social norm of using the DAG protocol, then P_{grd}'s best response is also to use the G strategy. This is because P_{hon}'s cooperation is not credible, and P_{grd} can always benefit from betraying P_{hon}.*

• Scenario 3 **(B) When Some Coordination is Allowed**. It is possible for players to coordinate their behavior and achieve a better outcome for both of them, both playing H strategy. It must be common knowledge to the players that P_{hon} uses *grim trigger strategy* [24], [20]. This means that P_{hon} will cooperate as long as P_{grd} cooperates. However, if P_{grd} defects even once (playing G), then P_{hon} will switch to the G strategy forever. P_{grd} must also have a high discount factor. This means that she must value future payoffs more than immediate payoffs. If P_{grd}'s discount factor is too low, then she will be tempted to defect even if she knows it will lead to punishment in the long run.

• Scenario 4 **(A) Purely Non-Cooperative Interpretation**. We choose utility functions as follows: $a = 2$, $b = 1$, $c = 3$ and $d = 0$. This scenario is an anti-coordination game [24] instance, so the game has two PNEs (H,G) & (G,H), and one Mixed Nash Equilibrium (*MNE*) in mixed strategic profile $\left((\frac{1}{2},\frac{1}{2}),(\frac{1}{2},\frac{1}{2})\right)$.

Claim 1. *The most reasonable behavior in Scenario 4 is to play $(\frac{1}{2},\frac{1}{2})$ for both players.*

Proof: (Informal) Both players have two equally good choices: either be H or be G. From P_{hon}'s perspective, mixed behavior $(\frac{1}{2},\frac{1}{2})$ guarantees the best stable outcome. If P_{grd} expects $(\frac{1}{2},\frac{1}{2})$ behavior from P_{hon}, then P_{grd}'s best response is to play the same mixed behavior that establishes MNE. The players gain $(\frac{3}{2},\frac{3}{2})$ in that MNE, which is the highest expectation they can obtain. ■

Therefore, the most reasonable behavior for both players is to play a mixed strategy where they are half-H and half-G.

• Scenario 4 **(B) When Some Coordination is Allowed**. Similarly to Scenario 3, it is possible for players to coordinate their behavior and agree to always be H. This would be a good outcome for both players, as they would both get a payoff of 2. The same principle and consequences apply as in scenario 3(B) (Grimm trigger strategy). This will make the P_{grd} player regret defecting, and it will make her more likely to cooperate in the future. Therefore, the conclusion from Scenario 3 applies here as well.

• Scenario 5 **(A) Purely Non-Cooperative Interpretation**. In this scenario, the game is a zero-sum game, which means that no player can gain more than 100% profit, regardless of their chosen strategy. This is because the sum of all incoming transaction fees is fixed in any set of rounds. As a result, the total profit for all players is always constant if they all play the honest or greedy strategy. Therefore, the only rational outcome of this scenario is for both players to play the G strategy.

If we consider a social norm, it may be tempting to appeal to players' sense of responsibility and ask them to refrain from playing the G strategy. However, this is unlikely to be effective, as the H strategy does not benefit either player. Scenario 5 is highly similar to the classic game-theoretical model called *The Tragedy of Commons* [21]. In this model, individuals are incentivized to use a shared resource to the maximum extent possible, even if this depletes the resource and harms the group as a whole. In anonymous environments, where individuals cannot be held accountable for their actions, it is even more likely that they will prioritize their own interests over the interests of the group. This is because they know that they will not be punished for acting in their self-interest, meaning there is no harm to play G strategy.

C. Summary

We conclude that Hypothesis 2 is not valid. The (H,H) profile is not a PNE in any of our scenarios. Incentives enforcing H-behavior are hardly feasible in the anonymous (permissionless) environment of blockchains. A community of honest miners can follow the DAG-PROTOCOL until the attacker appears. The attacker playing the G strategy can parasite on the system and there is no defense against such a behavior (since greedy miners can leave the system anytime and mine elsewhere, which is not assumed in [17]). Therefore, H is not an *evolutionary stable strategy* [31], and thus H does not constitute a stable equilibrium.

VI. SIMULATION MODEL

We created a simulation model to conduct various experiments investigating the behavior of DAG-PROTOCOL under incentive attacks related to the problems identified in Sec. III and thus Hypothesis 1. Some experiments were designed to provide empirical evidence for the conclusions from Sec. V.

A. *Abstraction of* **DAG-PROTOCOL**

For evaluation purposes, we simulated the DAG-PROTOCOL (with RTS) by modeling the following aspects:

- All blocks in DAG are deterministically ordered.
- The mining rewards consist of transaction fees only.
- A fee of a particular transaction is awarded only to a miner of the block that includes the transaction as the first one in the sequence of totally ordered blocks.

Also, in terms of PHANTOM/GHOSTDAG terminology, we generalize and do not reduce transaction fees concerning the delay from "appearing" of the block until it is strongly connected to the DAG. Hence, we utilize $\gamma = 1$. In other words, for each block A, the discount function does not penalize a block according to its gap parameter $c(A)$, i.e. $\gamma(c(A)) = 1$. Such a setting is optimistic for honest miners and maximizes their profits from transaction fees when following the RTS strategy. This abstraction enables us to model the concerned problems of considered DAG-PROTOCOLS (see Sec. IV).

B. (Simple) Network Topology

We created a simple network topology that is convenient for proof-of-concept simulations and encompasses some important aspects of the real-world blockchain network. In particular, we were interested in emulating the network propagation delay τ to be similar to Bitcoin (i.e., $\sim 5s$ at most of the time in 2022), but using a small ring topology. To create such a topology, we assumed that the Bitcoin network contains 7592 nodes, according to the snapshot of reachable Bitcoin nodes found on May 24, 2022.[10] In Bitcoin core, the default value of the consensus node's peers is set to 8 (i.e., the node degree).[11] Therefore, the maximum number of hops that a gossiped message requires to reach all consensus nodes in the network is ~ 4.29 (i.e., $log_8(7592)$). Moreover, if we were to assume $2-3x$ more independent blockchain clients (that are not consensus nodes), then this number would be increased to 4.83–4.96. To model this environment, we used the ring network topology with 10 consensus nodes, which sets the maximum value of hops required to propagate a message to 5. Next, we set the inter-node propagation delay $\partial\tau$ to $1s$, which fits assumed τ (i.e., 5s / 5 hops = 1s).

C. Simulator

There are simulators [25] that model blockchain protocols, mainly focusing on network delays, different consensus protocols, and behaviors of specific attacks (e.g., SimBlock [4], Blocksim [1], Bitcoin-Simulator [11]). However, none of these simulators was sufficient for our purposes due to missing support for multiple chains and incentive schemes assumed in DAG-PROTOCOLS. To verify Hypothesis 1, we built a simulator that focuses on the mentioned problems of DAG-PROTOCOLS. In detail, we started with the Bitcoin mining simulator [9], which is a discrete event simulator for the PoW mining on a single chain, enabling a simulation of network propagation delay within a specified network topology. We extended this simulator to support DAG-PROTOCOLs, enabling us to monitor transaction duplicity, throughput, and relative profits of miners with regard to their mining power. The simulator is written in C++ (see details and its evaluation in [27]. In addition, we added more simulation complexity to simulate each block, including the particular transactions (as opposed to simulating only the number of transactions in a block [9]). Most importantly, we implemented two different transaction selection strategies – greedy and random. For demonstration purposes, we implemented the exponential distribution of transaction fees in mempool, based on several graph cuts of fee distributions in mempool of Bitcoin from [14].[12] Our simulator is available at https://github.com/Tem12/DAG-simulator.

VII. EVALUATION

We designed a few experiments with our simulator, which were aimed at investigating the relative profit of greedy miners and transaction collision rate (thus throughput) to investigate Hypothesis 1. In all experiments, honest miners followed the RTS, while greedy miners followed the greedy strategy. Unless

[10] https://bitnodes.io/nodes/

[11] Nevertheless, the node degree is often higher than 8 in reality [22].

[12] Distribution of transaction fees in mempool might change over time; however, it mostly preserves the low number of high-fee transactions.

stated otherwise, the block creation time was set to $\lambda = 20s$. However, we abstracted from τ of transactions and ensured that the mempools of nodes were regularly filled (i.e., every 60s) by the same set of new transactions, while the number of transactions in the mempool was always sufficient to fully satisfy the block capacity that was set to 100 transactions. We set the size of mempool equal to 10000 transactions, and thus the ratio between these two values is similar to Bitcoin [14] in common situations. In all experiments, we executed multiple runs and consolidated their results; however, in all experiments with the simple topology, the spread was negligible, and therefore we do not depict it in graphs.

A. Experiment I

Goal. The goal of this experiment was to compare the relative profits earned by two miners/phenotypes in a network, corresponding to our game theoretical settings (see Sec. V). Thus, one miner was greedy and followed the greedy strategy, while the other one was honest and followed the RTS.

Methodology and Results. The ratio of total mining power between the two miners was varied with a granularity of 10%, and the network consisted of 10 miners, where only the two miners had assigned the mining power. Other miners acted as relays, emulating the maximal network delay of 5 hops between the two miners in a duel. The relative profits of the miners were monitored as their profit factor $\mathbb{P}$ w.r.t. their mining power. We conducted 10 simulation runs and averaged their results (see Fig. 2). Results show that the greedy miner earned a profit disproportionately higher than her mining power, while the honest miner's relative profit was negatively affected by the presence of the greedy miner. We can observe that $\mathbb{P}$ of greedy miner was indirectly proportional to her κ, which was caused by the exponential distribution of transaction fees that contributed more significantly to the higher $\mathbb{P}$ of a smaller miner. In sum, the profit advantage of the greedy miner aligns with the conclusions from the game theoretical model (Scenario 5, see Sec. V) in particular, which represents the case of κ=50%. Nevertheless, our results indicate that the greedy strategy is more profitable than the RTS for any non-zero κ.

B. Experiment II

Goal. The goal of this experiment was investigation of the relative profits of a few greedy miners following the greedy strategy in contrast to honest miners following the RTS.

Methodology and Results. We experimented with 10 miners, where the number of greedy miners $\mathbb{G}$ vs. the number of honest miners (i.e., 10 - $\mathbb{G}$) was varied, and each held 10% of the total mining power. We monitored their profit factor $\overline{\mathbb{P}}$ averaged per miner. We conducted 10 simulation runs and averaged their results (see Fig. 3a). Alike in Sec. VII-A, we can see that greedy miners earned profit disproportionately higher than their mining power. Similarly, this experiment showed that the profit advantage of greedy miners decreases as their number increases. This is similar to increasing κ in a duel of two miners from Sec. VII-A; however, in contrast to it, $\overline{\mathbb{P}}$ of greedy miners is slightly lower with the same total κ of all greedy miners, while $\overline{\mathbb{P}}$ of honest miners had not suffered with such a decrease. Intuitively, this happened because multiple greedy miners increase transaction collision. In detail, since miners are only rewarded for transactions that were first to be included in a new block, the profit for the second and later miners is lost if a duplicate transaction is included. This observation might be seen as beneficial for the protocol as it disincentivizes multiple miners to use the greedy transaction selection strategy, which would support the sequential equilibrium from [17]. However, the authors of [17] do not assume cooperating players, which is unrealistic since miners can cooperate and create the pool to avoid collisions and thus maximize their profits (resulting in a similar outcome, as in Sec. VII-A).

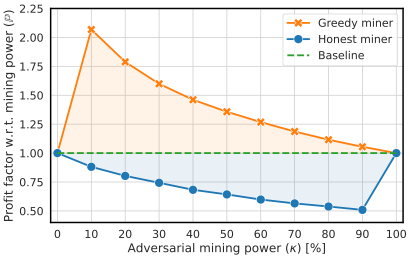


Fig. 2: The profit factor $\mathbb{P}$ of an honest vs. a greedy miner with their mining powers of 100% - κ and κ, respectively. The baseline shows the expected $\mathbb{P}$ of the honest miner; $\lambda = 20s$.

C. Experiment III

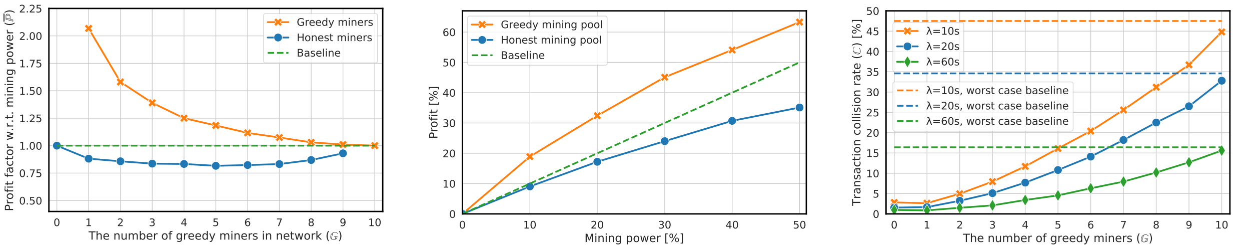
Goal. The goal of this experiment was to investigate the relative profit of the greedy mining pool depending on its κ versus the honest mining pool with the same mining power. It is equivalent to Scenario 5 of game theoretical analysis (see Sec. V) although there is the honest rest of the network.

Methodology and Results. We experimented with 10 miners, and out of them, we choose one greedy miner and one honest miner, both having equal mining power, while the remaining miners in the network were honest and possessed the rest of the network's mining power. Thus, we emulated a duel of the greedy pool versus the honest pool. We conducted 10 simulation runs and averaged their results (see Fig. 3b). The results demonstrate that the greedy pool's relative earned profit grows proportionally to κ as compared to the honest pool with equal mining power, supporting our conclusions from Sec. V.

D. Experiment IV

Goal. The goal of this experiment was to investigate the transaction collision rate under the occurrence of greedy miners who selected transactions using the greedy strategy.

Methodology and Results. In contrast to the previous experiments, we considered three different values of block creation time ($\lambda \in \{10s, 20s, 60s\}$). We experimented with 10 miners, where the number of greedy miners $\mathbb{G}$ vs. the number of honest miners (i.e., 10 - $\mathbb{G}$) was varied, and each held 10% of the total mining power. For all configurations, we computed the transaction collision rate (see Fig. 3c). We can see that the increase of $\mathbb{G}$ causes the increase in the transaction collision rate. Note that lower λ has a higher impact on the collision rate, and DAG protocols are designed with the intention to have small λ (i.e., even smaller than τ). Consequently, the



(a) The averaged profit factor $\overline{\mathbb{P}}$ per honest miner and greedy miner, each with 10% of mining power. The number of honest miners is 10 - $\mathbb{G}$. The baseline shows the expected $\overline{\mathbb{P}}$ of an honest miner with 10% of mining power.

(b) The relative profit of the honest pool and the greedy pool, both with equal mining power (i.e., κ), w.r.t. the total mining power of the network. The baseline shows the expected profit of the honest mining pool, and $\lambda = 20s$.

(c) The transaction collision rate $\mathbb{C}$ w.r.t. # of greedy miners $\mathbb{G}$ (each with $\kappa = 10\%$), where # of honest miners was $10 - \mathbb{G}$ and $\lambda \in \{10s, 20s, 60s\}$. The worst case baseline shows $\mathbb{C}$ when all transactions are duplicates.

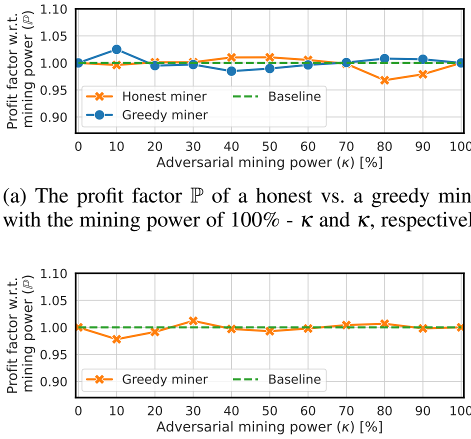
Fig. 3: Experiment II, Experiment III (i.e., duel of mining pools) and Experiment IV (i.e., transaction collision rate & throughput).

increased collision rate affected the overall throughput of the network (which is complementary to Fig. 3c).

VIII. Countermeasures

Experiments supported Hypothesis 1. The main problem is **not sufficiently enforcing the RTS**, i.e., verifying that transaction selection was indeed random at the protocol level. Therefore, using the RTS in the DAG-Protocol that does not enforce the interpretation of randomness will never avoid the occurrence of attackers from greedy transaction selection that increases their individual (or pooled) profits.

Enforcing Interpretation of the Randomness. One countermeasure how to avoid arbitrary interpretation of the randomness in the RTS is to enforce it by the consensus protocol. An example of a DAG-based design using this approach is Sycomore [2], which utilizes the prefix of cryptographically-secure hashes of transactions as the criteria for extending a particular chain in DAG. The PoW mining in Sycomore is further equipped with the unpredictability of a chain that the miner of a new block extends, avoiding the concentration of the mining power on "rich" chains. Note that transactions are evenly spread across all chains of the DAG, which happens because prefixes of transaction hashes respect the uniform distribution – transactions are created by clients (different from miners) who have no incentives for biasing their transactions.



(a) The profit factor $\mathbb{P}$ of a honest vs. a greedy miner with the mining power of 100% - κ and κ, respectively.

(b) The averaged profit factor $\overline{\mathbb{P}}$ of a greedy miner with κ. The rest of the network consisted of 9 honest miners, each equipped with $\frac{100\%-\kappa}{9}\%$ of mining power.

Fig. 4: Profit factors of honest and greedy miners. The baseline shows the expected $\mathbb{P}$ of the honest miner; $\lambda = 20s$.

Fixed Transaction Fees. Another option how to make the RTS viable is to employ fixed fees for all transactions as a blockchain network-adjusted parameter. In the case of the full block capacity utilization within some period, the fixed fee parameter would be increased and vice versa in the case of not sufficiently utilized block capacity. In contrast to the previous countermeasure, this mechanism does not enforce the interpretation of randomness while at the same time does not make incentives for greedy miners to follow other than the RTS strategy. Therefore, miners using other than the RTS would not earn extra profits – we demonstrate it in Fig. 4a and Fig. 4b, considering one honest vs. one greedy miner and one greedy vs. 9 honest miners, respectively. Note that small deviations from the baseline are caused by the inherent simulation error that is present in the original simulator that we extended. On the other hand, greedy miners may still cause increased transaction collision rate, and thus decreased throughput. Therefore, we consider the fixed transaction fee option weaker than the previous one.

IX. Related Work

Wang et al. [37] performed a detailed systematic overview of DAG-designs. They described six categories containing more than thirty DAG-based blockchain systems classified based on their characteristics and principles. They extend the commonly used classification based on the type of ledgers [36]. GHOST [35], Inclusive Blockchain [17], Conflux [18], Haootia [36], and Byteball [3] represent DAG with the main chain. Hashgraph [15] and Nano [16] represent ledgers with parallel chains. Nevertheless, out of these categories, only DAGs with the main chain are related to our research, such as Inclusive [17], SPECTRE [33], PHANTOM and GHOSTDAG [34]. We refer the reader to Sec. IV for details about these protocols.

Sycomore [2] and its extension Sycomore++ [8] is another DAG-oriented consensus protocol that utilizes DAGs to increase Nakamoto consensus throughput. The protocol proposes that the chain responds to the dynamically increased number of transactions and splits them into multiple chains, thus creating DAG structure. Transactions are evenly partitioned based on the prefix of their hash, and they are randomly inserted into their corresponding chain (branch). The protocol does not directly suffer from our proposed attacks, although it might suffer from different problems related to double spending of transactions mined in parallel, which is, however, common for all DAG-oriented protocols.

X. Conclusion

We started with an overview of DAG-oriented consensus protocols for PoW blockchains, which promise to increase the transaction throughput by using the RTS strategy. We formulated a hypothesis that DAG protocols using the RTS can be exploited by attackers not respecting such a strategy and instead selecting transaction based on the fees (i.e., greedy strategy), which can lead to deterioration of the overall transaction throughput. We made a game theoretical analysis of concerned DAG-oriented protocols and concluded that the RTS strategy, as proposed in these protocols, does not constitute a Nash equilibrium since honest players enable the greedy player to "parasite" on the system. This is contradictory result to Inclusive paper [17], which does not assume that multiple greedy miners can form a mining pool. We conducted several experiments on a network topology consisting of 10 nodes, where we analyzed the impact of greedy miners who deviated from the modeled DAG protocol by selecting transactions based on the highest fee. We demonstrated that greedy miners have a significant advantage over honest miners in terms of profit maximization. Moreover, we showed that greedy miners have a detrimental impact on transaction throughput and have the incentive to form a mining pool, worsening the decentralization of assumed DAG-protocols.

References

[1] Maher Alharby and Aad van Moorsel. Blocksim: A simulation framework for blockchain systems. *SIGMETRICS Perform. Eval. Rev.*, 46(3):135–138, January 2019.

[2] Emmanuelle Anceaume et al. Sycomore: A Permissionless Distributed Ledger that Self-Adapts to Transactions Demand. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–8. IEEE, 2018.

[3] Anton Churyumov. Byteball: A Decentralized System for Storage and Transfer of Value. https://byteball.org/Byteball.pdf, 2016.

[4] Yusuke Aoki et al. SimBlock: A Blockchain Network Simulator. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 325–329. IEEE, 2019.

[5] Vivek Bagaria et al. Prism: Deconstructing the blockchain to approach physical limits. In *ACM SIGSAC CCS '19*, pages 585–602. ACM New York, NY, USA, 2019.

[6] R. Bowden et al. Block arrivals in the Bitcoin blockchain. arXiv:1801.07447, 2018.

[7] Christian Decker and Roger Wattenhofer. Information propagation in the Bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10. IEEE, 2013.

[8] Aimen Djari et al. Simulation study of Sycomore++, a self-adapting graph-based permissionless distributed ledger. In *BRAINS '22*, pages 103–110. IEEE, 2022.

[9] Gavin Andresen. Bitcoin Miningsim. https://github.com/gavinandresen/bitcoin_miningsim, 2015.

[10] Arthur Gervais et al. On the Security and Performance of Proof of Work Blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 3–16, New York, NY, USA, 2016. Association for Computing Machinery.

[11] Vasilis Glykantzis and Arthur Gervais. Bitcoin-Simulator, capable of simulating any re-parametrization of Bitcoin. online, april 2016.

[12] J. Göbel et al. Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay. *Performance Evaluation*, 104:23–41, 2016.

[13] Cyril Grunspan and Ricardo Pérez-Marco. The mathematics of Bitcoin. arXiv:2003.00001, 2020.

[14] Jochen Hoenicke. Johoe's Bitcoin Mempool Statistic. https://jochen-hoenicke.de/queue/, 2022.

[15] Leemon Baird. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. https://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf, 2016.

[16] Colin LeMahieu. Nano : A feeless distributed cryptocurrency network. https://www.exodus.com/assets/docs/nano-whitepaper.pdf, 2018.

[17] Yoad Lewenberg et al. Inclusive Block Chain Protocols. In *Financial Cryptography and Data Security*, pages 528–547. Springer, Berlin Heidelberg, 2015.

[18] Chenxing Li et al. Scaling Nakamoto Consensus to Thousands of Transactions per Second. arXiv:1805.03870, 2018.

[19] Ziyao Liu et al. A Survey on Blockchain: A Game Theoretical Perspective. doi.org/10.1109/ACCESS.2019.2909924, 2019.

[20] George J. Mailath and Larry Samuelson. *Repeated Games and Reputations: Long-Run Relationships*. Oxford University Press, November 2006.

[21] J.D. Miller. *Game Theory at Work: How to Use Game Theory to Outthink and Outmaneuvar Your Competition*. McGraw Hill LLC, 2003.

[22] Jelena Mišić et al. Modeling of Bitcoin's blockchain delivery network. *IEEE TNSE*, 7(3):1368–1381, 2019.

[23] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[24] Martin J. Osborne and Ariel Rubinstein. *A course in game theory*. The MIT Press, Cambridge, USA, 1994. electronic edition.

[25] Remigijus Paulavičius et al. A Systematic Review and Empirical Analysis of Blockchain Simulators. doi.org/10.1109/ACCESS.2021.3063324, 2021.

[26] Martin Perešíni et al. DAG-Oriented Protocols PHANTOM and GHOSTDAG under Incentive Attack via Transaction Selection Strategy. *arXiv:2109.01102*, 2021.

[27] Martin Perešíni et al. DAG-Sword: A Simulator of Large-Scale Network Topologies for DAG-Oriented Proof-of-Work Blockchains. *arXiv:2311.04638*, 2023.

[28] Martin Perešíni et al. Incentive Attacks on DAG-Based Blockchains with Random Transaction Selection. arXiv:2305.16757, 2023.

[29] Wellington Fernandes Silvano and Roderval Marcelino. IOTA Tangle: A cryptocurrency to communicate Internet-of-Things data. *Future Generation Computer Systems*, 112:307–319, 2020.

[30] Rajani Singh et al. A game theoretic analysis of resource mining in blockchain. *Cluster Computing*, 23(3):2035–2046, 2020.

[31] John Maynard Smith. *Evolution and the Theory of Games*. Cambridge University Press, 1982.

[32] Yonatan Sompolinsky. Kaspa. online, 2022. https://kaspa.org/.

[33] Yonatan Sompolinsky et al. SPECTRE: A Fast and Scalable Cryptocurrency Protocol. Cryptology ePrint Archive, Paper 2016/1159, 2016.

[34] Yonatan Sompolinsky et al. PHANTOM GHOSTDAG: A Scalable Generalization of Nakamoto Consensus. In *ACM AFT '21*, pages 57–70, New York, NY, USA, 2021. ACM.

[35] Yonatan Sompolinsky and Aviv Zohar. Accelerating Bitcoin's Transaction Processing. Fast Money Grows on Trees, Not Chains. Cryptology ePrint Archive, Paper 2013/881, 2013.

[36] Shuyang Tang. Bracing A Transaction DAG with A Backbone Chain. Cryptology ePrint Archive, Report 2020/472, 2020. https://ia.cr/2020/472.

[37] Qin Wang et al. SoK: Diving into DAG-based Blockchain Systems. arXiv:2012.06128, 2020.

[38] Taotao Wang et al. Game-Theoretical Analysis of Mining Strategy for Bitcoin-NG Blockchain Protocol. *IEEE Systems Journal*, 15(2):2708–2719, 2021.

Mitigating Undercutting Attacks: Fee-Redistribution Smart Contracts for Transaction-Fee-Based Regime of Blockchains with the Longest Chain Rule

Rastislav Budinsky
Brno University of Technology
Faculty of Information Technology
xbudin05@stud.fit.vutbr.cz

Ivana Stančíková
Brno University of Technology
Faculty of Information Technology
istancikova@fit.vutbr.cz

Ivan Homoliak
Brno University of Technology
Faculty of Information Technology
ihomoliak@fit.vutbr.cz

***Abstract*—In this paper, we review the undercutting attacks in the transaction-fee-based regime (i.e., without block rewards) of proof-of-work (PoW) blockchains with the longest chain fork-choice rule. Next, we focus on the two related problems: (1) fluctuations in mining revenue and (2) the mining gap – i.e., a situation, in which the immediate reward from transaction fees does not cover miners' expenditures.**

To mitigate these issues, we propose a solution that splits transaction fees from a mined block into two parts – (1) an instant reward for the miner of a block and (2) a deposit sent to one or more fee-redistribution smart contracts ($\mathcal{FRSC}$s) that are part of the consensus protocol. At the same time, these $\mathcal{FRSC}$s reward the miner of a block with a certain fraction of the accumulated funds over a predefined time. This setting enables us to achieve several interesting properties that improve the incentive stability and security of the protocol, which is beneficial for the honest miners. With our solution, the fraction of DEFAULT-COMPLIANT miners who strictly do not execute undercutting attacks is lowered from the state-of-the-art result of 66% to 30%.

I. INTRODUCTION

There are two sources of miners' income in cryptocurrencies – block rewards and transaction fees. Block rewards incentivize miners to produce new blocks. Transaction fees not only contribute to the revenue of miners, but motivate them to include transactions in blocks. Miners maximize their profits by prioritizing the transactions with the highest fees. In Bitcoin and its numerous clones [1], the block reward is divided by two approx. every four years (i.e., after every 210k blocks), which will eventually result in a zero block reward and thus **a pure transaction-fee-based regime**. There was only very little research made (see § VII) to investigate properties of transaction-fee-based regimes, which motivated our work.

Before 2016, there was a belief that the dominant source of the miners' income does not impact the security of the blockchain. However, Carlsten et al. [2] pointed out the effects of the high variance of the miners' revenue per block caused by exponentially distributed block arrival time in the transaction-fee-based protocols. The authors showed that *undercutting* (i.e., forking) a wealthy block is a profitable strategy for a malicious miner. Nevertheless, literature [3], [4] showed that this attack is viable even in blockchains containing traditional block rewards due to front-running competition of arbitrage bots who are willing to extremely increase transaction fees to earn Maximum Extractable Value profits.

In this paper, we focus on mitigation of the undercutting attack the in transaction-fee-based regime of PoW blockchains – i.e., blockchains that prefer availability over consistency within the CAP theorem and thus are designed to resolve forks often. We also discuss related problems present (not only) in transaction-fee-based regime. In particular, we focus on minimizing the mining gap [2], [5], (i.e., the situation, where the immediate reward from transaction fees does not cover miners' expenditures) as well as balancing significant fluctuations in miners' revenue. To mitigate these issues, we propose a solution that splits transaction fees from a mined block into two parts – (1) an instant reward for the miner and (2) a deposit sent into one or more fee-redistribution smart contracts ($\mathcal{FRSC}$s). At the same time, these $\mathcal{FRSC}$s reward the miner of a block with a certain fraction of the accumulated funds over a fixed number of blocks. Our solution can be deployed with hard-fork and imposes only negligible performance overhead.

Contributions In detail, our contributions are as follows:

1) We propose an approach that normalizes the mining rewards coming from transaction fees by one or more $\mathcal{FRSC}$s that emulate moving average on a certain portion of the transaction fees.
2) We evaluate our approach using various fractions of the transaction fees from a block distributed between a miner and $\mathcal{FRSC}$s. We experiment with the various numbers and lengths of $\mathcal{FRSC}$s, and we demonstrate that usage of multiple $\mathcal{FRSC}$s of various lengths has the best advantages mitigating the problems we are addressing.
3) We demonstrate that with our approach, the mining gap can be minimized since the miners at the beginning of the mining round can get the reward from $\mathcal{FRSC}$s, which stabilizes their income.
4) We demonstrate that the threshold of DEFAULT-COMPLIANT miners who strictly do not execute undercutting attack is lowered from 66% (as reported in the original work [2]) to 30% with our approach.

II. PROBLEM DEFINITION

In transaction fee-based regime schemes, a few problems have emerged, which we can observe even nowadays in Bitcoin protocol [2]. We have selected three main problems and aim to lower their impact for protocols relying on transaction fees only. In detail, we focus on the following problems:

1) **Undercutting attack.** In this attack (see Fig. 1), a malicious miner attempts to obtain transaction fees by re-mining a top block of the longest chain, and thus

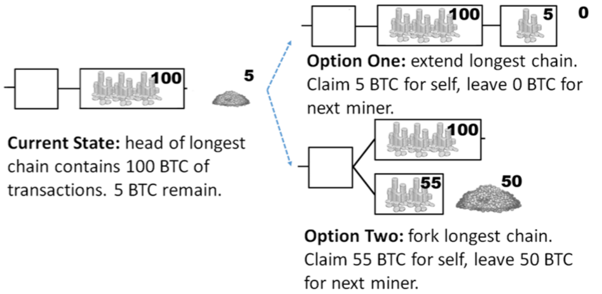

100
5
0
5
100
55
50
Option One: extend longest chain. Claim 5 BTC for self, leave 0 BTC for next miner.
Current State: head of longest chain contains 100 BTC of transactions. 5 BTC remain.
Option Two: fork longest chain. Claim 55 BTC for self, leave 50 BTC for next miner.

Fig. 1: The undercutting attack, according to Carlsten et al. [2].

motivates other miners to mine on top of her block [2]. In detail, consider a situation, where an honest miner mines a block containing transactions with substantially higher transaction fees than is usual. The malicious miner can fork this block while he leaves some portion of the "generous" transactions un-mined. These transactions motivate other miners to mine on top of the attacker's chain, and thus undercut the original block. Such a malicious behavior might result in higher orphan rate, unreliability of the system, and even double spending.

2) **The mining gap.** As discussed in [2], the problem of mining gap arises once the mempool does not contain enough transaction fees to motivate miners in mining. Suppose a miner succeeds at mining a new block shortly after the previous block was created, which can happen due to well known exponential distribution of block creation time in PoW blockchains. Therefore, the miner might not receive enough rewards to cover his expenses because most of the transactions from the mempool were included in the previous block, while new transactions might not have yet arrived or have small fees. Consequently, the miners are motivated to postpone mining until the mempool is reasonably filled with enough transactions (and their fees). The mining gap was also analyzed by the simulation in the work of Tsabary and Eyal [5], who further demonstrated that mining gap incentivizes larger mining coalitions (i.e., mining pools), negatively impacting decentralization.
3) **Varying transaction fees over time.** In the transaction-fee-based regime, any fluctuation in transaction fees directly affects the miners' revenue. High fluctuation of transaction fees during certain time frames, e.g., in a span of a day or a week [6], can lead to an undesirable lack of predictability in miners' rewards and indirectly affect the security of the underlying protocol.

III. Proposed Approach

We propose a solution that collects a percentage of transaction fees in a native cryptocurrency from the mined blocks into one or multiple fee-redistribution smart contracts (i.e., $\mathcal{FRSC}$s). Miners of the blocks who must contribute to these contracts are at the same time rewarded from them, while the received reward approximates a moving average of the incoming transaction fees across the fixed sliding window of the blocks. The fraction of transaction fees (i.e., $\mathbb{C}$) from the mined block is sent to the $\mathcal{FRSC}$ and the remaining fraction of transaction fees (i.e., $\mathbb{M}$) is directly assigned to the miner, such that $\mathbb{C}+\mathbb{M}=1$. The role of $\mathbb{M}$ is to incentivize the miners in prioritization of the transactions with the higher fees while the role of $\mathbb{C}$ is to mitigate the problems of undercutting attacks and the mining gap.

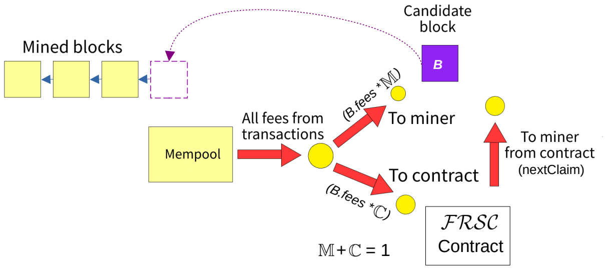


Fig. 2: Overview of our solution.

Overview. We depict the overview of our approach in Fig. 2, and it consists of the following steps:

1) Using $\mathcal{FRSC}$, the miner calculates the reward for the next block B (i.e., $nextClaim(\mathcal{FRSC})$ – see Eq. 4) that will be payed by $\mathcal{FRSC}$ to the miner of that block.
2) The miner mines the block B using the selected set of the highest fee transactions from her mempool.
3) The mined block B directly awards a certain fraction of the transaction fees (i.e., $B.fees\ *\ \mathbb{M}$) to the miner and the remaining part (i.e., $B.fees\ *\ \mathbb{C}$) to $\mathcal{FRSC}$.
4) The miner obtains $nextClaim$ from $\mathcal{FRSC}$.

Our approach is embedded into the consensus protocol, and therefore consensus nodes are obliged to respect it in order to ensure that their blocks are valid. It can be implemented with standard smart contracts of the blockchain platform or within the native code of the consensus protocol.

A. Prioritization of High-Fee Transactions

In the environment with constant transaction fees, a miner would receive the same amount with or without our solution. However, in public blockchains (especially with transaction-fee-based regime) there exists a prioritization in processing of transactions with higher fees, which might result into fluctuations in rewards of the miners. In our approach, we preserve the transaction prioritization since we directly attribute a fixed fraction of the transaction fees to the miner (i.e., $\mathbb{M}$).

B. Fee-Redistribution Smart Contracts

We define the fee-redistribution smart contract as a tuple

$$\mathcal{FRSC} = (\nu, \lambda, \rho), \tag{1}$$

where ν is the accumulated amount of tokens in the contract, λ denotes the size of $\mathcal{FRSC}$'s sliding window in terms of the number of preceding blocks that contributed to ν, and ρ is the parameter defining the ratio for redistribution of incoming transaction fees among multiple contracts (if there are multiple $\mathcal{FRSC}$s), while the sum of ρ across all $\mathcal{FRSC}$s must be equal to 1:

$$\sum_{x\ \in\ \mathcal{FRSC}s} x.\rho \quad = \quad 1. \tag{2}$$

In contrast to a single $\mathcal{FRSC}$, we envision multiple $\mathcal{FRSC}$s to enable better adjustment of compensation to miners during periods of higher transaction fee fluctuations or in an unpredictable environment (we show this in § V-C).

We denote the state of $\mathcal{FRSC}$s at the blockchain height H as $\mathcal{FRSC}_{[H]}$. Then, we determine the reward from $\mathcal{FRSC}_{[H]} \in \mathcal{FRSC}s_{[H]}$ for the miner of the next block with height $H+1$ as follows:

$$\partial Claim^{\mathcal{FRSC}_{[H]}}_{[H+1]} = \frac{\mathcal{FRSC}_{[H]}.\nu}{\mathcal{FRSC}_{[H]}.\lambda}, \tag{3}$$

while the reward obtained from all $\mathcal{FRSC}$s is

$$nextClaim_{[H+1]} = \sum_{\mathcal{X}_{[H]} \ \in \ \mathcal{FRSC}s_{[H]}} \partial Claim^{\mathcal{X}_{[H]}}_{[H+1]}. \tag{4}$$

Then, the total reward of the miner who mined the block $B_{[H+1]}$ with all transaction fees $B_{[H+1]}.fees$ is

$$rewardT_{[H+1]} = nextClaim_{[H+1]} + \mathbb{M} * B_{[H+1]}.fees. \tag{5}$$

The new state of contracts at the height $H+1$ is

$$\mathcal{FRSC}s_{[H+1]} = \{\mathcal{X}_{[H+1]}(\nu, \lambda, \rho) \ \mid \tag{6}$$

$$\begin{aligned}
\lambda &= \mathcal{X}_{[H]}.\lambda, & (7)\\
\rho &= \mathcal{X}_{[H]}.\rho, & (8)\\
\nu &= \mathcal{X}_{[H]}.\nu - \partial Claim_{[H+1]} + deposit * \rho, & (9)\\
deposit &= B_{[H+1]}.fees * \mathbb{C}\}, & (10)
\end{aligned}$$

where $deposit$ represents the fraction $\mathbb{C}$ of all transaction fees from the block $B_{[H+1]}$ that are deposited across all $\mathcal{FRSC}$s in ratios respecting Eq. 2.

C. Example

We consider Bitcoin [7] with the current height of the blockchain H. We utilize only a single $\mathcal{FRSC}$:

$$\mathcal{FRSC}_{[H]} = (2016, 2016, 1).$$

We set $\mathbb{M} = 0.4$ and $\mathbb{C} = 0.6$, which means a miner directly obtains 40% of the $B_{[H+1]}.fees$ and $\mathcal{FRSC}$ obtains 60%. Next, we compute the reward from $\mathcal{FRSC}$ obtained by the miner of the block with height $H+1$ as

$$\partial Claim_{[H+1]} = \frac{\mathcal{FRSC}_{[H]}.\nu}{\mathcal{FRSC}_{[H]}.\lambda} = \frac{2016}{2016} = 1 \text{ BTC},$$

resulting into

$$nextClaim_{[H+1]} = \partial Claim_{[H+1]} \ = \ 1 \text{ BTC}.$$

Further, we assume that the total reward collected from transactions in the block with height $H+1$ is $B_{[H+1]}.fees = 2$ BTC. Hence, the total reward obtained by the miner of the block $B_{[H+1]}$ is

$$\begin{aligned}
rewardT_{[H+1]} &= nextClaim_{[H+1]} + \mathbb{M} * B_{[H+1]}.fees\\
&= 1 + 0.4 * 2 \ = \ 1.8 \text{ BTC},
\end{aligned}$$

and the contribution of transaction fees from $B_{[H+1]}$ to the $\mathcal{FRSC}$ is

$$deposit = B_{[H+1]}.fees * \mathbb{C} = 1.2 \text{ BTC}.$$

Therefore, the value of ν in $\mathcal{FRSC}$ is updated at height H + 1 as follows:

$$\begin{aligned}
v_{[H+1]} &= \mathcal{FRSC}_{[H]}.\nu - nextClaim_{[H+1]} + deposit\\
&= 2016 - 1 + 1.2 \text{ BTC} \ = \ 2016.2 \text{ BTC}.
\end{aligned}$$

Traditional Way in Tx-Fee Regime In traditional blockchains, $rewardT_{[H+1]}$ would be equal to the sum of all transaction fees $B_{[H+1]}.fees$ (i.e., 2 BTC); hence, using $\mathbb{M} = 1$. In our approach, $rewardT_{[H+1]}$ is equal to the sum of all transaction fees in the block $B_{[H+1]}$, if:

$$B_{[H+1]}.fees = \frac{nextClaim_{[H+1]}}{\mathbb{C}}. \tag{11}$$

In our example, a miner can mine the block $B_{[H+1]}$ while obtaining the same total reward as the sum of all transaction fees in the block if the transactions carry 1.66 BTC in fees:

$$B_{[H+1]}.fees = \frac{1}{0.6} = 1.66 \text{ BTC}.$$

D. Initial Setup of $\mathcal{FRSC}$s Contracts

To enable an even start, we propose to initiate $\mathcal{FRSC}$s of our approach by a genesis value. The following formula calculates the genesis values per $\mathcal{FRSC}$ and initializes starting state of $\mathcal{FRSC}s_{[0]}$:

$$\{\mathcal{FRSC}^{x}_{[0]}(\nu, \lambda, \rho) \mid \nu = \overline{fees} * \mathbb{C} * \rho * \lambda\}, \tag{12}$$

where $\overline{fees}$ is the expected average of incoming fees.

IV. Implementation

Our implementation is based on Bitcoin Mining Simulator [8], introduced in [2], which we modified for our purposes.

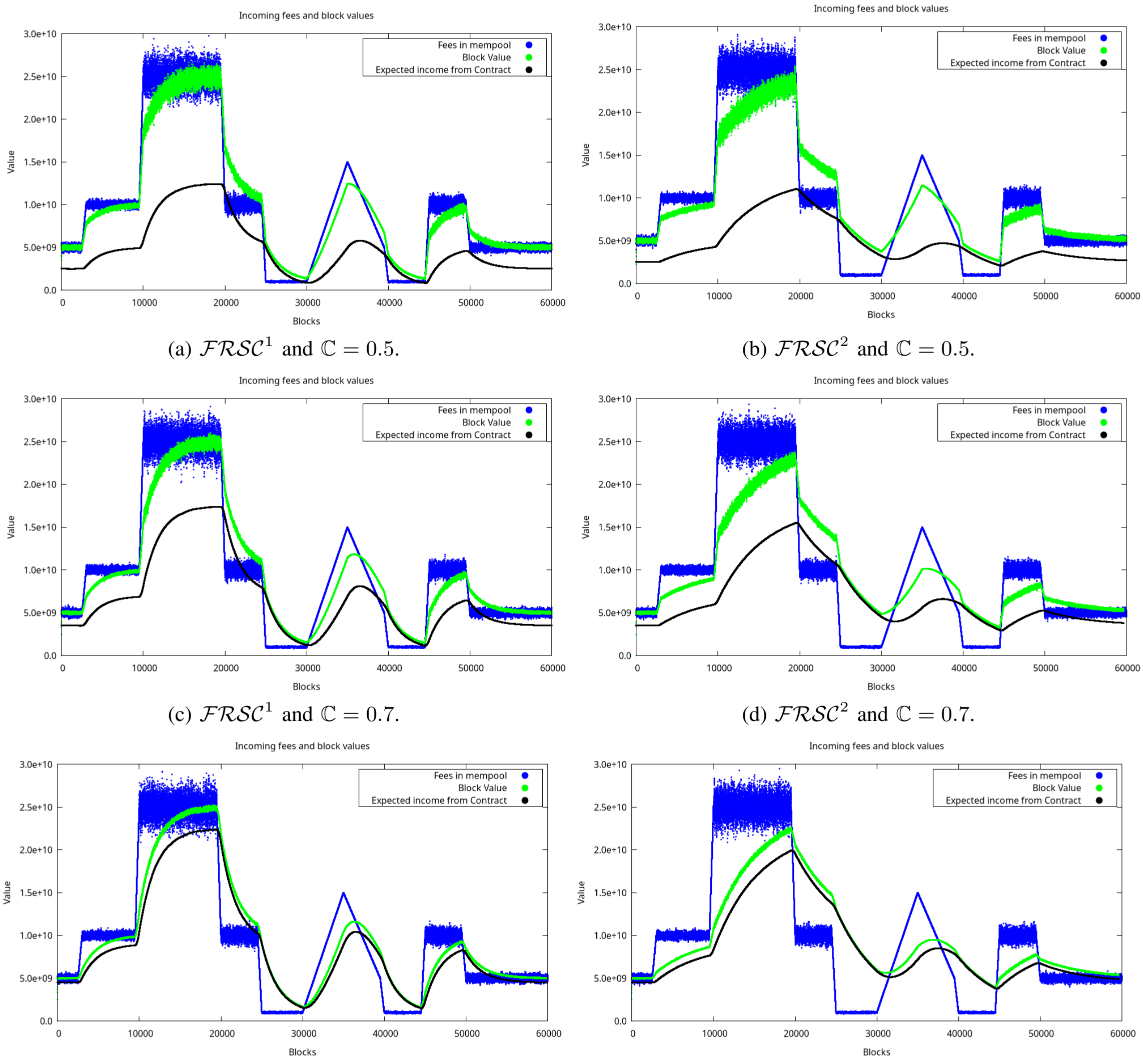
1) ***Our Changes in Simulator.*** We have created a configuration file to simulate custom scenarios of incoming transactions instead of the accumulated fees in the original design [2]. We added an option to switch simulation into a mode with a full mempool, and thus bound the total fees (and consequently the total number of transactions) that can be earned within a block – this mostly relates to blocks whose mining takes longer time than the average time to mine a block.[1] Next, we moved several parameters to arguments of the simulator to eliminate the need for frequent recompilation of the program, and therefore simplified the process of running various experiments with the simulator. Finally, we integrated our $\mathcal{FRSC}$-based solution into the simulator. $\mathcal{FRSC}$s are initiated from a corresponding configuration file. The source code of our modified simulator is available at https://github.com/The-Huginn/mining_simulator.

V. Evaluation

We evaluated our proof-of-concept implementation of $\mathcal{FRSC}$s on a custom long-term scenario designed to demonstrate significant changes in the total transaction fees in the mempool evolving across the time. This scenario is depicted in the resulting graphs of most of our experiments, represented by the "*Fees in mempool*" series – see § V-A and § V-B.

We experimented with different parameters and investigated how they influenced the total rewards of miners coming from $\mathcal{FRSC}$s versus the baseline without our solution. Mainly, these included a setting of $\mathbb{C}$ as well as different lengths λ of $\mathcal{FRSC}$s. For demonstration purposes, we used the value of transaction fees per block equal to 50 BTC, the same as Carlsten et al. [2] used. Across all our experiments but the last one (i.e., § V-D), we enabled the full mempool option to ensure more realistic conditions.

[1] Note that the original simulator [2] assumes that the number of transactions (and thus the total fees) in the block is constrained only by the duration of a time required to mine the block, which was also criticized in [9].



(a) $\mathcal{FRSC}^1$ and $\mathbb{C} = 0.5$.

(b) $\mathcal{FRSC}^2$ and $\mathbb{C} = 0.5$.

(c) $\mathcal{FRSC}^1$ and $\mathbb{C} = 0.7$.

(d) $\mathcal{FRSC}^2$ and $\mathbb{C} = 0.7$.

(e) $\mathcal{FRSC}^1$ and $\mathbb{C} = 0.9$.

(f) $\mathcal{FRSC}^2$ and $\mathbb{C} = 0.9$.

Fig. 3: Experiment I investigating various $\mathbb{C}s$ and λs of a single $\mathcal{FRSC}$, where $\mathcal{FRSC}^1.\lambda = 2016$ and $\mathcal{FRSC}^2.\lambda = 5600$. *Fees in mempool* show the total value of fees in the mined block (i.e., representing the baseline). *Block Value* is the reward a miner received in block B as a sum of the fees he obtained directly (i.e. $\mathbb{M} * B.fees$) and the reward he got from $\mathcal{FRSC}$ (i.e., $nextClaim_{[H]}$). *Expected income from Contract* represents the reward of a miner obtained from $\mathcal{FRSC}$ (i.e., $nextClaim_{[H]}$).

A. Experiment I

*1) **Methodology.*** The purpose of this experiment was to investigate the amount of the reward a miner earns with our approach versus the baseline (i.e., the full reward is based on all transaction fees). We investigated how $\mathbb{C}$ influences the total reward of the miner and how λ of the sliding window averaged the rewards. In detail, we created two independent $\mathcal{FRSC}$s with different λ – one was set to 2016 (i.e., $\mathcal{FRSC}^1$), and the second one was set to 5600 (i.e., $\mathcal{FRSC}^2$). We simulated these $\mathcal{FRSC}$s with three values of $\mathbb{C} \in \{0.5,\ 0.7,\ 0.9\}$.

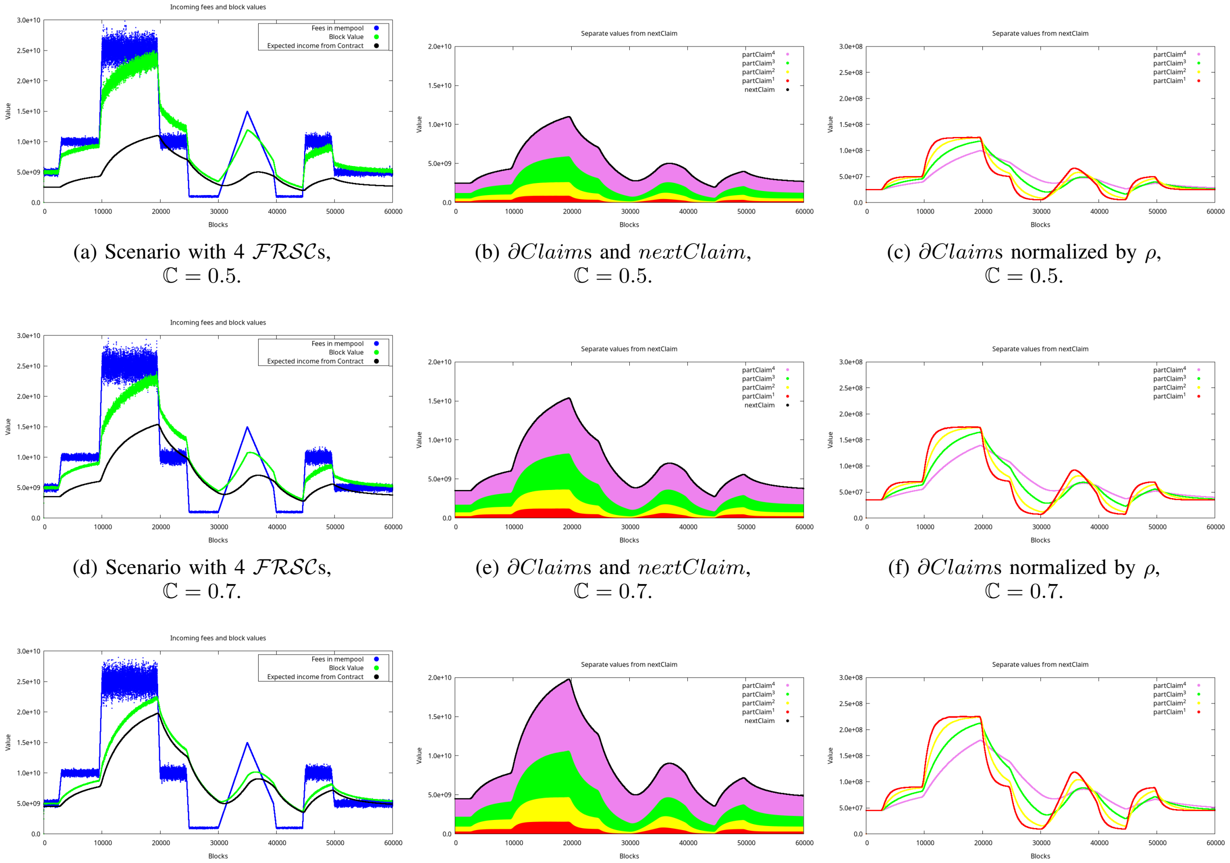
*2) **Results.*** The results of this experiment are depicted in Fig. 3. Across all runs of our experiment, we can observe that $\mathcal{FRSC}^2$ adapts slower as compared to $\mathcal{FRSC}^1$, which leads to a more significant averaging of the total reward paid to the miner.

B. Experiment II

*1) **Methodology.*** In this experiment, we investigated how multiple $\mathcal{FRSC}$s dealt with the same scenario as before – i.e., varying $\mathbb{C}$. In detail, we investigated how individual $\mathcal{FRSC}$s contributed to the $nextClaim_{[H+1]}$ by their individual $\partial Claim^{\mathcal{FRSC}_{[H]}}_{[H+1]}$. This time, we varied only the parameter $\mathbb{C} \in \{0.5,\ 0.7,\ 0.9\}$, and we considered four $\mathcal{FRSC}$s:

$$\mathcal{FRSC}s = \{$$
$$\mathcal{FRSC}^1(\_, 1008, 0.07), \mathcal{FRSC}^2(\_, 2016, 0.14),$$
$$\mathcal{FRSC}^3(\_, 4032, 0.28), \mathcal{FRSC}^4(\_, 8064, 0.51)\},$$

where their lengths λ were set to consecutive multiples of 2 (to see differences in more intensive averaging across longer intervals), and their redistribution ratios ρ were set to maximize the potential of averaging by longer $\mathcal{FRSC}$s.



(a) Scenario with 4 $\mathcal{FRSC}$s, $\mathbb{C} = 0.5$.

(b) $\partial Claims$ and $nextClaim$, $\mathbb{C} = 0.5$.

(c) $\partial Claims$ normalized by ρ, $\mathbb{C} = 0.5$.

(d) Scenario with 4 $\mathcal{FRSC}$s, $\mathbb{C} = 0.7$.

(e) $\partial Claims$ and $nextClaim$, $\mathbb{C} = 0.7$.

(f) $\partial Claims$ normalized by ρ, $\mathbb{C} = 0.7$.

(g) Scenario with 4 $\mathcal{FRSC}$s, $\mathbb{C} = 0.9$.

(h) $\partial Claims$ and $nextClaim$, $\mathbb{C} = 0.9$.

(i) $\partial Claims$ normalized by ρ, $\mathbb{C} = 0.9$.

Fig. 4: Experiment II investigating various $\mathbb{C}$s in the setting with multiple $\mathcal{FRSC}$s with their corresponding $\lambda = \{1008, 2016, 4032, 8064\}$ and $\rho = \{0.07, 0.14, 0.28, 0.51\}$. $\partial Claims$ represents contributions of individual $\mathcal{FRSC}$s to the total reward of the miner (i.e., its $nextClaim$ component).

*2) **Results.*** The results of this experiment are depicted in Fig. 4. We can observe that the shorter $\mathcal{FRSC}$s quickly adapted to new changes and the longer $\mathcal{FRSC}$s kept more steady income for the miner. In this sense, we can see that $\partial Claim^4$ held steadily over the scenario while for example $\partial Claim^1$ fluctuated more significantly. Since the scenarios of fees evolution in the mempool was the same across all our experiments (but § V-C), we can compare the $\mathcal{FRSC}$ with $\lambda = 5600$ from § V-A and the current setup involving four $\mathcal{FRSC}$s – both had some similarities. This gave us intuition for replacing multiple $\mathcal{FRSC}$s with a single one (see § V-C).

*3) **Different Fee Redistribution Ratios Across $\mathcal{FRSC}$s*** In Fig. 5 we investigated different values of ρ in the same set of four contracts and their impact on $\partial Claims$. The results show that the values of ρ should correlate with λ of multiple $\mathcal{FRSC}$s to maximize the potential of averaging by longer $\mathcal{FRSC}$s.

C. Experiment III

*1) **Methodology.*** In this experiment, we investigated whether it is possible to use a single $\mathcal{FRSC}$ setup to replace a multiple $\mathcal{FRSC}$s while preserving the same effect on the $nextClaim$. To quantify a difference between such cases, we introduced a new metric of $\mathcal{FRSC}s$, called effective_λ, which can be calculated as follows:

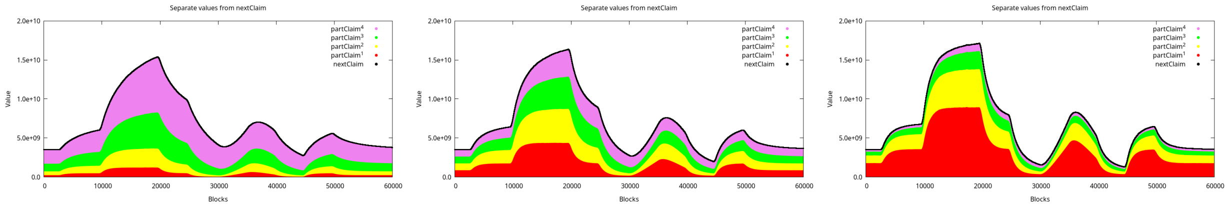
$$\text{effective\_}\lambda(\mathcal{FRSC}s) = \sum_{x\ \in\ \mathcal{FRSC}s} x.\rho * x.\lambda. \tag{13}$$

We were interested in comparing a single $\mathcal{FRSC}$ with 4 $\mathcal{FRSC}$s, both configurations having the equal effective_λ. The configurations of these two cases are as follows:

$$(1)\ \mathcal{FRSC}(\_, 5292, 1) \text{ and}$$
$$(2)\ \mathcal{FRSC}s = \{$$
$$\mathcal{FRSC}^1(\_, 1008, 0.07), \mathcal{FRSC}^2(\_, 2016, 0.19),$$
$$\mathcal{FRSC}^3(\_, 4032, 0.28), \mathcal{FRSC}^4(\_, 8064, 0.46)\}.$$

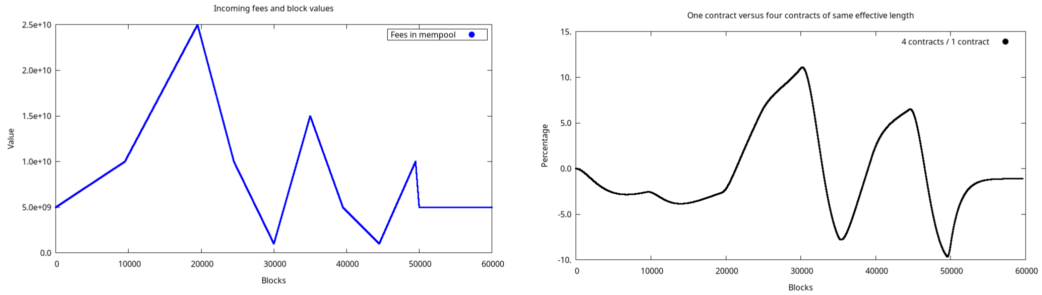
We can easily verify that the effective_λ of 4 $\mathcal{FRSC}$s is the same as in a single $\mathcal{FRSC}$ using Eq. 13: $0.07 * 1008 + 0.19 * 2016 + 0.28 * 4032 + 0.46 * 8064 = 5292$.

We conducted this experiment using a custom fee evolution scenario involving mainly linearly increasing/decreasing fees in the mempool (see Fig. 6a), and we set $\mathbb{C}$ to 0.7 for both configurations. The custom scenario of the fee evolution in mempool in this experiment was chosen to contain extreme changes in fees, emphasizing possible differences in two investigated setups.



(a) ρ correlates with λ. (b) ρ equal for every $\mathcal{FRSC}$. (c) ρ negatively correlates with λ.

Fig. 5: Experiment II – multiple $\mathcal{FRSC}$s using various distributions of ρ and their impact on $\partial Claim$, where $\mathbb{C}$ = 0.7.



(a) A custom fee scenario for Experiment III.

(b) A relative difference in $nextClaim$ between 4 $\mathcal{FRSC}$s and a single $\mathcal{FRSC}$.

Fig. 6: Experiment III comparing 4 $\mathcal{FRSC}$s and 1 $\mathcal{FRSC}$, both configurations having the same effective_λ.

*2) **Results.*** In Fig. 6b, we show the relative difference in percentages of $nextClaim$ rewards between the settings of 4 $\mathcal{FRSC}$s versus 1 $\mathcal{FRSC}$. It is clear that the setting of 4 $\mathcal{FRSC}$s in contrast to a single $\mathcal{FRSC}$ provided better reward compensation in times of very low fees value in the mempool, while it provided smaller reward in the times of higher values of fees in the mempool. Therefore, we concluded that it is not possible to replace a setup of multiple $\mathcal{FRSC}$s with a single one while retaining the same fee redistribution behavior.

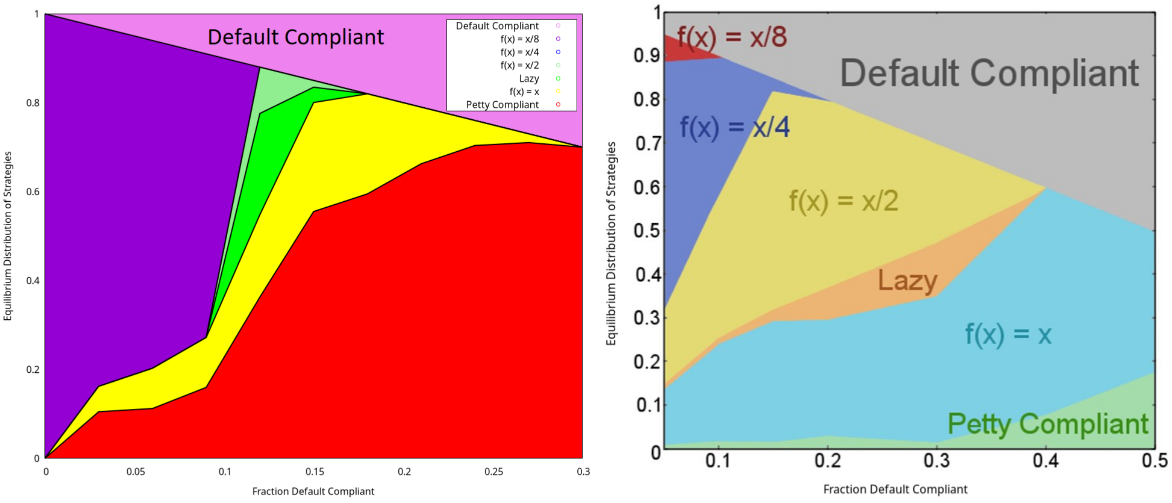
D. Experiment IV

We focused on reproducing the experiment from Section 5.5 of [2], while utilizing our approach. The experiment is aimed on searching for the minimal ratio of Default-Compliant miners, at which the undercutting attack is no longer profitable strategy. Default-Compliant miners are honest miners who follow the rules of the consensus protocol such as building on top of the longest chain. We executed several simulations, each consisting of multiple games (i.e., 300k as in [2]) with various fractions of Default-Compliant miners. From the remaining miners we evenly created *learning miners*, who learn on the previous runs of games and switch with a certain probability the best strategy out of the following:

- PettyCompliant: This miner behaves as Default-Compliant except one difference. In the case of seeing two chains, he does not mine on the oldest block but rather the most profitable block. Thus, this miner is not the (directly) attacking miner.
- LazyFork: This miner checks which out of two options is more profitable: (1) mining on the longest-chain block or (2) undercutting that block. In either way, he leaves half of the mempool fees for the next miners, which prevents another LazyFork miner to undercut him.
- Function-Fork() The behavior of the miner can be parametrized with a function f(.) expressing the level of his undercutting. The higher the output number the less reward he receives and more he leaves to incentivize other miners to mine on top of his block. This miner undercuts every time he forks the chain.

*1) **Methodology.*** With the missing feature for difficulty re-adjustment (in the simulator from [2] that we extended) the higher orphan rate occurs, which might directly impact our $\mathcal{FRSC}$-based approach. If the orphan rate is around 40%, roughly corresponding to [2], our blocks would take on average 40% longer time to be created, increasing the block creation time (i.e., time to mine a block). This does not affect the original simulator, as there are no $\mathcal{FRSC}s$ that would change the total reward for the miner who found the block.

Nevertheless, this is not true for $\mathcal{FRSC}$-based simulations as the initial setup of $\mathcal{FRSC}s$ is calculated with $\overline{fees} = 50$ BTC (as per the original simulations). However, with longer block creation time and transaction fees being calculated from it, the amount of $\overline{fees}$ also changes. With no adjustments, this results in $\mathcal{FRSC}s$ initially paying smaller reward back to the miner before $\mathcal{FRSC}s$ are saturated. To mitigate this problem, we increased the initial values of individual $\mathcal{FRSC}$s by the orphan rate from the previous game before each run. This results in very similar conditions, which can be verified by comparing the final value in the longest chain of our simulation versus the original simulations. We decided to use this approach to be as close as possible to the original experiment. This is particularly important when the full mempool parameter is equal to $false$ (see § IV), which means that the incoming transaction fees to mempool are calculated based on the block creation time. In our simulations, we used the following parameters: 100 miners, 10 000 blocks per game, 300 000 games (in each simulation



(a) Simulations of our approach.

(b) Simulations of the original work [2].

Fig. 7: Experiment IV – The ratio of DEFAULT-COMPLIANT miners in our approach is ∼30% (in contrast to ∼ 66% of [2]).

run), exp3 learning model, and $\mathbb{C} = 0.7$. Modeling of fees utilized the same parameters as in the original paper [2]: the full mempool parameter disabled, a constant inflow of 5 000 000 000 Satoshi (i.e., 50 BTC) every 600s. For more details about the learning strategies and other parameters, we refer the reader to [2].

Setup of $\mathcal{FRSC}s$. Since we have a steady inflow of fees to the mempool, we do not need to average the income for the miner. Therefore, we used only a single $\mathcal{FRSC}$ defined as $\mathcal{FRSC}$(7 056 000 000 000, 2016, 1), where the initial value of $\mathcal{FRSC}.\nu$ was adjusted according to Eq. 12, assuming $\overline{fees} = 50$ BTC. In the subsequent runs of each game, $\mathcal{FRSC}.\nu$ was increased by the orphan rate from the previous runs, as mentioned above.

2) ***Results.*** The results of this experiment, depicted in Fig. 7, demonstrate, that with our approach using $\mathcal{FRSC}$s, we decreased the number of DEFAULT-COMPLIANT miners from the original 66% to 30%. This means that the profitability of undercutting miners is avoided with at least 30% of DEFAULT-COMPLIANT miners, indicating more robust results.

VI. SECURITY ANALYSIS AND DISCUSSION

In this section, we analyze security implications of our solutions, we discuss its limitations and possible improvements.

A. Contract-Drying Attack

In this potential attack, the adversary aims to get his reward only from $\mathcal{FRSC}$s and does not include transactions in the block (or includes only a small number of them). This might result in slow drying of the funds from $\mathcal{FRSC}$s and would mean less reward for future honest miners. Moreover, the attacker can mine in "good" times of higher saturation of $\mathcal{FRSC}$s and, after some time, decide to switch off the mining. This might cause a deterioration in profitability for honest miners, consequently leading to deteriorated security w.r.t., undercutting attacks. However, the contract-drying attack is not profitable for the adversary and we state it only as the example of the abusive attack aimed at attacking the functionality of our scheme. If the attacker keeps repeating this behavior, the mempool of honest nodes will contain more un-mined transactions (and thus more fees). Therefore, if an honest miner mines a block, he gets a higher reward and at the same time deposits a higher amount to $\mathcal{FRSC}$s, which indicates a certain self-regulation of our approach, mitigating this kind of attack.

Next, we can think of further lowering the impact of this attack by rewarding the miner with the full $nextClaim_{[H+1]}$ by $\mathcal{FRSC}$s only if the block contains enough transaction fees (e.g., specified by the network-regulated parameter). However, this assumes that there is always a reasonable amount of fees in the mempool, which might not be the case all the time and might result in a situation where the miners temporarily stop mining due to the insufficient amount of available transactions. Nonetheless, it would require additional research to investigate solutions mitigating this type of abusive attack, which we left for future work.

B. Possible Improvements

$\mathcal{FRSC}$ can contain the parameter enabling the interval of the possible change in the reward paid by $\mathcal{FRSC}^x$ (i.e., $nextClaim^x_{[H+1]}$) from the median of its value (computed over λ). If $nextClaim^x_{[H+1]}$ would drastically increase from its median value as significantly more fees would come into the mempool, then $\mathcal{FRSC}^x_{[H]}$ would reward the miner with a certain value (specified by the parameter) from the interval $\langle average, nextClaim^x_{[H+1]} \rangle$ instead of the full $nextClaim^x_{[H+1]}$. This would be particularly useful for $\mathcal{FRSC}$s with a small λ parameter. The parameter used for sampling might contain a stochastic function (e.g., exponential), attributing a higher likelihood of getting the values not far from the median. However, we left the evaluation of such a technique to our future work, while in this work, we focused solely on demonstrating the feasibility of our approach.

C. Adjustment of Mining Difficulty

If the PoW blockchain with the longest chain fork-choice rule uses transaction-fee-based regime, the profitability of miners might be more volatile, which can further lead to decreased security w.r.t. undercutting attacks. Although our solution with $\mathcal{FRSC}$s helps in mitigation of this problem, we propose another functionality that resides in adjusting the mining difficulty based on the total collected fees during the epoch. In detail, the difficulty can be increased with higher fees collected from transactions during the epoch and vice versa. Further research would be needed to evaluate this proposition.

D. Out-of-Band Fees

Any fees paid outside the protocol, e.g., transaction fees not being paid in BTC, in the case of Bitcoin, are referred to as out-of-band fees. These fees can be leveraged e.g., by big exchanges, trusted wallet providers, etc. who control their mining pool and might influence transaction selection from the mempool, enabling to mine even low fee/0-fee transactions [10]. Under normal circumstances, such a transaction would not be chosen by a rational miner due to unprofitability. However, if the user has a mutual agreement with some miners, that upon including such transaction, they would pay them out-of-band fees, one of the miners will include this transaction. Even though out-of-band fees are not widely used nowadays, our fee redistribution scheme might lead to higher utilization of this concept. As with redistribution contracts, the miner would be losing his income in favor of a future miner; therefore, he might opt to receive a slightly smaller fee if this fee goes directly to him, i.e., circumventing $\mathbb{C}$ fraction. Also, the user would benefit from this, as he would be paying smaller fee overall, with the miner preferring out-of-band fees.

E. Practical Deployment

Our solution can be deployed by the hard-fork to enforce the execution of $\mathcal{FRSC}$s as part of the consensus protocol. The solution imposes only negligible performance overhead since the (optimized) logic of $\mathcal{FRSC}$s emulates the moving average, and the execution is triggered once per each block.

VII. Related work

The work of Carlsten et al. [2] is the inspiration for our paper, which for the first time describes undercutting attacks arising from the exponential distribution of block creation time and significant differences in transaction fees. The authors simulated Bitcoin under transaction-fee-based regime and found a minimal threshold of DEFAULT-COMPLIANT miners equal to 66%. Gong et al. [9] argue that using all fees accumulated in the mempool regardless of the block size limit is infeasible in practice and can inflate the profitability of undercutting that was originally described in [2]. Furthermore, Houy [11] demonstrates that a constraint on the block size limit (thus the number of transactions) has economic importance and prevents the transaction fees from dropping to zero. Therefore, Gong et al. [9] model the profitability of undercutting with the block size limit present, which bounds the claimable fees in a mining round. The authors presented a countermeasure that selectively assembles transactions into the new block, while claiming fewer fees to avoid undercutting. We argue that in contrast to our approach, this solution cannot be enforced by the consensus protocol, and thus might still enable undercutting to occur. Zhou et al. [12] deal with the problem of a mining gap, which is more significant when the throughput of blockchain is high. Therefore, the authors propose the self-adaptive algorithm to adjust the block size every 1000 blocks and thus ensure that blocks have enough space to pack new transactions. Even though Bitcoin-NG [13] proposes a new consensus mechanism, it also contains the idea of splitting the transaction fees between two entities – the current leader and the miner of block – which should incentivize the miner to include blocks created by the leader. However, Bitcoin-NG is, in some sense, centralized and therefore, undercutting attacks are not its subject.

VIII. Conclusion

We focused on 3 problems related to transaction-fee-based regime of blockchains with the longest chain rule: (1) the instability of mining rewards, (2) the mining gap, and (3) the possibility of undercutting attacks. To mitigate these problems, we proposed an approach approximating a moving average based on the fee-redistributions smart contracts (enforced by the consensus protocol) that accumulate a certain fraction of transaction fees and, at the same time, reward the miners from their reserves. This way, the miners are sufficiently rewarded even at times of very low transaction fees, such as at the beginning of the mining round, for new miners entering the mining protocol, during market deviations, etc.

IX. Acknowledgment

The work was mainly supported by the LoLiPoP IoT project, which has received funding from the KDT Joint Undertaking and the Czech Ministry of Education, Youth and Sports, grant agreement 101112286. Also, this work was supported by the FIT BUT internal project FIT-S-23-815.

References

[1] Q. Hum *et al.*, "Coinwatch: A clone-based approach for detecting vulnerabilities in cryptocurrencies," in *2020 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2020, pp. 17–25.

[2] M. Carlsten *et al.*, "On the instability of bitcoin without the block reward," in *ACM SIGSAC CCS*, 2016, pp. 154–167.

[3] P. Daian *et al.*, "Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability," in *2020 IEEE S&P*. IEEE, 2020, pp. 910–927.

[4] R. McLaughlin *et al.*, "A large scale study of the ethereum arbitrage ecosystem," in *USENIX Security '23*, 2023, pp. 3295–3312.

[5] I. Tsabary and I. Eyal, "The gap game," in *ACM SIGSAC CCS '18*, 2018, pp. 713–728.

[6] B. Wiki, "Miner Fees," 2022. [Online]. Available: https://en.bitcoin.it/wiki/Miner_fees

[7] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[8] H. Kalodner, "Bitcoin mining simulator," online, 2015. [Online]. Available: https://github.com/citp/mining_simulator

[9] T. Gong *et al.*, "Towards overcoming the undercutting problem," in *Financial Cryptography '22*. Springer, 2022, pp. 444–463.

[10] Bitcoin Stackexchange, "How did these zero-transaction fee transactions make it into the Bitcoin network?" 2018. [Online]. Available: https://bit.ly/3FASdmD

[11] N. Houy, "The economics of bitcoin transaction fees," *GATE WP*, vol. 1407, 2014.

[12] D. Zhou *et al.*, "A robust throughput scheme for bitcoin network without block reward," in *IEEE International Conference on High Performance Computing and Communications '19*. IEEE, 2019, pp. 706–713.

[13] I. Eyal *et al.*, "{Bitcoin-NG}: A scalable blockchain protocol," in *USENIX NSDI 16*, 2016, pp. 45–59.

BBB-Voting: Self-Tallying End-to-End Verifiable 1-out-of-k Blockchain-Based Boardroom Voting

Ivan Homoliak
Brno University of Technology
Faculty of Information Technology
ihomoliak@fit.vutbr.cz

Zengpeng Li
Shandong University
zengpeng@email.sdu.edu.cn

Pawel Szalachowski
Singapore University of Technology and Design
pjszal@gmail.com

***Abstract*—Voting is a means to agree on a collective decision based on available choices (e.g., candidates), where participants agree to abide by their outcome. To improve some features of e-voting, decentralized blockchain-based solutions can be employed, where the blockchain represents a public bulletin board that in contrast to a centralized bulletin board provides extremely high availability, censorship resistance, and correct code execution. A blockchain ensures that all entities in the voting system have the same view of the actions made by others due to its immutability and append-only features. The existing remote blockchain-based boardroom voting solution called Open Vote Network (OVN) provides the privacy of votes, universal & End-to-End verifiability, and perfect ballot secrecy; however, it supports only 2 choices and lacks recovery from stalling participants.**

We present BBB-Voting, an equivalent blockchain-based approach for decentralized voting such as OVN, but in contrast to it, BBB-Voting supports 1-out-of-k choices and provides robustness that enables recovery from stalling participants. We make a cost-optimized implementation using an Ethereum-based environment respecting Ethereum Enterprise Alliance standards, which we compare with OVN and show that our work decreases the costs for voters by 13.5% in normalized gas consumption. Finally, we show how BBB-Voting can be extended to support the number of participants limited only by the expenses paid by the authority and the computing power to obtain the tally.

I. Introduction

Voting is an integral part of democratic governance, where eligible participants can cast a vote for their representative choice (e.g., candidate or policy) through a secret ballot. The outcome of voting announces a tally of votes. Voting is usually centralized and suffers from a single point of failure that can be manifested in censorship, tampering, and issues with availability of a service. Blockchain is an emerging decentralized technology that provides interesting properties such as decentralization, censorship-resistance, immutability of data, correct execution of code, and extremely high availability, which can be harnessed in addressing the existing issues of e-voting. A few blockchain-based e-voting solutions have been proposed in recent years, mostly focusing on boardroom voting [1], [2], [3], [4] or small-scale voting [5], [6], [3].

Decentralization was a desired property of e-voting even before invention of blockchains. For example, (partially) decentralized e-voting that uses the homomorphic properties of El-Gamal encryption was introduced by Cramer et al. [7]. It assumes a threshold number of honest election authorities to provide the privacy of vote. However, when this threshold is adversarial, it does not protect from computing partial tallies, making statistical inferences about it, or even worse – the vote choices of participants. A solution that removed trust in tallying authorities was for the first time proposed by Kiayias and Yung [8] in their privacy-preserving self-tallying boardroom voting protocol. A similar protocol was later proposed by Hao et al. [9], which was later extended to a blockchain environment by McCorry et al. [1] in their Open Vote Network (OVN). An interesting property of OVN is that it requires only a single honest voting participant to maintain the privacy of the votes. However, OVN supports only two vote choices (based on [9]), assumes no stalling participants, and requires expensive on-chain tally computation (limiting its scalability). The scalability of OVN was improved by Seifelnasr et al. [5], but retaining the limitation of 2 choices and missing robustness.

Our goal is to build a remote boardroom voting protocol that resolves these limitations and enables a straightforward extension to support scalability. Therefore, we introduce BBB-Voting, a blockchain-based boardroom voting system providing 1-out-of-k voting, while additionally offering a mechanism for resolution of faulty participants. Alike OVN, BBB-Voting also provides the maximum privacy of votes in the setting that outputs the full tally of votes (as opposed to *tally-hiding* protocols [10], [11]). Both OVN and BBB-Voting require the authority whose role is limited to registering participants and shifting the phases of the protocol. The communication between the participants and the blockchain is semi-synchronous; i.e., each participant is expected to execute certain actions within a given time frame. When all registered participants submit their votes, the result can be tallied by anybody and the correctness of the result is verified by the blockchain.

Contributions. We make the following contributions.

i) We present BBB-Voting, an approach for remote end-to-end verifiable privacy-preserving self-tallying 1-out-of-k boardroom voting on the blockchain (see § IV). In detail, we start with the voting protocol proposed by Hao et al. [9] that provides a low bandwidth requirements and computational costs but is limited to 2 vote choices. We extend this protocol to support k choices utilizing the 1-out-of-k proof verification proposed by Kiayias and Yung [8]. We accommodate this approach to run on the blockchain with Turing-complete smart contract capability, enabling on-chain zero-knowledge proof verification

The work was supported by the DAIS project, which has received funding from the ECSEL Joint Undertaking and the Czech Ministry of Education, Youth and Sports, grant agreement 101007273. Also, this work was supported by the FIT BUT internal project FIT-S-23-815. Further, this research was partly supported by the National Key Research and Development Program of China under Grant No. 2021YFA1000600 as well as the National Natural Science Foundation of Shandong Province, China, under Grant No. ZR2023MF045 and the Natural Science Foundation of Qingdao, China, under Grant No. 23-2-1-152-zyyd-jch.

of blinded votes (and other proofs).

ii) We incorporate a robustness approach [12] into our protocol, which enables us to eliminate (even reoccurring) stalling (i.e., faulty) participants and thus finish voting without restarting the protocol (see § IV-B).

iii) We make two implementations, one based on discrete logarithm problem (DLP) for integers modulo p and the second one based on the elliptic curve DLP. For both implementations we propose various optimizations reducing the costs imposed by the blockchain platform (§ V-A). Due to the optimizations, our implementation (with elliptic curve DLP) increases the number of participants fitting a single block by 9% in contrast to OVN [1] under the same assumptions, while it decreases the costs for voters by 13.5%.

iv) We outline a scalability extension of our work, enabling the number of participants to be limited only by the expenses paid by the authority to register participants and compute their multi-party computation (MPC) keys as well as the computing power to obtain the tally. For demonstration purposes, we evaluate its utility in the context of the voting that is a magnitude greater than the boardroom voting (i.e., up to 1000 participants) while preserving almost the same per-participant costs paid by the authority as without this extension (see § VI-A).

II. Preliminaries

In this section, we describe voting terminology. We assume that the reader is familiar with blockchains and smart contracts.

An *involved party* refers to any stakeholder of the voting process and it covers *all participants* and the *authority*. A voting protocol is expected to meet several properties. A list of such properties appears in the works of Kiayias and Yung [8], Groth [13], and Cramer et al. [7].

(1) Privacy of Vote: ensures the secrecy of the ballot contents [8]. Hence, a participant's vote must not be revealed other than by the participant herself upon her discretion (or through the collusion of all remaining participants). Usually, privacy is ensured by trusting authorities in traditional elections or by homomorphic encryption in some decentralized e-voting solutions (e.g., [8], [9], [1], [5], [6]). **(2) Perfect Ballot Secrecy:** is an extension of the privacy of the vote. It implies that a partial tally (i.e., prior to the end of voting) is available only if all remaining participants are involved in its computation. **(3) Fairness:** ensures that a tally may be calculated only after all participants have submitted their votes. Therefore, no partial tally can be revealed to anyone before the end of the voting protocol [8]. **(4a) Universal Verifiability:** any involved party can verify that all cast votes are correct and they are correctly included in the final tally [8]. **(4b) End-to-End (E2E) Verifiability:** The verifiability of voting systems is also assessed by E2E verifiability [14], which involves *cast-as-intended*, *recorded-as-cast*, and *tallied-as-recorded* verifiability [15]. **(5) Dispute-Freeness:** extends the notion of verifiability. A dispute-free [8] voting protocol contains built-in mechanisms eliminating disputes between participants; therefore, anyone can verify whether a participant followed the protocol. Such a scheme has a publicly-verifiable audit trail that contributes to the reliability and trustworthiness of the scheme. **(6) Self-Tallying:** once all the votes are cast, any involved party can compute the tally. Self-tallying systems need to deal with the fairness issues (see (3) above) because the last participant is able to compute the tally even before casting her vote. This can be rectified with an additional verifiable dummy vote [8]. **(7) Robustness (Fault Tolerance):** the voting protocol is able to recover from faulty (stalling) participants, where faults are publicly visible and verifiable due to dispute-freeness [8]. Fault recovery is possible when all the remaining honest participants are involved in the recovery. **(8) Resistance to Serious Failures:** Serious failures are defined as situations in which voting results were changed either by a simple error or an adversarial attack. Such a change may or may not be detected. If detected in non-resistant systems, it is irreparable without restarting the entire voting [16].

III. System Model & Overview

Our system model has the following actor/components: (1) *a participant (P)* who votes, (2) *a voting authority (VA)* who is responsible for validating the eligibility of Ps to vote, their registration, and (3) *a smart contract (SC)*, which collects the votes, acts as a verifier of zero-knowledge proofs, enforces the rules of voting, and verifies the tally.

A. Adversary Model

The adversary $\mathcal{A}$ has bounded computing power, is unable to break used cryptographic primitives, and can control at most t of n participants during the protocol, where $t \leq n - 2 \wedge n \geq 3$. Any P under the control of $\mathcal{A}$ can misbehave during the protocol execution. $\mathcal{A}$ is also a passive listener of all communication entering the blockchain network but cannot block it or replace it with a malicious message since all transactions sent to the blockchain are authenticated by signatures of Ps or VA. Finally, VA **is only trusted in terms of identity management**, i.e., it performs identity verification of Ps honestly, and neither censor any P nor register any spoofed P. Nevertheless, no other trust in VA is required.

IV. BBB-Voting Scheme

BBB-Voting scheme provides all properties mentioned in § II. Similar to OVN [1], BBB-Voting publishes the full tally at the output and uses homomorphic encryption to achieve privacy of votes and perfect ballot secrecy. In detail, we extend the protocol of Hao et al. [9] to support k choices utilizing the 1-out-of-k proof verification proposed by Kiayias and Yung [8], and we accommodate this approach to run on the blockchain. Additionally, we extend our protocol to support the robustness, based on Khader et al. [12], which enables the protocol to recover (without a restart) from faulty participants who did not submit their votes. As a consequence, robustness increases the resistance of our protocol to serious failures.

A. Base Variant

The base variant of BBB-Voting does not involve a fault recovery and is divided into five stages: **registration** (identity verification, key ownership verification, enrollment at SC), a **setup** (an agreement on system parameters, submission of ephemeral public keys), **pre-voting** (computation of MPC keys), **voting** (vote packing, blinding, and verification), and **tally** phases. All faulty behaviors of Ps and VA are subject to deposit-based penalties. In detail, P who submitted her

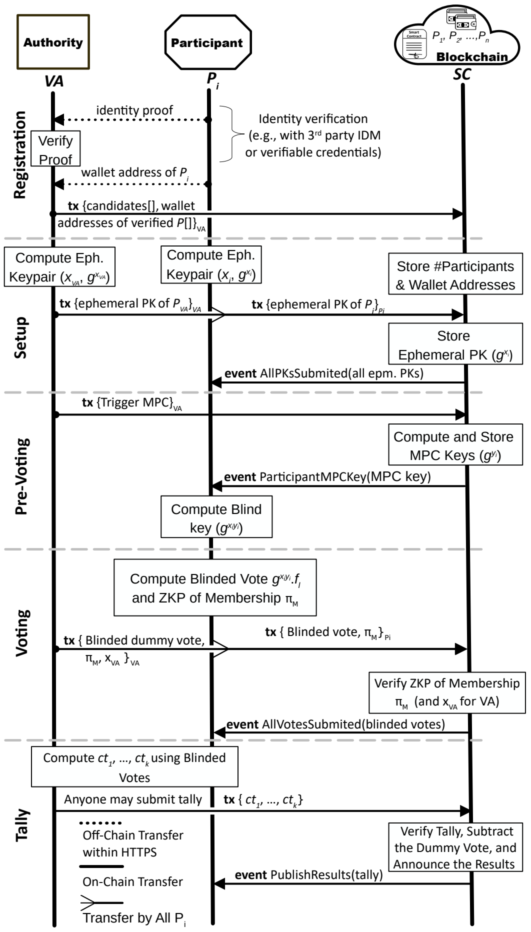

Authority
Participant
Smart Contract
$P_1, P_2, \ldots, P_n$
Blockchain
VA
P_i
SC
Registration
identity proof
Identity verification (e.g., with 3rd party IDM or verifiable credentials)
Verify Proof
wallet address of P_i
tx {candidates[], wallet addresses of verified P[]}$_{VA}$
Setup
Compute Eph. Keypair (x_{VA}, $g^{x_{VA}}$)
Compute Eph. Keypair (x_i, g^{x_i})
Store #Participants & Wallet Addresses
tx {ephemeral PK of P_{VA}}$_{VA}$
tx {ephemeral PK of P_i}$_{Pi}$
Store Ephemeral PK (g^{x_i})
event AllPKsSubmited(all epm. PKs)
Pre-Voting
tx {Trigger MPC}$_{VA}$
Compute and Store MPC Keys (g^{y_i})
event ParticipantMPCKey(MPC key)
Compute Blind key ($g^{x_iy_i}$)
Voting
Compute Blinded Vote $g^{x_iy_i}.f_l$ and ZKP of Membership π_M
tx { Blinded dummy vote, π_M, x_{VA} }$_{VA}$
tx { Blinded vote, π_M}$_{Pi}$
Verify ZKP of Membership π_M (and x_{VA} for VA)
event AllVotesSubmited(blinded votes)
Tally
Compute $ct_1, \ldots, ct_k$ using Blinded Votes
Anyone may submit tally
tx { $ct_1, \ldots, ct_k$}
Off-Chain Transfer within HTTPS
On-Chain Transfer
Transfer by All P_i
Verify Tally, Subtract the Dummy Vote, and Announce the Results
event PublishResults(tally)

Fig. 1: Basic protocol of BBB-Voting. $tx(param)_{\mathbb{A}}$ denotes a transaction signed by $\mathbb{A}$'s wallet private key.

ephemeral key (in the setup phase) and then has not voted within the timeout will lose the deposit. To achieve fairness, VA acts as the last P who submits a "dummy" vote with her ephemeral private key[1] after all other Ps cast their vote (or upon the voting timeout expiration).

*1) **Phase 1 (Registration)*** VA first verifies the identity proof of each P. For decentralized identity management (IDM), the identity proof is represented by the verifiable credentials (VC) [17] signed by the issuer, while in a centralized IDM the identity proof is interactively provided by a third-party identity provider (e.g., Google). First, VA verifies the issuer's signature on the identity proof. Next, VA challenges P to prove (using her VC) that she is indeed the owner of the identity. Further, each P creates her blockchain wallet address (i.e., the blockchain public key (PK)) and provides it to VA. The VA locally stores a bijective mapping between a P's identity and her wallet address.[2] Next, VA enrolls all verified Ps by sending their wallet addresses to SC.

*2) **Phase 2 (Setup)*** Ps agree on system parameters that are universal to voting – the parameters for voting are publicly visible on SC (deployed by VA in a transaction). Therefore, any P may verify these parameters before joining the protocol. Note the deployment transaction also contains the specification of timeouts for all further phases of the protocol as well as deposit-based penalties for misbehavior of VA and Ps. The parameters for voting are set as follows:

1) VA selects a common generator $g \in \mathbb{F}_p^*$. The value of p is chosen to be a safe prime, i.e., $p = 2 \cdot q + 1$, where q is a prime. A safe prime is chosen to ensure the multiplicative group of order $p - 1 = 2 \cdot q$, which has no small subgroups that are trivial to detect.[3] Let $n < p - 1$.
2) Any participant P_i is later permitted to submit a vote $\{v_i \mid i \in \{1, 2, ..., k\}\}$ for one of k choices. This is achieved by selecting k independent generators $\{f_1, ..., f_k\}$ in $\mathbb{F}_p^*$ (one for each choice). These generators for choices should meet a property described by Hao et al. [9] to preclude having two different valid tallies that fit Eq. 4:

$$f_i = \begin{cases} g^{2^0} & \text{for choice 1,} \\ g^{2^m} & \text{for choice 2,} \\ \quad \ldots & \\ g^{2^{(k-1)m}} & \text{for choice k,} \end{cases} \tag{1}$$

where m is the smallest integer such that $2^m > n$ (the number of participants).

Ephemeral Key Generation & Committing to Vote. Each P_i creates her ephemeral private key as a random number $x_i \in_R \mathbb{F}_p^*$ and ephemeral public key as g^{x_i}. Each P_i sends her ephemeral public key to SC in a transaction signed by her wallet, thereby, committing to submit a vote later.[4] Furthermore, P_i sends a deposit in this transaction, which can be retrieved back after the end of voting. However, if P_i does not vote within a timeout (or does not participate in a fault recovery (see § IV-B)), the deposit is lost, and it is split to the remaining involved parties. Ps who do not submit their ephemeral keys in this stage are indicating that they do not intend to vote; the protocol continues without them and they are not subject to penalties. Finally, each P obtains (from SC) the ephemeral public keys of all other verified Ps who have committed to voting. Ephemeral keys are one-time keys, and thus can be used only within one run of the protocol to ensure privacy of votes (other runs require fresh ephemeral keys).

*3) **Phase 3 (Pre-Voting)*** This phase represents multiparty computation (MPC), which is run to synchronize the keys among all Ps and achieve the *self-tallying* property. However, no direct interaction among Ps is required since all ephemeral public keys are published at SC. The MPC keys are computed by SC, when VA triggers the compute operation via a transaction. The SC computes and stores the MPC key for each P_i as follows:

$$h = g^{y_i} = \prod_{j=1}^{i-1} g^{x_j} / \prod_{j=i+1}^{n} g^{x_j}, \tag{2}$$

where $y_i = \sum_{j<i} x_j - \sum_{j>i} x_j$ and $\sum_i x_i y_i = 0$ (see Hao et al. [9] for the proof). While anyone can compute g^{y_i}, to reveal y_i, all Ps $\setminus$ P_i must either collude or solve the DLP

[1] Privacy for a dummy vote is not guaranteed since it is subtracted.

[2] Note that the address of P must not be part of identity proof – avoiding VA to possess a proof of identity to blockchain address mapping (see § VII-B).

[3] We use modular exponentiation by repeated squaring to compute $g^x \ mod \ p$, which has a time complexity of $\mathcal{O}((log\ x) \cdot (log^2\ p))$ [18].

[4] In contrast to OVN [1] (based on the idea from [9]), we do not require P_i to submit ZKP of knowledge of x_i to SC since P_i may only lose by submitting g^{x_i} to which she does not know x_i (i.e., a chance to vote + deposit).

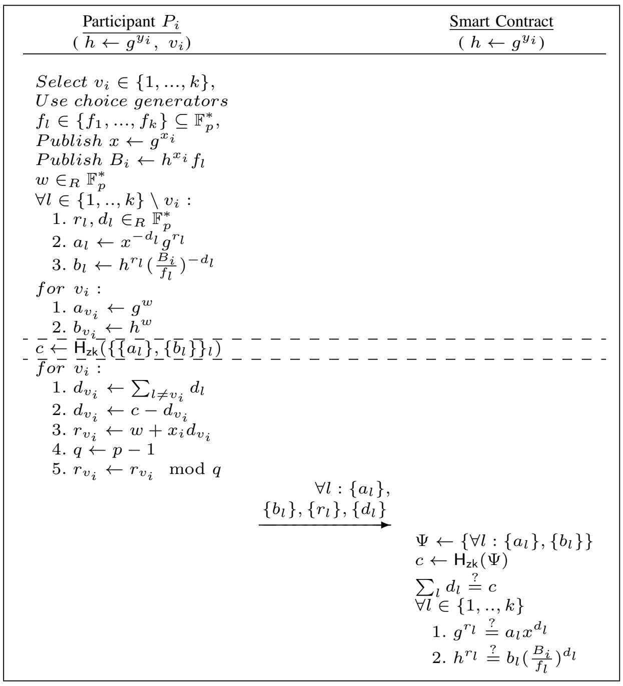

Participant P_i
$(h \leftarrow g^{y_i},\ v_i)$
Smart Contract
$(h \leftarrow g^{y_i})$
$Select\ v_i \in \{1, ..., k\},$
$Use\ choice\ generators$
$f_l \in \{f_1, ..., f_k\} \subseteq \mathbb{F}_p^*,$
$Publish\ x \leftarrow g^{x_i}$
$Publish\ B_i \leftarrow h^{x_i} f_l$
$w \in_R \mathbb{F}_p^*$
$\forall l \in \{1, .., k\} \setminus v_i :$
$1.\ r_l, d_l \in_R \mathbb{F}_p^*$
$2.\ a_l \leftarrow x^{-d_l} g^{r_l}$
$3.\ b_l \leftarrow h^{r_l} (\frac{B_i}{f_l})^{-d_l}$
$for\ v_i :$
$1.\ a_{v_i} \leftarrow g^w$
$2.\ b_{v_i} \leftarrow h^w$
$c \leftarrow \mathsf{H}_{\mathsf{zk}}(\{\{a_l\}, \{b_l\}\}_l)$
$for\ v_i :$
$1.\ d_{v_i} \leftarrow \sum_{l \neq v_i} d_l$
$2.\ d_{v_i} \leftarrow c - d_{v_i}$
$3.\ r_{v_i} \leftarrow w + x_i d_{v_i}$
$4.\ q \leftarrow p - 1$
$5.\ r_{v_i} \leftarrow r_{v_i} \mod q$
$\forall l : \{a_l\},$
$\{b_l\}, \{r_l\}, \{d_l\}$
$\Psi \leftarrow \{\forall l : \{a_l\}, \{b_l\}\}$
$c \leftarrow \mathsf{H}_{\mathsf{zk}}(\Psi)$
$\sum_l d_l \stackrel{?}{=} c$
$\forall l \in \{1, .., k\}$
$1.\ g^{r_l} \stackrel{?}{=} a_l x^{d_l}$
$2.\ h^{r_l} \stackrel{?}{=} b_l (\frac{B_i}{f_l})^{d_l}$

Fig. 2: ZKP of set membership for 1-out-of-k choices.

for Eq. 2. As the corollary of Eq. 2, the protocol preserves vote privacy if there are at least 3 Ps with at least 2 honest (we defer the proof to the journal version of the paper due to space constraints).

*4) **Phase 4 (Voting)*** In this phase, each P_i blinds and submits her vote to SC. These steps must ensure the recoverability of the tally, vote privacy, and well-formedness of the vote. Vote privacy is achieved by multiplying the P_i's blinding key with her vote choice. The blinded vote of the participant P_i is

$$B_i = \begin{cases} g^{x_i y_i} f_1 & \text{if } P_i \text{ votes for choice 1,} \\ g^{x_i y_i} f_2 & \text{if } P_i \text{ votes for choice 2,} \\ \quad ... & \\ g^{x_i y_i} f_k & \text{if } P_i \text{ votes for choice } k. \end{cases} \tag{3}$$

The P sends her choice within a blinded vote along with a 1-out-of-k non-interactive zero-knowledge (NIZK) proof of set membership to SC (i.e., proving that the vote choice $\in \{1, ..., k\}$). We modified the approach proposed by Kiayias et al. [8] to the form used by Hao et al. [9], which is convenient for practical deployment on existing smart contract platforms. The verification of set membership using this protocol is depicted in Fig. 2, where P_i is a prover and SC is the verifier. Hence, SC verifies the correctness of the proof and then stores the blinded vote. In this stage, it is important to ensure that no re-voting is possible, which is to avoid any inference about the final vote of P in the case she would change her vote choice during the voting stage. Such a re-voting logic can be enforced by SC, while user interface of the P should also not allow it. Moreover, to ensure fairness, VA acts as the last P who submits a dummy vote and her ephemeral private key.

*5) **Phase 5 (Tally)*** When the voting finishes (i.e., voting timeout expires or all Ps and VA cast their votes), the tally of votes received for each of k choices is computed off-chain by any party and then submitted to SC. When SC receives the tally, it verifies whether Eq. 4 holds, subtracts a dummy vote of VA, and notifies all Ps about the result. The tally is represented by vote counts $ct_i, \forall i \in \{1, ..., k\}$ of each choice, which are computed using an exhaustive search fitting

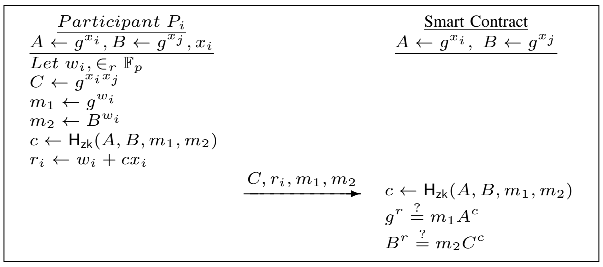

$Participant\ P_i$
$A \leftarrow g^{x_i}, B \leftarrow g^{x_j}, x_i$
Smart Contract
$A \leftarrow g^{x_i},\ B \leftarrow g^{x_j}$
$Let\ w_i, \in_r \mathbb{F}_p$
$C \leftarrow g^{x_i x_j}$
$m_1 \leftarrow g^{w_i}$
$m_2 \leftarrow B^{w_i}$
$c \leftarrow \mathsf{H}_{\mathsf{zk}}(A, B, m_1, m_2)$
$r_i \leftarrow w_i + c x_i$
C, r_i, m_1, m_2
$c \leftarrow \mathsf{H}_{\mathsf{zk}}(A, B, m_1, m_2)$
$g^r \stackrel{?}{=} m_1 A^c$
$B^r \stackrel{?}{=} m_2 C^c$

Fig. 3: ZKP verifying correspondence of $g^{x_i x_j}$ to public keys $A = g^{x_i}, B = g^{x_j}$.

$$\prod_{i=1}^{n} B_i = \prod_{i=1}^{n} g^{x_i y_i} f = g^{\sum_i x_i y_i} f = {f_1}^{ct_1} {f_2}^{ct_2} ... {f_k}^{ct_k}. \tag{4}$$

The maximum number of attempts is bounded by combinations with repetitions to $\binom{n+k-1}{k}$. Although the exhaustive search of 1-out-of-k voting is more computationally demanding in contrast to 1-out-of-2 voting [1], [9], this process can be heavily parallelized. See time measurements in § V-B.

B. Variant with Robustness

We extend the base variant of BBB-Voting by a fault recovery mechanism. If one or more Ps stall (i.e., are faulty) and do not submit their blinded vote in the voting stage despite committing in doing so, the tally cannot be computed directly. To recover from faulty Ps, we adapt the solution proposed by Khader et al. [12], and we place the fault recovery phase immediately after the voting phase. All remaining honest Ps are expected to repair their vote by a transaction to SC, which contains key materials shared with all faulty Ps and their NIZK proof of correctness. SC verifies all key materials with proofs (see Fig. 3), and then they are used to invert out the counter-party keys from a blinded vote of an honest P who sent the vote-repairing transaction to SC. Even if some of the honest (i.e., non-faulty) Ps would be faulty during the recovery phase (i.e., do not submit vote-repairing transaction), it is still possible to recover from such a state by repeating the next round of the fault recovery, but this time only with key materials related to new faulty Ps. To disincentivize stalling participants, they lose their deposits, which is split across remaining Ps as a compensation for the cost of fault recovery.

V. Implementation & Evaluation

We selected the Ethereum-based environment for evaluation due to its widespread adoption and open standardized architecture (driven by the Enterprise Ethereum Alliance [19]), which is incorporated by many blockchain projects. We implemented SC components in Solidity, while VA and P components were implemented in Javascript as testing clients of the *truffle* project. Executing smart contracts over blockchain, i.e., performing computations and storing data, has its costs. In Ethereum Virtual Machine (EVM), these costs are expressed by the level of execution complexity of particular instructions, referred to as *gas*. In this section, we analyze the costs imposed by our approach, perform a few optimizations, and compare

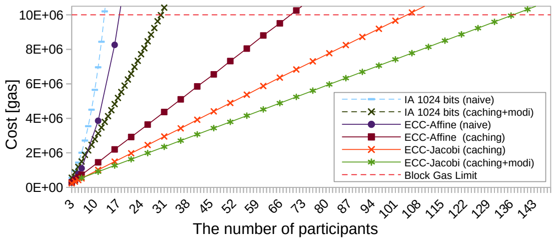


Fig. 4: A computation of MPC keys by the authority.

the costs with OVN [1]. In the context of this work, we assume 10M as the block gas limit. With block gas limit assumed, our implementation supports up to 135 participants (see Fig. 4), up to 7 vote choices (see Fig. 5a), and up to 9 simultaneously stalling faulty participants (see Fig. 5b).[5]

We made two different implementations, the first one is based on DLP for integers modulo p (denoted as integer arithmetic (IA)), and the second one is based on the elliptic curve DLP (denoted as ECC). In the ECC, we used a standardized *Secp256k1* curve from existing libraries [20], [1]. In the case of IA, we used a dedicated library [21] for operations with big numbers since EVM natively supports only 256-bit long words, which does not provide sufficient security level with respect to the DLP for integers modulo p.[6] We consider 1024 bits the minimal secure (library-supported) length of numbers in IA. As we will show below, IA implementation even with minimal 1024 bits is overly expensive, and thus in many cases does not fit the block gas limit by a single transaction. Therefore, in our experiments, we put emphasis on ECC implementation. The source code of our implementation is available at https://github.com/ivan-homoliak-sutd/BBB-Voting.

A. *Cost Optimizations*

Since ECC operations in ZKP verifications and computation of MPC keys impose a high execution cost when running at the blockchain, we have made several cost optimizations.

(1) Caching in MPC Key Computation. If implemented naively, the computation of all MPC keys in SC would contain a high number of overlapped additions, and hence the price would be excessively high (see series *"ECC-Affine (naive)"* in Fig. 4).[7] Therefore, in the code of SC we accumulate and reuse the value of the left side of MPC key computation during iteration through all participants. Similarly, the sum at the right side can be computed when processing the first participant, and then in each follow-up iteration, it can be subtracted by one item. However, we found out that subtraction imposes non-negligible costs since it contains one affine transformation (which we later optimize). In the result, we found pre-computation of all intermediary right items in the expression during the processing of the first participant as the most optimal. The resulting savings are depicted as *"ECC-Affine (caching)"* series in Fig. 4. We applied the same optimization for IA; however, even after adding a further optimization (i.e., pre-computation of modular inverses; see § V-A.4), the costs were still prohibitively high (see *"IA 1024 bits (caching+modi)"* series in Fig. 4), with the max. number of participants fitting the gas limit only 29.

(2) Affine vs. Jacobi Coordinates. In the ECC libraries employed [20] [1], by default, all operations are performed with points in Jacobi coordinates and then explicitly transformed to affine coordinates. However, such a transformation involves one modular inversion and four modular multiplications over 256-bit long integers, which is costly. Therefore, we maximized the utilization of internal functions from the Witnet library [20], which do not perform affine transformation after operation execution but keep the result in Jacobi coordinates. This is possible only until the moment when two points are compared – a comparison requires affine coordinates. Hence, a few calls of the affine transformation are inevitable. This optimization is depicted in Fig. 4 (series *"ECC-Jacobi (caching)"*) and Fig. 5a (series *"ECC-Jacobi"*). In the case of computation of MPC keys, this optimization brought improvement of costs by 23% in contrast to the version with affine coordinates and caching enabled. Due to this optimization, up to 111 participants can be processed in a single transaction not exceeding the block gas limit. In the case of vote submission, this optimization brought improvement of costs by 33% in contrast to the version with affine coordinates.

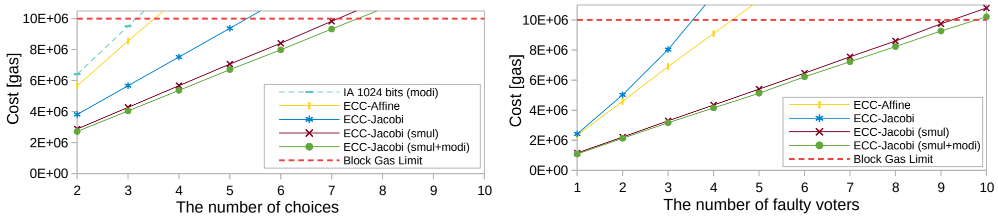
(3) Multiplication with Scalar Decomposition. The most expensive operation on an elliptic curve is a scalar multiplication; based on our experiments, it is often 5x-10x more expensive than the point addition since it involves several point additions (and/or point doubling). The literature proposes several ways of optimizing the scalar multiplication, where one of the most significant ways is w-NAF (Non-Adjacent Form) scalar decomposition followed by two simultaneous multiplications with scalars of halved sizes [23]. This approach was also adopted in the Witnet library [20] that we base on. The library boosts the performance (and decreases costs) by computing the sum of two simultaneous multiplications $kP + lQ$, where $k = (k_1 + k_2\lambda)$, $l = (l_1 + l_2\lambda)$, and λ is a specific constant to the endomorphism of the elliptic curve. To use this approach, a scalar decomposition of k and l needs to be computed beforehand. Nevertheless, such **a scalar decomposition can be computed off-chain (and verified on-chain)**, while only a simultaneous multiplication is computed on-chain. However, to leverage the full potential of the doubled simultaneous multiplication, one must have the expression $kP + lQ$, which is often not the case. In our case, we modified the check at SC to fit this form. Alike the vote submission, this optimization can be applied in vote repair. We depict the performance improvement brought by this optimization as series *"ECC-Jacobi (smul)"* in Fig. 5.

(4) Pre-Computation of Modular Inversions. Each affine transformation in the vote submission contains one operation of modular inversion – assuming previous optimizations, ZKP verification of one item in 1-out-of-k ZKP requires three affine transformations (e.g., for $k = 5$, it is 15). Similarly, the ZKP verification of correctness in the repair vote requires two affine transformations per each faulty participant submitted. The modular inversion operation runs the extended Euclidean algorithm, which imposes non-negligible costs. However, all modular inversions **can be pre-computed off-chain, while only their verification can be made on-chain** (i.e., modular

[5] The max. corresponds to a single recovery round but the total number of faulty participants can be unlimited since the fault recovery round can repeat.

[6] Since this DLP was already computed for 795-bit long safe prime in 2019 [22], only values higher than 795-bit are considered secure enough.

[7] The same phenomenon occurs in IA (see Eq. 2) but with overlapped multiplications (see series *"IA 1024 bits (naive)"*).



(a) Vote submission by P_i.

(b) Vote repair by P_i.

Fig. 5: Vote submission and vote repair (i.e., fault recovery) with various optimizations.

multiplication), which imposes much lower costs. We depict the impact of this optimization as *"ECC-Jacobi (smul+modi)"* series in Fig. 5 and "...modi..." series in Fig. 4. In the result, it has brought 5% savings of costs in contrast to the version with the simultaneous multiplication.

B. Tally Computation

In Tab. I, we provide time measurements of tally computation through the entire search space on 1 core vs. all cores of the i7-10510U CPU laptop.[8] We see that for $n \leq 100$ and $k \leq 6$, the tally can be computed even on a commodity PC in a reasonable time. However, for higher n and k, we recommend using a more powerful machine or distributed computation across all Ps. One should realize that our measurements correspond to the upper bound, and if some ranges of tally frequencies are more likely than other ones, they can be processed first – in this way, the computation time can be significantly reduced. Moreover, we emphasize that an exhaustive search for tally computation is not specific only to our scheme but to homomorphic-encryption-based schemes providing perfect ballot secrecy and privacy of votes (e.g., [8], [9], [1]).

C. Cost Comparison

In Tab. II, we made a cost comparison of BBB-Voting (using ECC) with OVN [1], where we assumed two choices and 40 participants (the same setting as in [1]). We see that the total costs are similar but BBB-Voting improves P's costs by 13.5% and VA's cost by 0.9% even though using more complex setting that allows 1-out-of-k voting. We also emphasize that the protocol used for vote casting in BBB-Voting **contains more operations** than OVN but regardless of it, the costs are close to those of OVN, which is mostly caused by the proposed optimizations.[9] Next, we found that OVN computes tally on-chain, which is an expensive option. In contrast, BBB-Voting computes tally off-chain and SC performs only verification of its correctness, which enables us to minimize the cost of this operation. Another gas saving optimization of BBB-Voting in contrast to OVN (and Hao et al. [9]) is that we do not require voters to submit ZKP of knowledge of x_i in g^{x_i} during the registration phase to SC since P_i may only lose by providing incorrect ephemeral public key g^{x_i} – she might lose the chance to vote and her deposit. Finally, we note that we consider the deployment costs of our SC equal to 4.8M units of gas; however, our SC implementation contains a few auxiliary view-only functions for a pre-computation of modular inverses, with which, the deployment costs would increase to 7.67M due to code size. Nevertheless, these operations can be safely off-chained and we utilized them on-chain only for simplicity.

VI. Discussion of Extensions

In this section, we discuss the extensions addressing the scalability and performance limitations of BBB-Voting.

A. Scalability Limitation & Extension

The limitation of BBB-Voting (like in OVN) is a lack of scalability, where the block gas limit might be exceeded with a high number of Ps. Therefore, we primarily position our solution as boardroom voting; however, we will show in this section that it can be extended even to larger voting. Our voting protocol (see § IV-A) has a few platform-specific bottlenecks. The first bottleneck is in the setup phase, where VA submits the wallet addresses of all Ps to SC in a single transaction, which might exceed the block gas limit when the number of participants $n > 201$. The second bottleneck occurs in the pre-voting phase, where VA calls the function of SC to compute all MPC keys; exceeding the block gas limit occurs when the number of participants $n > 135$. The next bottleneck occurs in the voting phase, where voters submit their blinded votes together with 1-out-of-k ZKP, exceeding the block gas limit when the number of choices $k > 7$. The last bottleneck occurs in the fault recovery phase and the block gas limit is exceeded when the number of simultaneously faulty participants $f > 9$.

Voters	**Choices**			
(n)	$k = 2$	$k = 4$	$k = 6$	$k = 8$
20	$0.01s$	$0.01s$	$0.07s$	$0.07s$
30	$0.01s$	$0.01s$	$0.53s$	$13.3s$
40	$0.01s$	$0.04s$	$02.6s$	$112s$
50	$0.01s$	$0.08s$	$10.0s$	$606s$
60	$0.01s$	$0.16s$	$28.2s$	$2424s$
70	$0.01s$	$0.48s$	$69.6s$	$\sim 2.1h$
80	$0.01s$	$0.82s$	$160s$	$\sim 5.8h$
90	$0.01s$	$1.08s$	$320s$	$\sim 14.2h$
100	$0.01s$	$1.2s$	$722s$	$\sim 33h$

(a) 1 core

Voters	**Choices**	
(n)	$k = 6$	$k = 8$
20	$0.01s$	$0.01s$
30	$0.08s$	$2.0s$
40	$0.39s$	$16.8s$
50	$1.5s$	$90.9s$
60	$4.44s$	$267s$
70	$11.85s$	$773s$
80	$19.46s$	$2210s$
90	$44.02s$	$\sim 2.7h$
100	$108.3s$	$\sim 4.9h$

(b) 8 cores

Tab. I: Upper time bound for tally computation.

	Gas Paid by	**OVN**	**BBB-Voting**
Deployment of Voting SC	VA	3.78M	4.8M
Deployment of Cryptographic SCs	VA	2.44M	2.15M (1.22M+0.93M)
Enroll voters	VA	2.38M (2.15M+0.23M)	1.93M
Submit Ephemeral PK	P	0.76M	0.15M
Cast Vote	P	2.50M	2.72M
Tally	VA *(or* P*)*	0.75M	0.39M
Total Costs for P		3.26M	2.87M
Total Costs for VA		9.35M	9.27M

Tab. II: A normalized cost comparison of BBB-Voting with OVN for $n = 40$ and $k = 2$.

[8] In some cases we estimated the time since we knew the number of attempts.

[9] To verify 1-out-of-k ZKP in vote casting, BBB-Voting computes $5 \cdot k$ multiplications and $3 \cdot k$ additions on the elliptic curve – i.e., 10 multiplications and 6 additions for $k = 2$. In contrast, OVN computes only 8 multiplications and 5 additions for $k = 2$.

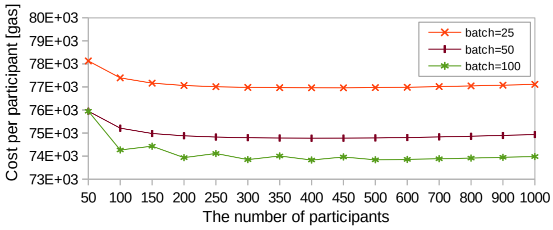


Fig. 6: A computation of MPC keys by the authority VA using various batch sizes and the most optimized ECC variant.

Nevertheless, transaction batching can be introduced for the elimination of all these bottlenecks. To realize a batching of pre-voting, voting, and fault-recovery phases, the additional integrity preservation logic across batches needs to be addressed while the verification of integrity has to be made by SC. For demonstration purposes, we addressed the bottleneck of the pre-voting stage (see Fig. 6) and setup stage, which further improves the vote privacy in BBB-Voting (see § VII-B) and causes only a minimal cost increase due to the overhead of integrity preservation (i.e., 1%). With this extension, n is limited only by the expenses paid by VA to register Ps and compute their MPC keys, and the computing power to obtain the tally.[10] If a certain combination of high n and k would make computation of a tally overly computationally expensive (or the cost of its verification by SC), it is further possible to partition Ps into multiple groups (i.e., voting booths), each managed by an instance of BBB-Voting, while the total results could be summed across instances. Scalability evaluation and other operations is subject to our future work, which provides more empirical evidence [24].

B. Cost and Performance Limitations

Although we thoroughly optimized the costs (and thus performance) of our implementation (see § V-A), the expenses imposed by a public permissionless smart contract platform might be still high, especially during peaks of the gas price and/or in the case of a larger voting than boardroom voting (see § VI-A). Besides, the transactional throughput of such platforms might be too small for such larger voting instances to occur in a specified time window. Therefore, to further optimize the costs of BBB-Voting and its performance, it can run on a public permissioned Proof-of-Authority platforms, e.g., using Hyperledger projects (such as safety-favored Besu with Byzantine Fault Tolerant (BFT) protocol). Another option is to use smart contract platforms utilizing the trusted computing that off-chains expensive computations (e.g., Ekiden [25], TeeChain [26], Aquareum [27]), or other partially-decentralized second layer solutions (e.g., Plasma [28], Polygon Matic [29], Hydra [30]). Even though these solutions might preserve most of the blockchain features harnessed in e-voting, availability and decentralization might be decreased, which the security/performance trade-off.

VII. Security Analysis

We first analyze security of BBB-Voting with regard to the voting properties specified in § II. Next, we analyze blockchain-specific security & privacy issues and discuss their mitigations. The security of our scheme relies on well-known cryptographic primitives under their standard assumptions.

A. Properties of Voting

(1) Privacy in BBB-Voting requires at least 3 Ps, out of which at least 2 are honest (see § IV-A). Privacy in BBB-voting is achieved by blinding votes using ElGamal encryption [31], whose security is based on the decisional Diffie-Hellman assumption. Unlike the conventional ElGamal algorithm, a decryption operation is not required to unblind the votes. Instead, we rely on the self-tallying property of our voting protocol. The ciphertext representing a blinded vote is a tuple (c_1, c_2), where $c_1 = g^{xy}.f$ and $c_2 = g^y$, where the purpose of c_2 is to assist with the decryption. Decryption involves computing $(c_2)^{-x} \cdot c_1$ to reveal f, which unambiguously identifies a vote choice. As a result, the blinding operation for participant P_i in Eq. 3 is equivalent to ElGamal encryption involving the computation of c_1 but not the decryption component c_2. Furthermore, the blinding keys are ephemeral and used exactly once for encryption (i.e., blinding) of the vote within a single run of voting protocol[11] – i.e., if the protocol is executed correctly, there are no two votes f_l and f_m encrypted with the same ephemeral blinding key of P_i, such that

$$\frac{(g^{xy} \cdot f_l)}{(g^{xy} \cdot f_m)} = \frac{f_l}{f_m}, \tag{5}$$

from which the individual votes could be deduced. For the blockchain-specific privacy analysis, see also § VII-B.

(2) Ballot Secrecy. It is achieved by blinding the vote using ElGamal homomorphic encryption [7], and it is not required to possess a private key to decrypt the tally because of the self-tallying property ($g^{\sum_i x_i y_i} = 1$). Therefore, given a homomorphic encryption function, it is possible to record a sequence of encrypted votes without being able to read the votes choices. However, if all Ps are involved in the recovery of a partial tally consisting of a recorded set of votes, these votes can be unblinded (as allowed by ballot secrecy). Even a subset of $n - 2$ Ps[12] who have already cast their votes cannot recover a partial tally that reveals their vote choices because of the self-tallying property ($g^{\sum_i x_i y_i} = 1$) has not been met.

(3) Fairness. If implemented naively, the last voting P can privately reveal the full tally by solving Eq. 4 before she casts her vote since all remaining blinded votes are already recorded on the blockchain (a.k.a., the last participant conundrum). This can be resolved by VA who is required to submit the final dummy vote including the proof of her vote choice, which is later subtracted from the final tally by SC.[13]

(4a) Universal Verifiability. Any involved party can check whether all recorded votes in the blockchain are correct and are correctly included in the final tally [8]. Besides, the blinded votes are verified at SC, which provides correctness of its execution and public verifiability, relying on the honest majority of the consensus power in the blockchain (see § VII-B).

(4b) E2E Verifiability. To satisfy E2E verifiability [2]: (I) each P can verify whether her vote was cast-as-intended and

[10] E.g., for $n = 1000$, $k = 2$ (and $k = 4$), it takes 0.15s (and $\sim$ 4h) to obtain the tally on a commodity PC with 8 cores, respectively.

[11] As a consequence, BBB-Voting can be utilized in a repetitive voting [32] with a limitation of a single vote within an epoch.

[12] Note that at least 2 Ps are required to be honest (see § III-A).

[13] Note that If VA were not to execute this step, the fault recovery would exclude VA's share from MPC keys, and the protocol would continue.

recorded-as-cast, (II) anyone can verify whether all votes are tallied-as-recorded. BBB-Voting meets (I) since each P can locally compute her vote choice (anytime) and compare it against the one recorded in the blockchain. BBB-Voting meets (II) since SC executes the code verifying that the submitted tally fits Eq. 4 that embeds all recorded votes.

(5) Dispute-Freeness. Since the blockchain acts as a tamper-resistant bulletin board (see § VII-B), and moreover it provides correctness of code execution (i.e., on-chain execution of verification checks for votes, tally, and fault recovery shares) and verifiability, the election remains dispute-free under the standard blockchain assumptions about the honest majority and waiting the time to finality.

(6) Self-Tallying. BBB-Voting meet this property since in the tally phase of our protocol (and anytime after), all cast votes are recorded in SC; therefore any party can use them to fit Eq. 4, obtaining the final tally.

(7) Robustness (Fault Tolerance). BBB-Voting is robust since it enables to remove (even reoccurring) stalling Ps by its fault recovery mechanism (see § IV-B). Removing of stalling Ps involves SC verifiability of ZKP submitted by Ps along with their counter-party shares corresponding to stalling Ps.

(8) Resistance to Serious Failures. The resistance of BBB-Voting to serious failures relies on the integrity and append-only features of the blockchain, which (under its assumptions § VII-B) does not allow the change of already cast votes.

B. Blockchain-Specific Aspects and Issues

In the following, we focus on the most important blockchain-specific aspects and issues related to BBB-Voting.

(1) Bulletin Board vs. Blockchains. The definition of a bulletin board [8] assumes its immutability and append-only feature, which can be provided by blockchains that moreover provide correct execution of code. CAP theorem [33] enables a distributed system (such as the blockchain) to select either **c**onsistency or **a**vailability during the time of network **p**artitions. If the system selects consistency (e.g., Algorand [34], BFT-based blockchains such as [35]), it stalls during network partitions and does not provide liveness (i.e., the blocks are not produced) but provides safety (i.e., all nodes agree on the same blocks when some are produced). On the other hand, if the system selects availability (e.g., Bitcoin [36], Ethereum [37]), it does not provide safety but provides liveness, which translates into possibility of creating *accidental forks* and eventually accepting one as valid. Many public blockchains favor availability over consistency, and thus do not guarantee immediate immutability. Furthermore, blockchains might suffer from *malicious forks* that are longer than accidental forks and are expensive for the attacker. Usually, their goal is to execute double-spending or selfish mining [38], violating the assumptions of the consensus protocol employed – more than 51% / 66% of honest nodes presented in PoW / BFT-based protocols. To prevent accidental forks and mitigate malicious forks in liveness-favoring blockchains, it is recommended to wait for a certain number of blocks (a.k.a., block confirmations). Another option to cope with forks is to utilize safety-favoring blockchains (e.g., [34], [35]).

Considering BBB-Voting, we argue that these forks are not critical for the proposed protocol since any transaction can be resubmitted if it is not included in the blockchain after a fork. Waiting for the time to finality (with a potential resubmission) can be done as a background task of the client software at Ps' devices, so Ps do not have to wait. Finally, we emphasize that the time to finality is negligible in contrast to timeouts of the protocol phases; therefore, there is enough time to make an automatic resubmission if needed.

(2) Privacy of Votes. In BBB-Voting, the privacy of vote choices can be "violated" only in the case of unanimous voting by all Ps, assuming $\mathcal{A}$ who can link the identities of Ps (approximated by their IP addresses) to their blockchain addresses by passive monitoring of network traffic. However, this is the acceptable property in the class of voting protocols that provide the full tally of votes at the output, such as BBB-Voting and other protocols (e.g., [1], [8], [9], [39], [4]). Moreover, $\mathcal{A}$ can do deductions about the probability of selecting a particular vote choice by Ps. For example, in the case that the majority m of all participants n voted for a winning vote choice, then $\mathcal{A}$ passively monitoring the network traffic can link the blockchain addresses of Ps to their identities (i.e., IP addresses), and thus $\mathcal{A}$ can infer that each P from the group of all Ps cast her vote to the winning choice with the probability equal to $\frac{m}{n} > 0.5$. However, it does not violate the privacy of votes and such an inferring is not possible solely from the data publicly stored at the blockchain since it stores only blinded votes and blockchain addresses of Ps, not the identities of Ps. To mitigate these issues, Ps can use anonymization networks or VPN services for sending transactions to the blockchain. Moreover, neither $\mathcal{A}$ nor VA can provide the public with the indisputable proof that links P's identity to her blockchain address.

(3) Privacy of Votes in Larger Voting. The privacy issue of unanimous and majority voting (assuming $\mathcal{A}$ with network monitoring capability) are less likely to occur in the larger voting than boardroom voting since the voting group of Ps is larger and potentially more divergent. We showed that BBB-Voting can be extended to such a large voting by integrity-preserving batching in § VI-A. We experimented with batching up to 1000 Ps, which is a magnitude greater voting than the boardroom voting. We depict the gas expenses paid by VA (per P) in Fig. 6, where we distinguish various batch sizes. In sum, the bigger the batch size, the lower the price per P.

VIII. Related Work

In this section, we briefly survey existing paradigms in e-voting and describe a few blockchain-based e-voting approaches. In particular, we focus on **remote voting** approaches (with sufficient specification), which we compare in Tab. III.

E-Voting Paradigms. Utilization of mix-nets that shuffle the votes to break the map between Ps and their votes was proposed by Chaum [40]. Benaloh and Fischer [41] were among the first who showed a paradigm shift from anonymizing Ps to providing privacy of the vote. Cramer et al. [7] present a model where all votes are sent to a single combiner, utilizing homomorphic properties of the ElGamal cryptosystem [31]. Using bulletin board and zero-knowledge proofs allow their protocol to be universally verifiable. The work of Kiayias and Yung [8] converts this scheme into a self-tallying combiner supporting 1-out-of-2 choices; further, the authors outline an extension of their base protocol to support 1-out-of-k choices. Hao et al. [9] improve upon the self-tallying protocol by proposing a simple general-purpose two-round voting protocol for 1-out-of-2 choices with low bandwidth requirements and computational costs. Khader et al. [12] take it a step further by adding fairness and robustness properties. Groth [13] introduces an anonymous broadcast channel with perfect message secrecy leveraged in his voting protocol that is simpler and more efficient than [8]. However, it requires sequential voting, where each voter has to download a fresh state of the bulletin board before voting. Zagorski et al. [42] propose Remotegrity that is based on Scanintegrity [43] ballots mailed to voters, allowing them remotely vote and verify that their ballots were correctly posted to the bulletin board and at the same time providing protection against malware in clients. Another direction (e.g., [10], [11]) focuses on the tally-hiding [44], [45] property that enable to reveal only the best m candidates.

Location. Voting systems can be classified by the physical location where the vote is cast. Some schemes allow Ps to submit a vote from their devices (i.e., *remote voting*), e.g., [46], [47], [42], and blockchain-based [48], [49]. Others systems require voting to be carried out at a designated site (a.k.a., *supervised voting*), e.g., [43], [50], [51], [52], and blockchain-based [53], [54], [55].

A. Blockchain-Based E-Voting

We extend the categorization of (remote) blockchain-based voting [4], and we focused on smart contract-based systems.

(1) Voting Systems Using Smart Contracts. McCorry et al. [1] proposed OVN, a self-tallying voting protocol (basing on [9]) that provides vote privacy and supports two vote choices. OVN is implemented as Ethereum SC and is suitable for boardroom voting. It does not provide robustness and expensive tally computation is made by SC. In contrast, BBB-Voting performs only tally verification in SC, while it is computed off-chain. Similar approach basing on [9] was proposed by Li et al. [3], who further provided robustness from [12]. Seifelnasr et al. [5] aimed to increase the scalability of OVN by off-chaining tally computation and registration at VA in a verifiable way. Due to the higher costs imposed by storing data on SC, they compute the Merkle tree of voter identities and store only its root hash at SC. Their approach requires only a single honest P to maintain the protocol's security by enabling her to dispute the incorrect tally submitted to SC. The scalability technique proposed in this paper is orthogonal to us, and it can be combined with our techniques (see § VI-A) to optimize on-chain costs. Yu et al. [4] employ ring signature to ensure that the ballot is from one of the valid choices, and they achieve scalability by linkable ring signature key accumulation. Their approach provides receipt-freeness under the assumption of trusted VA. However, due to receipt-freeness, this approach does not provide E2E verifiability. Killer et al. [39] present an E2E verifiable remote voting scheme with two vote choices. The authors employ threshold cryptography for achieving robustness using a scheme similar to Shamir secret sharing. However, it supports only integers up to a 256 bits (i.e., a size of the EVM word), which is far below a minimal secure length. Matile et al. [56] proposed a voting system providing cast-as-intended (but neither E2E nor universal) verifiability. Their system uses ElGamal encryption based on DLP with integers modulo p. Since existing blockchains support natively only up to 256-bit security for this DLP, the authors create the custom blockchain with sufficient security. Dagher et al. [6] proposed BroncoVote, a voting system that preserves vote privacy by homomorphic encryption (i.e., Paillier cryptosystem) – the authors off-chain all cryptographic operations to a trusted server (without verifiability), which introduces a vulnerability. Kostal et al. [57] propose voting system, in which VA serves as a trusted key generator that distributes private keys for homomorphic encryption to voters, enabling them to resolve robustness issues at the cost of putting a trust into VA.

Approach	**Privacy of Votes**	**Perfect Ballot Secrecy**	**Fairness**	**Self-Tallying**	**Robustness**	**Uses Blockchain**	**Uni. Verifiability**	**E2E Verifiability**	**Open Source**	**Choices**
Hao et al. [9]	✓	✓	✗	✓	✗	✗	✓	✓	✗	2
Khader et. [12]	✓	✓	✓	✓	✓	✗	✓	✓	✗	2
Kiayias and Yung [8]	✓	✓	✓	✓	✓	✗	✓	✓	✗	$2/k$
McCorry et. [1] (OVN)	✓	✓	✓	✓	✗	✓	✓	✓	✓	2
Seifelnasr et al. [5] (sOVN)	✓	✓	✓	✓	✗	✓	✓	✓	✓	2
Li et al. [3]	✓	✓	✓	✓	✓	✓	✓	✓	✗	2
Baudron et al. [61]	✓	✗	✗	✗	✓	✗	✓	✗	✗	k
Groth [13]	✓	✓	✓	✓	✗	✗	✓	✓	✗	k
Adida [47] (Helios)	✓*	✗	✗	✗	✓	✗	✓	✓	✓	k
Matile et al. [56] (CaIV)	✓	✗	✗	✗	✓	✓	✗	✗	✓	k
Killer [39] (Provotum)	✓	✗	✓	✗	✓	✓	✓	✓	✓	2
Dagher et al. [6] (BroncoVote)	✓	✓*	✓*	✗	✗	✓	✗	✗	✗	k
Kostal et al. [57]	✓*	✓*	✗	✓	✓*	✓	✓	✓	✗	k
Zagorski et al. [42] (Remotegrity)	✓*	✗	✗	✗	✓	✗	✗	✓	✓	k
Yu et al. [4]	✓	✓*	✓*	✓	✓*	✓	✓	✗	✗	k
BBB-Voting	✓	✓	✓	✓	✓	✓	✓	✓	✓	k

Tab. III: A comparison of various remote voting protocols. *Assuming a trusted VA.

(2) Voting Systems Using Cryptocurrency. Zhao and Chan [58] propose a privacy-preserving voting system with 1-out-2 choices based on Bitcoin, which uses a lottery-based approach with an off-chain distribution of voters' secret random numbers with their ZKPs. The authors use deposits incentivizing Ps to comply with the protocol; however, a malicious P can sabotage the voting by refusing to vote or vote in a wrong order. Tarasov and Tewari [59] proposed a voting system based on Zcash. The privacy of vote is ensured by the z-address that preserves unlinkability. The correctness of the voting is guaranteed by the trusted VA and the candidates. If VA is compromised, double-voting or tracing the source of the ballot (violating privacy) is possible. Liu and Wang [60] propose a conceptual voting approach based on blind signatures with 2 vote choices. Blockchain is utilized only for (auditable) sending of the messages among parties. However, despite using blind signatures, Ps send their vote to blockchain in plain-text.

IX. Conclusion

In this paper, we proposed BBB-Voting, a 1-out-of-k blockchain-based boardroom voting solution that supports fault tolerance. We made two variants of full implementation on EVM: one based on ECC DLP and the other one based on DLP for integer modulo p. We showed that only ECC variant is feasible in the real settings of public blockchains. We performed several cost optimizations and discussed further improvements concerning costs and scalability, where we made a proof-of-concept implementation with up to 1000 voters. Finally, we compared our solution with OVN and results indicate that BBB-Voting reduces the costs for voters and the authority by 13.5% and 0.9%, respectively.

References

[1] P. McCorry *et al.*, "A Smart Contract for Boardroom Voting with Maximum Voter Privacy," in *Financial Cryptography*. Springer, 2017, pp. 357– 375.

[2] S. Panja and B. K. Roy, "A secure end-to-end verifiable e-voting system using zero knowledge based blockchain," *IACR Cryptology ePrint Archive*, vol. 2018, p. 466, 2018.

[3] Y. Li *et al.*, "A blockchain-based self-tallying voting protocol in decentralized iot," *IEEE TDSC*, pp. 1–1, 2020.

[4] B. Yu *et al.*, "Platform-independent secure blockchain-based voting system," in *International Conference on Information Security*. Springer, 2018, pp. 369–386.

[5] M. Seifelnasr *et al.*, "Scalable open-vote network on ethereum," in *Financial Cryptography and Data Security*, ser. LNCS, vol. 12063. Springer, 2020, pp. 436–450.

[6] G. Dagher *et al.*, "BroncoVote: Secure Voting System using Ethereum's Blockchain," in *ICISSP'18*. SCITEPRESS, 2018, pp. 96–107.

[7] R. Cramer *et al.*, "A secure and optimally efficient multi-authority election scheme," in *EUROCRYPT '97, Konstanz, Germany*, vol. 1233. New York City: Springer, 1997, pp. 103–118.

[8] A. Kiayias and M. Yung, "Self-tallying elections and perfect ballot secrecy," in *Public Key Cryptography*, D. Naccache and P. Paillier, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 141–158.

[9] F. Hao *et al.*, "Anonymous voting by two-round public discussion," *IET Information Security*, vol. 4, no. 2, pp. 62–67, 2010.

[10] R. Küsters *et al.*, "Ordinos: A verifiable tally-hiding e-voting system," in *EuroS&P*. IEEE, 2020, pp. 216–235.

[11] N. Huber *et al.*, "Kryvos: Publicly tally-hiding verifiable e-voting," in *ACM SIGSAC Conference on CCS*, 2022, pp. 1443–1457.

[12] D. Khader *et al.*, "A fair and robust voting system by broadcast," in *(EVOTE 2012), July 11-14, 2012, Castle Hofen, Bregenz, Austria*. Bonn, Germany: GI, 2012, pp. 285–299.

[13] J. Groth, "Efficient maximal privacy in boardroom voting and anonymous broadcast," in *Financial Cryptography*. Springer, 2004, pp. 90–104.

[14] J. Benaloh *et al.*, "End-to-end verifiability," *arXiv preprint arXiv:1504.03778*, 2015.

[15] H. Jonker *et al.*, "Privacy and verifiability in voting systems: Methods, developments and trends," *Computer Science Review*, vol. 10, pp. 1–30, 2013.

[16] S. Park *et al.*, "Going from bad to worse: from internet voting to blockchain voting," *Journal of Cybersecurity*, vol. 7, no. 1, 2021.

[17] W3C Working Group, "Verifiable credentials use cases," [Online], 2019.

[18] N. Koblitz, *A course in number theory and cryptography, 2nd Edition*, ser. Graduate texts in mathematics. Springer, 1994, vol. 114.

[19] entethalliance.org, "Enterprise Ethereum Alliance," [Online], 2017.

[20] Witnet Team, "elliptic-curve-solidity," [Online], 2019.

[21] Zerocoin Team, "Big number library for solidity," [Online], 2017.

[22] F. Boudot *et al.*, "Discrete Logarithms in GF(p) – 795 bits," [Online], 2019.

[23] D. Hankerson *et al.*, "Guide to elliptic curve cryptography," *Computing Reviews*, vol. 46, no. 1, p. 13, 2005.

[24] I. Stančíková and I. Homoliak, "SBvote: Scalable Self-Tallying Blockchain-Based Voting," in *ACM/SIGAPP SAC*, 2023.

[25] R. Cheng *et al.*, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *EuroS&P*. IEEE, 2019, pp. 185–200.

[26] J. Lind *et al.*, "Teechain: a secure payment network with asynchronous blockchain access," in *ACM SOPS*, 2019, pp. 63–79.

[27] I. Homoliak and P. Szalachowski, "Aquareum: A centralized ledger enhanced with blockchain and trusted computing," *arXiv preprint arXiv:2005.13339*, 2020.

[28] Ethereum Foundation, "PLASMA CHAINS," [Online], 2022.

[29] Nathan Reiff, "What Is Polygon (MATIC)? Definition, Strengths, and Weaknesses," [Online], 2022.

[30] M. M. Chakravarty *et al.*, "Hydra: Fast isomorphic state channels," *Cryptology ePrint Archive*, 2020.

[31] T. E. Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE TIT*, vol. 31, no. 4, pp. 469–472, 1985.

[32] S. Venugopalan *et al.*, "Always on voting: A framework for repetitive voting on the blockchain," *IEEE TETC*, 2023.

[33] E. A. Brewer, "Towards robust distributed systems," in *PODC*, vol. 7, no. 10.1145. Portland, OR, 2000, pp. 343 477–343 502.

[34] Y. Gilad *et al.*, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *SOSP*. ACM, 2017.

[35] Hyperledger Team, "Hyperledger architecture, volume 1: Consensus," [Online], 2017.

[36] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009.

[37] G. Wood, "Ethereum: A secure decentralized generalized transaction ledger," [Online], 2014.

[38] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," *Communications of the ACM*, vol. 61, no. 7, 2018.

[39] C. Killer *et al.*, "Provotum: A blockchain-based and end-to-end verifiable remote electronic voting system," in *IEEE LCN*, 2020, pp. 172–183.

[40] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 24, no. 2, pp. 84–90, Feb. 1981.

[41] J. D. Cohen and M. J. Fischer, "A robust and verifiable cryptographically secure election scheme," in *SFCS '85*. IEEE, 1985, pp. 372–382.

[42] F. Zagórski *et al.*, "Remotegrity: Design and use of an end-to-end verifiable remote voting system," in *ACNS*. Springer, 2013, pp. 441–457.

[43] D. Chaum *et al.*, "Scantegrity: End-to-end voter-verifiable optical-scan voting," *IEEE Security & Privacy*, vol. 6, no. 3, pp. 40–46, 2008.

[44] J. Benaloh, "Improving privacy in cryptographic elections (technical report)," *Tech. Rep.*, 1986.

[45] A. Hevia and M. Kiwi, "Electronic jury voting protocols," in *LATIN '02*. Springer, 2002, pp. 415–429.

[46] A. Kiayias *et al.*, "An internet voting system supporting user privacy," in *ACSAC*. IEEE, Dec 2006, pp. 165–174.

[47] B. Adida, "Helios: Web-based open-audit voting," in *USENIX Security*. USENIX Assoc., 2008, pp. 335–348.

[48] Follow My Vote, Inc., "Blockchain Voting as the End-to-End Process," [Online], 2018.

[49] Tivi, "Tivi voting," [Online], 2017.

[50] D. Sandler *et al.*, "Votebox: A tamper-evident, verifiable electronic voting system," in *USENIX Security*. USENIX Assoc., 2008, pp. 349–364.

[51] Bell *et al.*, "Star-vote: A secure, transparent, auditable, and reliable voting system," in *EVT/WOTE*. USENIX Assoc., 2013, pp. 18–37.

[52] S. F. Shahandashti and F. Hao, "Dre-ip: A verifiable e-voting scheme without tallying authorities," in *ESORICS*. Berlin, Heidelberg: Springer-Verlag, 2016, pp. 223–240.

[53] M. Soud *et al.*, "Trustvote: on elections we trust with distributed ledgers and smart contracts," in *IEEE BRAINS*. IEEE, 2020, pp. 176–183.

[54] F. H. Hjálmarsson *et al.*, "Blockchain-based e-voting system," in *IEEE CLOUD*. IEEE, 2018, pp. 983–986.

[55] Agora, "Bringing voting systems into the digital age," https://www.agora.vote/, 2018.

[56] R. Matile *et al.*, "CaIV: Cast-as-Intended Verifiability in Blockchain-based Voting," in *ICBC 2019, Seoul, Korea (South), May 14-17, 2019*. Washington, DC: IEEE, 2019, pp. 24–28.

[57] K. Košt'ál *et al.*, "Blockchain e-voting done right: Privacy and transparency with public blockchain," in *ICSESS*. IEEE, 2019, pp. 592–595.

[58] Z. Zhao and T.-H. H. Chan, "How to vote privately using Bitcoin," in *ICICS*. Springer, 2015, pp. 82–96.

[59] P. Tarasov and H. Tewari, "Internet voting using Zcash," *Cryptology ePrint Archive*, 2017.

[60] Y. Liu and Q. Wang, "An e-voting protocol based on blockchain," *Cryptology ePrint Archive*, 2017.

[61] O. Baudron *et al.*, "Practical multi-candidate election system," in *PODC '01*. New York, NY, USA: ACM, 2001, pp. 274–283.

SBvote: Scalable Self-Tallying Blockchain-Based Voting

Ivana Stančíková
Brno University of Technology, Faculty of Information Technology, Czech Republic
istancikova@fit.vutbr.cz

Ivan Homoliak
Brno University of Technology, Faculty of Information Technology, Czech Republic
ihomoliak@fit.vutbr.cz

ABSTRACT

Decentralized electronic voting solutions represent a promising advancement in electronic voting. One of the e-voting paradigms, the self-tallying scheme, offers strong protection of the voters' privacy while making the whole voting process verifiable. Decentralized smart contract platforms became interesting practical instantiation of the immutable bulletin board that this scheme requires to preserve its properties. Existing smart contract-based approaches employing the self-tallying scheme (such as OVN or BBB-Voting) are only suitable for a boardroom voting due to their scalability limitation. The goal of our work is to build on existing solutions to achieve scalability without losing privacy guarantees and verifiability. We present SBvote, a blockchain-based self-tallying voting protocol that is scalable in the number of voters, and therefore suitable for large-scale elections. The evaluation of our proof-of-concept implementation shows that the protocol's scalability is limited only by the underlying blockchain platform. We evaluated the scalability of SBvote on two public smart contract platforms – Gnosis and Harmony. Despite the limitations imposed by the throughput of the blockchain platforms, SBvote can accommodate elections with millions of voters.

CCS CONCEPTS

• **Applied computing** → **Voting / election technologies**; • **Security and privacy** → Distributed systems security;

KEYWORDS

E-voting, blockchain, scalability, privacy, smart contracts.

ACM Reference Format:
Ivana Stančíková and Ivan Homoliak. 2023. SBvote: Scalable Self-Tallying Blockchain-Based Voting. In *The 38th ACM/SIGAPP Symposium on Applied Computing (SAC '23), March 27 – March 31, 2023, Tallinn, Estonia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3555776.3578603

1 INTRODUCTION

Voting is an essential means of achieving a collective decision. Traditionally, in large-scale voting such as national elections, the participants cast anonymous paper ballots that are later tallied by a trusted authority. With the advances in information technology, electronic voting systems have been introduced. While several small-scale or boardroom e-voting protocols with decentralized architecture have been proposed [19, 27], large-scale electronic voting systems mostly follow a centralized model [1, 8]. However, a centralized entity that is in a control of the voting process represents a single point of failure as well as a possible element for misbehavior.

The verifiability of many e-voting protocols depends on the assumed existence of a public bulletin board (PBB) that allows append-only modifications and immutability of the historical data [14]. Voting systems such as Helios [1] implement the public bulletin board as a single web server. However, this introduces a possibility of several issues, including unavailability of the server (e.g., due to a denial-of-service attack) or a censorship by the authority controlling the server.

Other systems [19, 27, 31] instantiate the public bulletin board by a blockchain with a smart contract platform. On top of the immutability and append-only features, such blockchains also provide correct execution of a code that enables decentralized e-voting schemes to utilize public verifiability of the data submitted to the bulletin board (e.g., votes and a tally). Protocols such as Open Vote Network (OVN) [19] and BBB-Voting [27] use smart contracts to orchestrate the procedures of the boardroom voting protocol. The distributed nature of these protocols also eliminates the need to rely on the authority to tally the votes. In these approaches, referred to as *self-tallying voting*, any participant can sum and verify the tally.

We base our work on BBB-Voting since it enables more than two voting choices and a recovery of faulty participants in contrast to OVN. Our goal is to build a voting protocol that resolves the scalability limitation of the self-tallying approaches while maintaining the maximum voter privacy. Therefore, we introduce SBvote, a decentralized blockchain-based e-voting protocol providing scalability in the number of participants by grouping them into voting booths instantiated as dedicated smart contracts that are controlled and verified by the aggregation smart contract. Our approach is suitable for privacy-preserving self-tallying large-scale e-voting.

***Contributions*.** We make the following contributions:

i) We introduce SBvote, a blockchain-based self-tallying e-voting protocol that enables scalability in the number of voters and is based on BBB-Voting protocol. SBvote introduces multiple voting smart contracts booths that are managed and aggregated by the main smart contract.
ii) Our extended solution maintains all properties of decentralized e-voting, including public verifiability, perfect ballot secrecy, and fault tolerance. Moreover, it improves the privacy of voters within booths.
iii) We made a proof-of-concept implementation and evaluated it on two smart contract platforms, Harmony and Gnosis. We

SAC '23, March 27 – March 31, 2023, Tallinn, Estonia

ACM ISBN 978-1-4503-9517-5/23/03…$15.00
https://doi.org/10.1145/3555776.3578603

achieved the best scalability results using the Harmony blockchain, allowing us to run elections with 1.5M voters and two candidates within a two-days interval.

Organization. The rest of this paper is organized as follows. Essential preliminaries are presented in Section 2. We introduce SBvote in Section 3 and evaluate its scalability in Section 4. We provide the security analysis of SBvote and discuss its properties in Section 5. We review the related work in Section 6 and finally conclude our paper in Section 7.

2 PRELIMINARIES

We briefly review the properties of voting protocols and provide a short description of blockchains and smart contracts. Finally, we describe the BBB-Voting protocol on which we base our work.

2.1 Voting

We provide a list of the most important "desired" properties of voting protocols in the following.

Privacy. The votes remain anonymous. Only the voter herself knows which candidate she voted for. Privacy protection is a crucial attribute of voting systems used in practice and is very important in publicly verifiable voting schemes.

Perfect Ballot Secrecy. It was defined by Kiayias and Yung [16], and it extends the privacy property. In a scheme with perfect ballot secrecy, a partial result can only be revealed if all remaining voters collude to uncover it.

Self-Tallying. This property ensures that any interested party can compute the tally once all the votes are cast.

Fault Tolerance. The protocol allows for excluding faulty participants in a publicly verifiable manner without restarting the whole voting protocol.

Verifiability. Verifiability includes *individual verifiability* (allowing the voter to verify her vote has been counted) and *universal verifiability* (allowing any interested party to verify all cast votes have been correctly tallied). Furthermore, the verifiability of a voting system can be described [3] as follows:

- ***cast-as-intended:*** a voter can verify the encrypted vote contains her choice of candidate,
- ***recorded-as-cast:*** a voter can verify the system recorded her vote correctly,
- ***tallied-as-recorded:*** any interested party is able to verify whether the final tally corresponds to the recorded votes.

A voting system that satisfies all these properties is considered end-to-end verifiable.

Dispute-Freeness. The protocol's design prevents any disputes among involved parties by allowing anyone to verify whether a participant followed the protocol.

Completeness. All valid votes are included in the final tally.

2.2 Blockchains and Smart Contracts

Blockchain is a continuously growing distributed ledger consisting of blocks maintained by a network of consensus nodes that run a consensus protocol. Once the consensus nodes agree on a new block, it is added to the blockchain. The blocks are cryptographically linked to ensure the immutability of the entire ledger and typically contain records of cryptocurrency transfers executed within the network. Orders to execute transfers are communicated to the network in messages called *transactions.* A block may also contain application code written in a supported language of a blockchain equipped with a smart contract platform. This code is invoked by a transaction containing execution orders (i.e., function calls of a smart contract). The blockchain network then acts as a decentralized computation platform – the blockchain nodes execute the smart contract code.

Smart contract platforms (such as Ethereum [30]) measure the execution complexity of smart contracts in units of gas. The sender of a transaction containing a smart contract invocation has to pay for the consumed gas to cover the expenditures of the consensus nodes executing the computation. The gas price is volatile and based on the demand on the network.

2.3 BBB-Voting

BBB-Voting [27] is a system for boardroom voting supporting $k \geq 2$ voting choices. The basic protocol used in BBB-Voting consists of five phases (i.e., registration, setup, pre-voting, voting, tally) and an optional fault-recovery phase.

In the registration phase, a voting authority registers eligible voters and their wallet addresses to the voting smart contract. In the setup phase, the cryptographic parameters of the voting are agreed upon by all participants. Each voter then creates her ephemeral private/public key pair and submits her ephemeral public key to the smart contract. The multi-party computation (MPC) key for each voter is computed by the smart contract in the pre-voting phase. In the voting phase, each voter computes her blinding key (consisting of ephemeral private key and MPC key), uses the blinding key to encrypt a vote to a selected candidate and then submits the blinded vote to the smart contract. A 1-out-of-k non-interactive zero-knowledge (NIZK) proof of set membership is submitted to the smart contract along with the blinded vote. Next, the smart contract verifies the correctness of such a vote. During the tally phase of the protocol, the tally of votes is computed off-chain by an arbitrary party and submitted to the smart contract. The smart contract verifies that the tally was computed correctly.

The protocol also includes an optional fault-recovery extension that can be placed after the voting phase. This phase is useful if some participants have stalled and have not cast their blinded votes.

The BBB-Voting scheme provides perfect ballot secrecy, fairness, public verifiability, self-tallying feature, dispute-freeness, resistance to serious failures, and maximizes the voters' privacy (see Section 2.1). Also, it introduces several optimizations of the implementation to decrease the costs of the protocol and accommodate a larger number of participants than the previous approaches (i.e., OVN [19]). BBB-Voting is designed as a single smart contract deployed on the Ethereum blockchain. Nevertheless, BBB-Voting is intended only for boardroom voting with a low number of involved participants. Hence it does not provide scalability as might be required in national elections. Another limitation of BBB-Voting is the low number of stalling participants the system can recover from in a single fault recovery round due to the block gas limit.

3 SCALABLE VOTING PROTOCOL

In this section, we propose SBvote, a scalable e-voting protocol that is based on BBB-Voting (see Section 2.3).

3.1 System Model

We focus on a decentralized e-voting that provides all desired properties of e-voting schemes mentioned in Section 2.1 as well as scalability in the number of the participants. We assume a centralized authority that is responsible for the enrollment of the participants and shifting the stages of the protocol. However, the authority can neither change nor censor the votes of the participants, and it cannot compromise the privacy of the votes.

We assume that a public bulletin board required for e-voting is instantiated by a blockchain platform that moreover supports the execution of smart contracts. We assume that all participants of voting have their thin clients that can verify the inclusion of their transactions in the blockchain as well as the correct execution of the smart contract code.

Adversary Model. We consider an adversary that passively listens to a communication on the blockchain network. The adversary cannot modify or replace any honest transactions since she does not hold the private keys of the participants. Next, we assume that the adversary cannot block an honest transaction due to the censorship-resistance property of the blockchain. The adversary can link a voter's IP address to her blockchain address. However, she does not possess the computational resources to break the cryptographic primitives used in the blockchain platform and the voting protocol. The adversary cannot access or compromise the voter's device or the user interface of the voting application. We assume that in each voting group of n participants, at most t of them can be controlled by the adversary and disobey the voting protocol, where $t \leq n - 2$ and $n \geq 3$. This eliminates the possibility of *full collusion* against a single voter [13].

3.2 Proposed Approach

Involved Parties. Our proposed approach has the following actors and components: (1) *a participant* $\mathbb{P}$ (*a voter*) who chooses a candidate (i.e., a voting choice) and casts a vote, (2) *a voting authority* $\mathbb{VA}$ responsible for the registration of participants and initiating actions performed by smart contracts, (3) *a booth contract* $\mathbb{BC}$, which is replicated into multiple instances, where each instance serves a limited number of participants. New instances might be added on-demand to provide scalability. (4) *The main contract* $\mathbb{MC}$, which assigns participants to voting booths, deploys booth contracts, and aggregates the final tally from booth contracts.

Protocol. We depict our protocol in Figure 1. SBvote follows similar phases as BBB-Voting but with several alterations that enable better scalability. The registration phase requires $\mathbb{VA}$ to authenticate users and generate a list of eligible voters. In BBB-Voting, the setup phase of the protocol allows users to submit their ephemeral public keys. However, in contrast to BBB-Voting, SBvote requires additional steps to set up the booth contracts. First, eligible voters are assigned to voting groups and then $\mathbb{BC}$ is deployed for each voting group. Once the setup is finished, voters proceed to submit their ephemeral public keys during a sign-up phase. These keys are further used to compute multi-party computation (MPC) keys within each voting group during a pre-voting phase. In the voting phase, voters cast their blinded votes along with corresponding NIZK proofs. The NIZK proof allows $\mathbb{BC}$ to verify that a blinded vote correctly encrypts one of the valid candidates. If some of the voters who submitted their ephemeral public keys have failed to cast their vote, the remaining active voters repair their votes in the subsequent fault recovery phase. This is achieved by removing the key material of stalling voters from the encryption of the correctly cast votes. The key material has to be provided by each active voter along with NIZK proof of correctness. After the repair of votes, the tallies for individual voting groups are computed during the tally phase of a booth. Then, partial tally results are aggregated to obtain the final tally by $\mathbb{MC}$.

In the following, we describe the phases of our protocol in more detail. Phases 2–6 are executed independently (and thus in parallel) within each of the voting groups/booth contracts.

Registration. In this phase, the participants interact with $\mathbb{VA}$ to register as eligible voters. A suitable identity management (IDM) system is required, allowing $\mathbb{VA}$ to verify participants' identities and eligibility to vote.[1] Each participant creates her blockchain wallet address and registers it with $\mathbb{VA}$ that stores a mapping between a participant's identity and her wallet address.

Phase 1 (Setup). First, $\mathbb{VA}$ deploys $\mathbb{MC}$ to the blockchain. Then, $\mathbb{VA}$ enrolls the wallet addresses of all registered participants to $\mathbb{MC}$ within a transaction.[2] Once all the registered participants have been enrolled, $\mathbb{VA}$ triggers $\mathbb{MC}$ to pseudo-randomly distribute enrolled participants into groups whose size is pre-determined and ensures a certain degree of privacy. Note that distributed randomness protocols such as RoundHound [25] might be used for this purpose, however, in this work we assume a trusted randomness source that is agreed upon by all voters (e.g., a hash of a Bitcoin block).

In every group, the participants agree on the parameters of the voting. Let n be the number of participants in the group and k the number of candidates. We specify the parameters of voting as follows:

1) a common generator $g \in \mathbb{F}_p^*$, where $p = 2 \cdot q + 1$, q is a prime and $n < p - 1$.
2) k independent generators $\{f_1, ..., f_k\}$ in $\mathbb{F}_p^*$ such that $f_i = g^{2^{(i-1)m}}$, where m is the smallest integer such that $2^m > n$.

Then, $\mathbb{VA}$ deploys a booth contract $\mathbb{BC}$ for each group of participants with these previously agreed upon voting parameters. $\mathbb{MC}$ stores a mapping between a participant's wallet address and the group she was assigned to.

Phase 2 (Sign-Up). Eligible voters enrolled in the setup phase review the candidates and the voting parameters. Each voter who intends to participate obtains the address of $\mathbb{BC}$ she was assigned to by $\mathbb{MC}$. From this point onward, each participant interacts only with her $\mathbb{BC}$ representing the group she is part of. Every participant P_i creates her ephemeral key pair consisting of a private key $x_i \in_R \mathbb{F}_p^*$ and public key g^{x_i}. The P_i then sends her public key to $\mathbb{BC}$. By

[1] The details of IDM are out-of-scope for this work.

[2] Note that in practice this step utilizes transaction batching to cope with the limits of the blockchain platform (see Section 3.3).

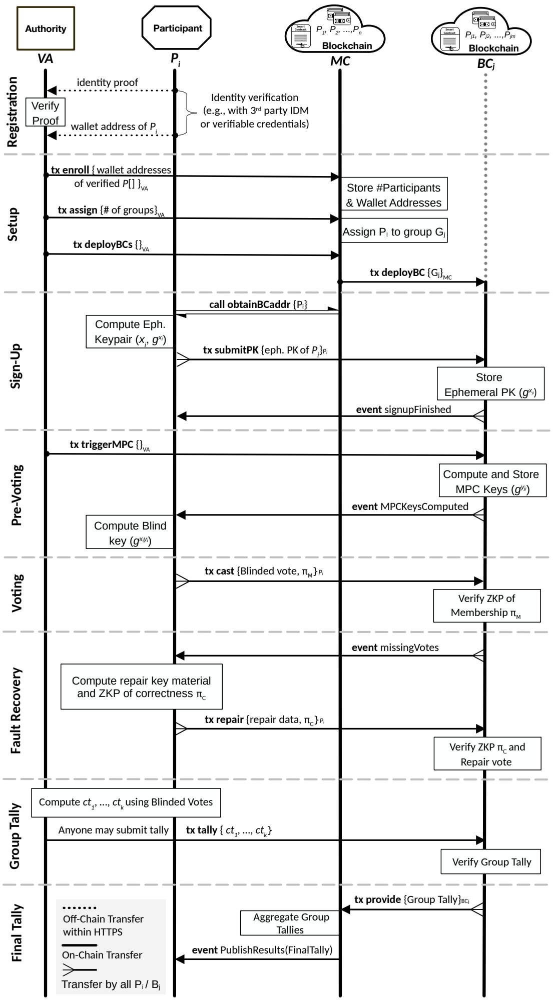

Authority
VA
Participant
P_i
$P_1, P_2, \ldots, P_n$
Blockchain
MC
$P_{j1}, P_{j2}, \ldots, P_{jm}$
Blockchain
BC_j
Registration
identity proof
Verify Proof
wallet address of P_i
Identity verification (e.g., with 3rd party IDM or verifiable credentials)
Setup
tx enroll { wallet addresses of verified P[] }VA
Store #Participants & Wallet Addresses
tx assign {# of groups}VA
Assign P_i to group G_j
tx deployBCs {}VA
tx deployBC {G_j}MC
Sign-Up
call obtainBCaddr {P_i}
Compute Eph. Keypair (x_i, g^{x_i})
tx submitPK {eph. PK of P_i}P_i
Store Ephemeral PK ($g^{x_{ji}}$)
event signupFinished
Pre-Voting
tx triggerMPC {}VA
Compute and Store MPC Keys ($g^{y_{ji}}$)
event MPCKeysComputed
Compute Blind key ($g^{x_i y_i}$)
Voting
tx cast {Blinded vote, π_M}P_i
Verify ZKP of Membership π_M
Fault Recovery
event missingVotes
Compute repair key material and ZKP of correctness π_C
tx repair {repair data, π_C}P_i
Verify ZKP π_C and Repair vote
Group Tally
Compute $ct_1, \ldots, ct_k$ using Blinded Votes
Anyone may submit tally
tx tally { $ct_1, \ldots, ct_k$}
Verify Group Tally
Final Tally
tx provide {Group Tally}BC_j
Aggregate Group Tallies
event PublishResults(FinalTally)
Off-Chain Transfer within HTTPS
On-Chain Transfer
Transfer by all P_i / B_j

Figure 1: Overview of SBvote protocol.

submitting an ephemeral public key, the participant commits to cast a vote later. Furthermore, participants are required to send a deposit within this transaction. If the voter does not cast her vote or later does not participate in the potential fault recovery phase, she will be penalized by losing the deposit. Voters who participate correctly retrieve their deposit at the end of the voting.

Phase 3 (Pre-Voting). In this step, each $\mathbb{BC}$ computes synchronized multi-party computation (MPC) keys from the participants' ephemeral public keys submitted in the previous step. To achieve scalability, the MPC keys are computed independently in each $\mathbb{BC}$ over the set of ephemeral public keys within the group. The MPC key for participant P_i is computed as follows:

$$g^{y_i} = \prod_{j=1}^{i-1} g^{x_j} / \prod_{j=i+1}^{n} g^{x_j}, \qquad (1)$$

Participant P_i	Booth Contract
$(h \leftarrow g^{y_i},\ v_i)$	$(h \leftarrow g^{y_i})$

$Select\ v_i \in \{1, \ldots, k\},$
$Use\ cand.\ generators$
$f_l \in \{f_1, \ldots, f_k\} \subseteq \mathbb{F}_p^*,$
$Publish\ x \leftarrow g^{x_i}$
$Publish\ B_i \leftarrow h^{x_i} f_l$
$w \in_R \mathbb{F}_p^*$
$\forall l \in \{1, .., k\} \setminus v_i :$
1. $r_l, d_l \in_R \mathbb{F}_p^*$
2. $a_l \leftarrow x^{-d_l} g^{r_l}$
3. $b_l \leftarrow h^{r_l} (\frac{B_i}{f_l})^{-d_l}$

$for\ \ v_i :$
1. $a_{v_i} \leftarrow g^w$
2. $b_{v_i} \leftarrow h^w$

$c \leftarrow H(\{\{a_l\}, \{b_l\}\}_l)$

$for\ \ v_i :$
1. $d_{v_i} \leftarrow \sum_{l \neq v_i} d_l$
2. $d_{v_i} \leftarrow c - d_{v_i}$
3. $r_{v_i} \leftarrow w + x_i d_{v_i}$
4. $q \leftarrow p - 1$
5. $r_{v_i} \leftarrow r_{v_i} \mod q$

$$\xrightarrow{\forall l : \{a_l\}, \{b_l\}, \{r_l\}, \{d_l\}}$$

$\Psi \leftarrow \{\forall l : \{a_l\}, \{b_l\}\}$
$c \leftarrow H(\Psi)$
$\sum_l d_l \overset{?}{=} c$
$\forall l \in \{1, .., k\}$
1. $g^{r_l} \overset{?}{=} a_l x^{d_l}$
2. $h^{r_l} \overset{?}{=} b_l (\frac{B_i}{f_l})^{d_l}$

Figure 2: Non-interactive zero-knowledge proof for 1-out-of-k set membership.

where $y_i = \sum_{j<i} x_j - \sum_{j>i} x_j$ and $\sum_i x_i y_i = 0$ (see Hao et al. [13] for the proof). The computation of MPC keys is triggered by $\mathbb{VA}$ in each $\mathbb{BC}$. After the computation, each participant obtains her MPC key from $\mathbb{BC}$ and proceeds to compute her ephemeral blinding key as $g^{x_i y_i}$ using her private key x_i.

Phase 4 (Voting). Before participating in this phase of the protocol, each voter must create her blinded vote and a NIZK proof of its correctness. The blinded vote of the participant P_i is $B_i = g^{x_i y_i} f_j$, where $f_j \in f_1, \ldots, f_k$ represents her choice of a candidate. The participant casts the blinded vote by sending it to $\mathbb{BC}$ in a transaction $cast(B_i, \pi_M)$, where π_M is a 1-out-of-k NIZK proof of set membership. This proof allows $\mathbb{BC}$ to verify that the vote contains one of the candidate generators from $f_1, \ldots, f_k$ without revealing the voter's choice. $\mathbb{BC}$ performs a check of the proof's correctness and accepts well-formed votes. Construction and verification of the NIZK proof are depicted in Figure 2.

$$\begin{array}{lcl}
\underline{\textit{Participant } P_i} & & \underline{\text{Smart Contract}} \\
(A \leftarrow g^{x_i}, B \leftarrow g^{x_j}, x_i) & & (A \leftarrow g^{x_i},\ B \leftarrow g^{x_j}) \\
 & & \\
\textit{Let } w_i, \in_R \mathbb{F}_p & & \\
C \leftarrow g^{x_i x_j} & & \\
m_1 \leftarrow g^{w_i} & & \\
m_2 \leftarrow B^{w_i} & & \\
c \leftarrow H(A, B, m_1, m_2) & & \\
r_i \leftarrow w_i + c x_i & & \\
 & \xrightarrow{C, r_i, m_1, m_2} & c \leftarrow H(A, B, m_1, m_2) \\
 & & g^r \overset{?}{=} m_1 A^c \\
 & & B^r \overset{?}{=} m_2 C^c
\end{array}$$

Figure 3: NIZK proof verifying correspondence of $g^{x_i x_j}$ to public keys $A = g^{x_i}, B = g^{x_j}$.

Phase 5 (Fault-Recovery). The use of synchronized MPC keys ensures that a vote cast by each voter contains the key material shared with all voters within the group. If some of the voters within a group stall during the voting phase, the tally cannot be computed from the remaining data. Therefore, we include a fault-recovery phase, where remaining voters provide $\mathbb{BC}$ with the key material they share with each stalling voter, enabling $\mathbb{BC}$ to repair their votes. In detail, for a stalling voter P_j and an active voter P_i ($i \neq j$), the shared key material $g^{x_i x_j}$ consists of the stalling voter's ephemeral public key g^{x_j} (previously published in $\mathbb{BC}$) and the active voter's ephemeral private key x_i. The active voters send the shared key material to $\mathbb{BC}$ along with a NIZK proof depicted in Figure 3. The NIZK proof allows $\mathbb{BC}$ to verify that the shared key material provided by the voter corresponds to the ephemeral public keys g^{x_i} and g^{x_j}.

Suppose some of the previously active voters become inactive during the fault-recovery phase (i.e., do not provide the shared key material needed to repair their votes). In that case, the fault-recovery phase can be repeated to exclude these voters. Note that this phase takes place in groups where all the voters who committed to vote during the sign-up phase have cast their votes.

Phase 6 (Booth Tallies). At first, the tally has to be computed for each group separately. Computation of the result is not performed by $\mathbb{BC}$ itself. Instead, $\mathbb{VA}$ (or any participant) obtains the blinded votes from $\mathbb{BC}$, computes the tally, and then sends the result back to $\mathbb{BC}$, which verifies whether a provided tally fits

$$\prod_{i=1}^{n} B_i = \prod_{i=1}^{n} g^{x_i y_i} f_j = g^{\sum_i x_i y_i} f_j = {f_1}^{ct_1} {f_2}^{ct_2} \ldots {f_k}^{ct_k}, \tag{2}$$

where $ct_j \in ct_1, \ldots, ct_k$ denotes the vote count for each candidate.

Phase 7 (Final Tally). Once $\mathbb{BC}$ obtains a correctly computed tally, it sends it to $\mathbb{MC}$. $\mathbb{MC}$ collects and summarizes the partial tallies from individual booths and announces the final tally once all booths have provided their results. The participants can also review the partial results from already processed booths without waiting for the final tally since the booth tallies are processed independently.

3.3 Design Choices and Optimizations

We introduce several specific features of SBvote, which allow us to achieve the scalability and privacy properties.

Storage of Voters' Addresses. If we were to store the voters' wallet addresses in the booth contracts, it would cause high storage overhead and thus high costs. However, we proposed to store these addresses only in $\mathbb{MC}$, while booth contracts can only query $\mathbb{MC}$ whenever they require these addresses (i.e., when they verify whether a voter belongs to the booth's group). As a result, this eliminates the costs of transactions when deploying booth contracts, and moreover saving the blockchain storage space.

Elimination of Bottlenecks. The main focus of our proposed approach is to eliminate the bottlenecks that limit the number of voters and thus the size of the voting groups. In particular, passing the necessary data within a single transaction could potentially exceed the block gas limit.

The scalability of the Setup phase of SBvote is straightforward to resolve since it does not involve any transient integrity violation checks (excluding duplicity checks). In all these cases, $\mathbb{VA}$ splits the data into multiple independent transactions. Similarly, each active voter can send the key material required to repair her vote in several batches in the Fault-Recovery phase, allowing the system to recover from an arbitrary number of stalling participants.

In contrast to the Setup and Fault-Recovery phases, batching in the Pre-Voting phase is not trivial since it requires transient preservation of integrity between consecutive batches of the particular voting group. Therefore, we designed a custom batching mechanism, which eliminates this bottleneck while also optimizing the cost of the MPC computation.

MPC Batching and Optimization. If computed independently for each participant, the computation of MPC keys leads to a high number of overlapping multiplications. Therefore, we optimize this step by dividing the computation into two parts, respecting both

Algorithm 1 Pre-computation of right side values from Equation 1.

Inputs:

- n: # of voters
- mpc_batch : batch size for MPC computation
- $voterPKs$: array of voters' ephemeral public keys

Outputs:

- $right\_markers$: pre-computed right side values

```
$right\_tmp \leftarrow 0$
if $n$ mod $mpc\_batch \neq 0$ then
    $right\_markers$.push($right\_tmp$)
end if
for $i \leftarrow 0$ to $n$ do
    if $n$ mod $mpc\_batch$ = $(i - 1)$ mod $mpc\_batch$ then
        $right\_markers$.push($right\_tmp$)
    end if
    $right\_tmp \leftarrow right\_tmp * voterPKs[n - i]$
end for
```

Algorithm 2 Computation of a batch of MPC keys.

Inputs:

- *voterPKs* : array of voters' ephemeral public keys
- *mpc_batch* : batch size for MPC computation
- *start*, *end* : start and end index of the current batch
- *right_marker* : pre-computed right side value for the first index in a batch
- *act_left* : left side value from the previous batch

Outputs:

- *act_left* : left side value at the last index of the current batch
- *mpc_keys* : array of MPC keys for the current batch

```
Compute right side values for the batch:
1: $right\_tab[mpc\_batch - 1] \leftarrow right\_marker$
2: for $i \leftarrow 0$ to $mpc\_batch$ do
3:     $j \leftarrow mpc\_batch - i$
4:     $right\_tab[j - 1] \leftarrow right\_tab[j] * voterPKs[i - 1]$
5: end for
Compute the current batch of MPC keys:
6: for $i \leftarrow start$ to $end$ do
7:     $act\_left \leftarrow act\_left * voterPKs[i - 1]$
8: end for
9: $mpc\_keys[i] \leftarrow act\_left \div right\_tab[i]$ mod $mpc\_batch$
```

sides of the expression in Equation 1 and reusing accumulated values for each side.

First, we pre-compute the right part (i.e., divisor) of Equation 1, which consists of a product of ephemeral public keys of voters with a higher index than the current voter's one (i.e., i in Equation 1). The product is accumulated and saved in the contract's storage at regular intervals during a single iteration over all ephemeral public keys. The size of these intervals corresponds to the batch size chosen for the computation of the remaining (left side) of the equation. We refer to these saved values as *right markers* (see Algorithm 1). We only choose to save the right markers in the storage of $\mathbb{BC}$ instead of saving all accumulated values due to the high cost of storing data in the smart contract storage. Though the intermediate values between right markers have to be computed again later, they are only kept in memory (not persistent between consecutive function calls). Therefore, they do not significantly impact the cost of the computation.

The second part of the computation is processed in batches. First, the right-side values for all voters within the current batch are obtained using the pre-computed right marker corresponding to this batch (see lines 1–5 of Algorithm 2). Then, the left part of Equation 1 is computed for each voter within the batch, followed by evaluating the entire equation to obtain the MPC key (lines 6–9 of Algorithm 2). This left-side value is not discarded; therefore, computing the left side for the next voter's MPC key only requires single multiplication. The last dividend value in the current batch is saved in the contract's storage to allow its reuse for the next batch.

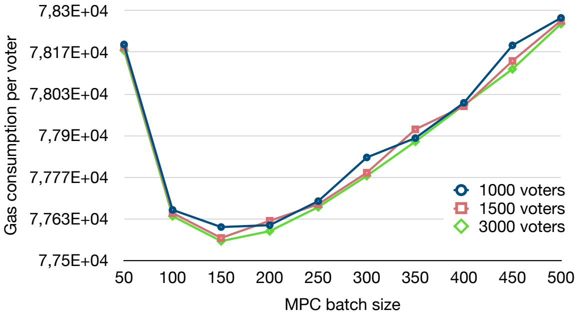


Figure 4: Per voter cost of the MPC key computation w.r.t. the batch size.

4 EVALUATION

To evaluate the scalability of SBvote, we created the proof of concept implementation that builds on BBB-Voting [27]. We used the Truffle framework and Solidity programming language to implement the smart contract part and Javascript for the client API of all other components. We also utilized the Witnet library [29] for on-chain elliptic curve operations on the standardized *Secp256k1* curve [23]. Although Solidity was primarily intended for Ethereum and its Ethereum Virtual Machine (EVM), we have not selected Ethereum for our evaluation due to its high operational costs and low transactional throughput, which is contrary to our goal of improving scalability. However, there are many other smart contract platforms supporting Solidity and EVM, out of which we selected Gnosis[3] and Harmony[4] due to their low costs and high throughput.

Throughout our evaluation, we considered the following parameters of the chosen platforms: 30M block gas limit with 5 second block creation time on Gnosis and 80M block gas limit with 2 second block creation time on Harmony.

MPC Batch Size. The MPC keys in the Pre-Voting phase are computed in batches (see Section 3.3). In detail, there is a pre-computed value available for the first voter in each batch. Using a small batch size imposes many transactions and high execution costs due to utilizing fewer pre-computed values. In contrast, using a large batch size requires more expensive pre-computation and storage allocation, which results in a trade-off. This trade-off is illustrated in Figure 4, depicting how the batch size affects the cost of the computation per voter. We can see that the best value for our setup is 150 voters per batch.

The Number of Candidates. The number of candidates our voting system can accommodate remains limited. This is mainly caused by the block gas limit of a particular platform. In detail, we can only run voting with a candidate set small enough so that the vote-casting transaction does not exceed the underlying platform's block gas limit. Such transaction must be accompanied by a NIZK proof of set membership (i.e., proof that the voter's encrypted choice belongs to the set of candidates), and the size of the candidate set determines its execution complexity. Figure 5 illustrates this

[3] https://developers.gnosischain.com, *Accessed: 2022-09-26.*
[4] https://www.harmony.one, *Accessed: 2022-09-26.*

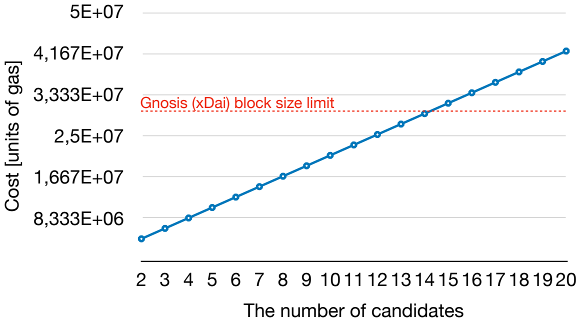


Figure 5: The cost of vote casting w.r.t. the number of candidates.

dependency. Our experiments show that the proposed system can accommodate up to 38 and 14 candidates on Harmony and Gnosis, respectively.

The Total Number of Participants. The time period over which the voters can cast their ballots typically lasts only several days in realistic elections. The platform's throughput over a restricted time period and the high cost of the vote-casting transactions result in a trade-off between the number of voters and the number of candidates. We evaluated the limitations of the proposed voting protocol on both Harmony and Gnosis, as shown in Figure 6 and Figure 7. Note that in these examples, we considered only the most expensive phase of the protocol (i.e., voting phase) to be time-restricted.

We determined that with two candidates, the proposed system can accommodate ~1.5M voters over a 2-day voting period and up to 3.8M voters over a 5-day voting period on the Harmony blockchain. On the other side of the trade-off, with the maximum number of 38 candidates on Harmony, maximally 216K voters can participate within a 5-day voting period.

5 SECURITY ANALYSIS AND DISCUSSION

We discuss the properties and scenarios affecting the security and privacy of SBvote.

Privacy. Within each voting group, SBvote maintains perfect ballot secrecy. The adversary, as defined in Section 3.1, cannot reveal a participant's vote through a collusion of all remaining participants since adversary can control at most $n - 2$ participants. The privacy of votes can be violated only if all participants in a voting group vote for the same candidate. However, this is a natural property of voting protocols, which output the tally rather than only the winning candidate. SBvote mitigates this problem by implementing transaction batching, which allows the authority to maintain a sufficiently large size of the voting groups to lower the probability of a unanimous vote within the groups. This probability is further decreased in SBvote by the smart-contract-based pseudo-random assignment of participants to the groups. We refer the reader to the work of Ullrich [26] that addresses the issue of unanimous voting and the probability of its occurrence.

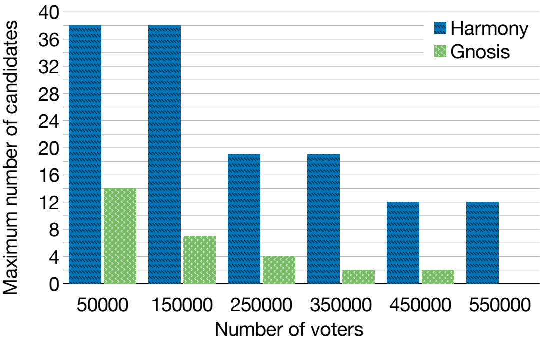


Figure 6: The maximum number of candidates our approach can process during a fixed 5-day voting interval, assuming various numbers of voting participants.

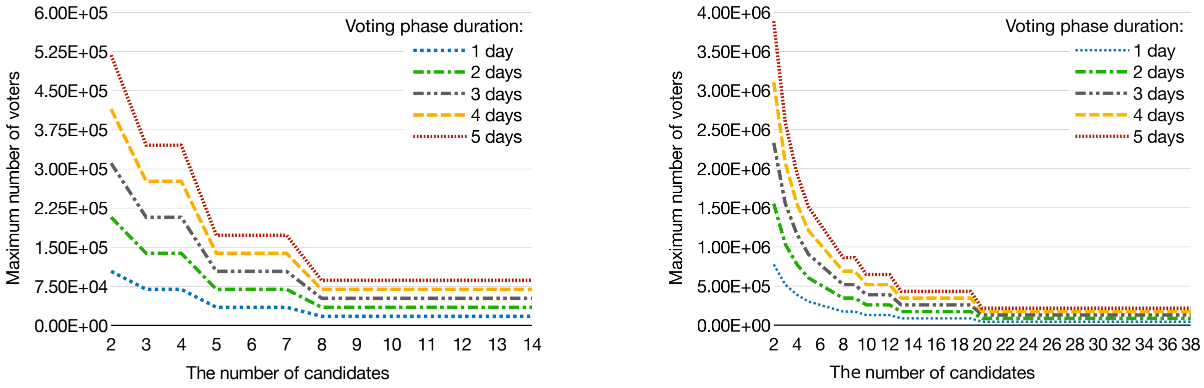
Deanonymization & Linking Addresses. In common blockchains, the network-level adversary might be able to link the participant's address with her IP address. Such an adversary can also intercept the participant's blinded vote; however, she cannot extract the vote choice due to the privacy-preserving feature of our voting protocol. Therefore, even if the adversary were to link the IP address to the participant's identity, the only information she could obtain is whether the participant has voted. Nevertheless, to prevent the linking of addresses, participants can use VPNs or anonymization services such as Tor.

Re-Voting. It is important to ensure that no re-voting is possible, which is to avoid any inference about the final vote of a participant in the case she would reuse her ephemeral blinding key to change her vote during the voting stage. Such a re-voting logic can be easily enforced by the smart contract, while the user interface of the participant should also not allow re-voting. Also, note that ephemeral keys are one-time keys and thus are intended to use only within one instance of e-voting protocol to ensure the security and privacy of the protocol. If a participant were to vote in a different instance of e-voting, she would generate new ephemeral keys.

Forks in Blockchain. Blockchains do not guarantee immediate immutability due to possible forks. This differentiates blockchains from public bulletin boards, as defined in [16]. However, since our protocol does not contain any two-phase commitment scheme with revealed secrets, its security is not impacted by accidental or malicious forks. Temporary forks also do not impact the voting stage since the same votes can be resubmitted by client interfaces.

Self-Tallying Property. The self-tallying property holds within each voting group since the correctness of obtained tallies can be verified by anybody. Consequently, this property holds for the whole voting protocol since the main contract aggregates the booth tallies of the groups in a verifiable fashion.

Verifiability. SBvote achieves both individual and universal verifiability. By querying the booth contract, each voter can verify her vote has been recorded. Each voter (and any interested party) can verify the booth tally since it satisfies the self-tallying property, i.e., the Equation 2 would not hold should any vote be left out. Any



(a) Gnosis.

(b) Harmony.

Figure 7: The maximum number of voters that our approach can accommodate w.r.t. the number of candidates.

party can verify the final tally aggregated in the main contact by querying all the booth contracts to obtain individual booth tallies.

Platform-Dependent Limitations. Although our system itself does not limit the number of participants, the required transactions are computationally intensive, which results in high gas consumption. Therefore, large-scale voting using our system might be too demanding on the underlying smart contract platform. As a potential solution, public permissioned blockchains dedicated to e-voting might be utilized.

Adversary Controlling Multiple Participants in the Fault Recovery. One issue that needs to be addressed in the fault recovery is the adversary controlling multiple participants and letting them stall one by one in each fault recovery round. Even though the fault recovery mechanism will eventually finish with no new stalling participants, such behavior might increase the costs paid by remaining participants who are required to submit counter-party shares in each round of the protocol. For this reason, similar to the voting stage, we require the fault recovery stage to penalize stalling participants by losing the deposit they put into the smart contract at the beginning of our protocol. On the other hand, the adversary can cause a delay in the voting protocol within a particular booth. However, it does not impact other booths. To further disincentivize the adversary from such a behavior, the fault-recovery might require additional deposits that could be increased in each round, while all deposits could be redeemed at the tally stage.

Tally computation. Tallying the results in individual booths requires an exhaustive search for a solution of Equation 2 with $\binom{n+k-1}{k-1}$ possible combinations [13], where n is the number of votes and t is the number of candidates. Therefore, the authority should select the size of the voting groups accordingly to the budget and available computational resources (see [27] for the evaluation).

6 RELATED WORK

We provide a brief survey of e-voting solutions in the following.

Several protocols have been proposed, focusing on ensuring the vote's privacy rather than breaking the map between the voter and her ballot. Cohen and Fisher [7] proposed a verifiable voting scheme where the participants cannot unveil the votes. However, the election authority in this scheme has the ability to read any vote. Cohen [6] provided an extension to this scheme, where the function of authority is distributed among a number of *tellers*: at least one honest teller is sufficient to ensure the privacy of the votes.

A few other works build on the approach from [7], such as [4, 9, 10, 22]. The multi-authority protocol proposed by Cramer et al. [10] employs the ElGamal cryptosystem to guarantee vote privacy. This protocol can tolerate malicious behavior of a constant fraction of authorities. Baudron et al. [2] focused on multi-candidate elections with hierarchical levels of authorities.

Kiayias and Yung [16] introduced a new voting paradigm with several properties they defined – perfect ballot secrecy, self-tallying, and dispute-freeness. The protocol presented by Groth [12] improved the computational complexity of [16] but required more rounds of computation as a trade-off. Hao et al. [13] further improved this approach and created a 2-round self-tallying voting scheme. Khader et al. [15] proposed a variant of [13] that also ensures fairness and robustness.

Protocols based on [7] and [16] as well as other approaches [1, 5, 8, 18, 21] require a public bulletin board (PBB), defined as a broadcast channel with memory. According to its definition, PBB is not affected by denial-of-service attacks and allows each participant to write solely in her designated section in an append-only manner. To achieve these properties in practice, Cramer et al. [10] suggest implementing PBB as a set of replicated servers running a Byzantine agreement protocol. The introduction of blockchain technology brought a suitable solution for a practical instantiation of PBB since it offers the required properties of immutability and availability.

McCorry et al. [19] were the first to implement the self-tallying scheme using smart contracts on Ethereum in the system called Open Vote Network (OVN). However, OVN is only suitable for a small-scale (boardroom) voting. Venugopalan et al. [27] presented BBB-Voting, also a boardroom voting protocol, but with several improvements in contrast to OVN. BBB-voting [27] supports multiple candidate choices, fault recovery, and provides cost-optimized implementation on Ethereum. Seifelnasr et al. [24] improved the scalability of [19] by reducing the storage requirements and delegating the tally computation to an off-chain entity.

Besides self-tallying approaches, other blockchain-based voting systems have been proposed, such as Zhang et al. [33], Dagher et al. [11] (BroncoVote), Killer et al. [17] (Provotum), Venugopalan et al. [28] (Always on Voting), and Zhang et al. [32] (Chaintegrity). Blockchain-based voting was also criticized by Park et al. [20] for bringing additional security issues rather than improvements.

7 CONCLUSION

In this paper, we present a scalable self-tallying blockchain-based voting protocol. We implemented the protocol and evaluated its performance on two EVM-compatible platforms – Gnosis and Harmony. We showed that our protocol is scalable to accommodate large-scale voting, with the only limitation being the throughput of the underlying blockchain platform. Our experiments show that our system can run voting with millions of participants on a sufficiently fast blockchain (e.g., Harmony).

In future work, we will focus on replacing the NIZK proofs of the voting phase with zk-SNARKs to improve on-chain overhead of vote casting. Furthermore, we intend to investigate the techniques that increase the throughput of smart contract platforms (e.g., sharding) and analyze the impact of these approaches on the security properties and scalability of SBvote.

ACKNOWLEDGMENTS

This work was supported by the VALU3S project, which has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876852. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Czech Republic, Germany, Ireland, Italy, Portugal, Spain, Sweden, and Turkiye. The views expressed in this document are the sole responsibility of the authors and do not necessarily reflect the views or position of the European Commission. The authors, the VALU3S Consortium, and the ECSEL JU are not responsible for the use which might be made of the information contained here. Also, this work was supported by the FIT BUT internal project FIT-S-23-8151.

REFERENCES

[1] Ben Adida. 2008. Helios: Web-based Open-Audit Voting. [Online]. In *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA.* USENIX Association, Berkeley, California, 335–348.

[2] Olivier Baudron et al. 2001. Practical Multi-candidate Election System. In *PODC '01* (Newport, Rhode Island, USA). ACM, New York, NY, USA, 274–283.

[3] Josh Benaloh, Ronald Rivest, Peter YA Ryan, Philip Stark, Vanessa Teague, and Poorvi Vora. 2015. End-to-end verifiability. *arXiv preprint arXiv:1504.03778* (2015).

[4] Josh C Benaloh and Moti Yung. 1986. Distributing the Power of a Government to Enhance the Privacy of Voters. In *PODC '86* (Calgary, Alberta, Canada). ACM, New York, NY, USA, 52–62.

[5] Michael R Clarkson, Stephen Chong, and Andrew C Myers. 2008. Civitas: Toward a secure voting system. In *2008 IEEE Symposium on Security and Privacy (sp 2008).* IEEE, 354–368.

[6] Josh D Cohen. 1986. *Improving privacy in cryptographic elections.*

[7] Josh D. Cohen and Michael J. Fischer. 1985. A Robust and Verifiable Cryptographically Secure Election Scheme. In *SFCS '85.* IEEE Computer Society, Washington, DC, USA, 372–382.

[8] Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. 2019. Belenios: a simple private and verifiable electronic voting system. In *Foundations of Security, Protocols, and Equational Reasoning.* Springer, 214–238.

[9] Ronald Cramer, Matthew Franklin, Berry Schoenmakers, and Moti Yung. 1996. Multi-authority secret-ballot elections with linear work. In *International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 72–83.

[10] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. 1997. A Secure and Optimally Efficient Multi-Authority Election Scheme. In *EUROCRYPT '97, Konstanz, Germany, May 11-15, 1997, Proceeding,* Vol. 1233. Springer, New York City, 103–118.

[11] Dagher et al. 2018. BroncoVote: Secure Voting System using Ethereum's Blockchain. [Online]. In *ICISSP 2018, Funchal, Madeira - Portugal, January 22-24, 2018.* SCITEPRESS, SetÃžbal, Portugal, 96–107.

[12] Jens Groth. 2004. Efficient Maximal Privacy in Boardroom Voting and Anonymous Broadcast. In *Financial Cryptography*, Ari Juels (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 90–104.

[13] Feng Hao, Peter Y. A. Ryan, and Piotr Zielinski. 2010. Anonymous voting by two-round public discussion. *IET Information Security* 4, 2 (2010), 62–67.

[14] James Heather and David Lundin. 2008. The append-only web bulletin board. In *International Workshop on Formal Aspects in Security and Trust.* Springer, 242–256.

[15] Dalia Khader, Ben Smyth, Peter Y. A. Ryan, and Feng Hao. 2012. A Fair and Robust Voting System by Broadcast. In *(EVOTE 2012), July 11-14, 2012, Castle Hofen, Bregenz, Austria.* GI, Bonn, Germany, 285–299.

[16] Aggelos Kiayias and Moti Yung. 2002. Self-tallying Elections and Perfect Ballot Secrecy. In *Public Key Cryptography*, David Naccache and Pascal Paillier (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 141–158.

[17] C. Killer, B. Rodrigues, E. J. Scheid, M. Franco, M. Eck, N. Zaugg, A. Scheitlin, and B. Stiller. 2020. Provotum: A Blockchain-based and End-to-end Verifiable Remote Electronic Voting System. In *2020 IEEE 45th Conference on Local Computer Networks (LCN).* 172–183. https://doi.org/10.1109/LCN48667.2020.9314815

[18] Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. 2020. {VoteAgain}: A scalable coercion-resistant voting system. In *29th USENIX Security Symposium (USENIX Security 20).* 1553–1570.

[19] Patrick McCorry, Siamak F. Shahandashti, and Feng Hao. 2017. A Smart Contract for Boardroom Voting with Maximum Voter Privacy. In *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers.* Springer-Verlag, Berlin, Heidelberg, 357–375.

[20] Sunoo Park, Michael Specter, Neha Narula, and Ronald L Rivest. 2020. Going from bad to worse: from internet voting to blockchain voting.

[21] Peter Y. A. Ryan, David Bismark, James Heather, Steve Schneider, and Zhe Xia. 2009. Prêt à voter: a voter-verifiable voting system. *IEEE Trans. Information Forensics and Security* 4, 4 (2009), 662–673.

[22] Berry Schoenmakers. 1999. A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic Voting. In *CRYPTO' 99.* Springer Berlin Heidelberg, Berlin, Heidelberg, 148–164.

[23] SECG SEC. 2000. 2: Recommended elliptic curve domain parameters. *Standards for Efficient Cryptography Group, Certicom Corp* (2000).

[24] Mohamed Seifelnasr, Hisham S Galal, and Amr M Youssef. 2020. Scalable open-vote network on ethereum. In *International Conference on Financial Cryptography and Data Security.* Springer, 436–450.

[25] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J Fischer, and Bryan Ford. 2017. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy (SP).* Ieee, 444–460.

[26] Peter Ullrich. 2017. The risk to breach vote privacy by unanimous voting. *Journal of Information Security and Applications* 35 (2017), 168–174.

[27] Sarad Venugopalan, Ivan Homoliak, Zengpeng Li, and Pawel Szalachowski. 2020. BBB-Voting: 1-out-of-k Blockchain-Based Boardroom Voting. *arXiv preprint arXiv:2010.09112* (2020).

[28] Sarad Venugopalan, Ivana Stančíková, and Ivan Homoliak. 2021. Always on voting: A framework for repetitive voting on the blockchain. *arXiv preprint arXiv:2107.10571* (2021).

[29] Witnet Team. 2019. elliptic-curve-solidity. [Online].

[30] Gavin Wood. 2014. Ethereum: A Secure Decentralized Generalized Transaction Ledger. [Online].

[31] Yang Yang, Zhangshuang Guan, Zhiguo Wan, Jian Weng, Hwee Hwa Pang, and Robert H Deng. 2021. PriScore: blockchain-based self-tallying election system supporting score voting. *IEEE Transactions on Information Forensics and Security* 16 (2021), 4705–4720.

[32] Shufan Zhang, Lili Wang, and Hu Xiong. 2020. Chaintegrity: blockchain-enabled large-scale e-voting system with robustness and universal verifiability. *International Journal of Information Security* 19, 3 (2020), 323–341.

[33] Wenbin Zhang et al. 2018. A Privacy-Preserving Voting Protocol on Blockchain. In *CLOUD 2018, San Francisco, CA, USA, July 2-7, 2018.* IEEE, Washington, DC, 401–408.

This article has been accepted for publication in IEEE Transactions on Emerging Topics in Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TETC.2023.3315748

Always on Voting: A Framework for Repetitive Voting on the Blockchain

Sarad Venugopalan, Ivana Stančíková, Ivan Homoliak

Abstract—Elections repeat commonly after a fixed time interval, ranging from months to years. This results in limitations on governance since elected candidates or policies are difficult to remove before the next elections, if needed, and allowed by the corresponding law. Participants may decide (through a public deliberation) to change their choices but have no opportunity to vote for these choices before the next elections. Another issue is the peak-end effect, where the judgment of voters is based on how they felt a short time before the elections. To address these issues, we propose Always on Voting (AoV) – a repetitive voting framework that allows participants to vote and change elected candidates or policies without waiting for the next elections. Participants are permitted to privately change their vote at any point in time, while the effect of their change is manifested at the end of each epoch, whose duration is shorter than the time between two main elections. To thwart the problem of peak-end effect in epochs, the ends of epochs are randomized and made unpredictable, while preserved within soft bounds. These goals are achieved using the synergy between a Bitcoin puzzle oracle, verifiable delay function, and smart contracts.

Index Terms—Blockchain Governance, Voting, Security, Peak-End Effect, Verifiable Delay Function.

1 Introduction

Voting is an integral part of democratic governance, where eligible participants can cast a vote for their representative (candidate or policy) through a secret ballot. The outcome is an announcement of winners through a tally of votes. In practice, the time interval between two regularly scheduled elections is usually large – ranging from months to years. Over time, a previously popular winning candidate (or policy) may have fallen out of favor with the majority of participants. Therefore, we argue that a common lacking attribute in governance is the ability of participants to reverse or correct the previous decisions that were collectively voted for when new information is available after the election. Reasons for poor decision making (i.e., error premise [1]) may arise from insufficient or false information, search engine manipulation [2], social media manipulation [3], or from agenda setters [4]. To deal with this issue, we propose a repetitive voting strategy, which gives its participants the ability to change their vote anytime they decide. Nevertheless, even in a repetitive voting with fixed time intervals, participants remain exposed to constant manipulation attacks. However, in contrast to standard voting with long time intervals, participants of repetitive voting might hold any elected candidate accountable by changing their vote choice.

A second concern is a peak-end effect, whose discovery in behavioral science is attributed to Nobel laureate Kahneman and his research collaborators [5]. Their study on the correlation of pain perception over time indicated that duration plays a minor part in retrospective evaluations of aversive experiences. The experiences are also dominated by discomfort at the worst and the final moments of episodes. Carmon and Kahneman [6] found that how participants felt at the final moment of the experience was a good predictor of their overall experience evaluation responses.

There are many studies in political science analyzing and confirming the existence and impact of the peak-end effect that might be caused by economic growth in the elections and pre-elections year [7], [8], [9], increased spending on highly visible areas [10], [11], [12], private and government credit easing [13], strategically planned welfare reforms [14], and cash transfer in exchange for using school and health services by poor households [15] (see background on peak-end-effects in Section 2.5).

Our Approach. We propose Always-on-Voting (AoV) that supports 1-out-of-k candidate voting and runs on a blockchain[1]. AoV has three key features: (1) it works in repetitive epochs, (2) voters are allowed to change their vote anytime before the end of each epoch (when the tally is computed), and (3) ends of epochs are randomized and unpredictable. Only the supermajority of votes can change the previous winning vote choice at the end of each epoch.

In AoV, to thwart[2] peak-end effects and decrease manipulation of participants, we randomize the time intervals between epochs of elections using public randomness and secure it with a verifiable delay function (VDF). The tally

- *Sarad Venugopalan currently has no affiliation.*
- *Ivana Stančíková and Ivan Homoliak are with Brno University of Technology, Czech Republic.*

[1] A Byzantine Fault Tolerant state machine replication protocol.

[2] The voting start and end times need to be known in advance to put in place the policies required to boost the elections campaign and entice a large number of voters over a short period of time. However, with AoV, though it is possible to make pre-elections promises, they run hollow after a while, and the spending budget is distributed over a longer period of time because of the repeated nature of voting and its uncertain tally timing, reducing the peak 'awe' effect, making it a less effective strategy to win votes.

time is random and unpredictable, precluding the interested parties from timely peak-end effect manipulations. One of the obstacles to implementing repetitive voting is increased resource expenditure (e.g., time & money). To alleviate cost concerns, improve decentralization and to enhance security properties such as tamper resistance, we recommend our voting framework to be run on a public permissioned blockchain (see Section 7.2).

Several e-voting and blockchain voting solutions under various security requirements are compared in Section 8.1. In this work, we focus on addressing security challenges arising from introducing repetitive strategy, i.e., randomized tally times and repetitive voting.

Contributions. We make the following contributions.

i) We identify two shortcomings in present governance systems for voting: a) the inability of participants to change their vote between two consecutive elections (e.g., that might be many months or a few years apart), and b) a manipulation of participants via peak-end effect (see Section 2.5).

ii) We propose Always-on-Voting (AoV) framework for repetitive voting, which incorporates voting epochs and alleviates the shortcomings of present governance systems (see Section 5).

iii) We propose the use of public randomness to determine when the current interval of voting should end using commitments to a future event in order to thwart the peak-end effect (see Section 5.4).

iv) Finally, we analyze the randomness of the Bitcoin Proof-of-Work puzzle solution (hereafter referred to as a *nonce*) and AoV entropy requirements in Section 6.3.

2 Background

In this section, we describe the preliminaries required to describe our approach.

2.1 The Blockchain

The data structure used in the blockchain represents an append-only distributed ledger. Its entries consist of transactions aggregated within ordered blocks. The order of the blocks is agreed upon by a mutually untrusted honest majority of participants running a consensus protocol, i.e., consensus nodes (a.k.a., miners). The blockchain is resistant to tampering by design since blocks are linked using a cryptographic hash function. The blocks are considered irreversible (with overwhelming probability) after elapsing a time to finality. Some blockchains are equipped with smart contract platforms that enable the users to write application code (i.e., smart contracts) and execute it. All transactions sent to the blockchain are executed and validated by mutually untrusted consensus nodes. In this way, smart contracts enable trusted code execution, where the trust relies on the honest threshold of consensus nodes (e.g., greater than 50% in Proof-of-Work protocols and 67% in Byzantine Fault Tolerant protocols).

2.2 Bitcoin Proof-of-Work Puzzle

Bitcoin uses Proof-of-Work (PoW) to achieve consensus among consensus nodes (a.k.a, miners). A block in Bitcoin is generated once every 10 minutes, on average. A block consists of 2 parts, the header and body. The Bitcoin header has a field called *nBits* that encodes Bitcoin mining difficulty. The *merkle root* field stores the root of the Merkle hash tree corresponding to the transactions in the block. The Bitcoin PoW puzzle is a lottery solved by finding a nonce s; such that the SHA-256 hash of the Bitcoin block header that contains s is lower than $target$.[3] The 32-bit nonce itself is a part of the header and adjusted using a random trial-and-error approach until a solution is found. The mining difficulty changes every 2016 blocks (i.e., ∼every 2 weeks): it is decreased if it took more time to mine 2016 blocks and increased if less time was required. In this work, we use Bitcoin headers from already generated blocks as the source of randomness (see details in Section 6.3).

2.3 Verifiable Delay Function

The functionality of VDF [16] is similar to a time lock,[4] but in addition to it, by providing a short proof, a verifier may easily check if the prover knows the output of the VDF. The function is effectively serialized, and parallel processing does not help to speed up VDF computation. A moderate amount of sequential computation is required to compute VDF. Given a time delay t, a VDF must satisfy the following conditions: for any input x, anyone equipped with commercial hardware can find y = VDF(x, t) in t sequential steps, but an adversary with p parallel processing units must not distinguish y from a random number in significantly fewer steps. For our purposes, the value of t is fixed once it is determined. Therefore, to simplify our notation, we use VDF(x) instead of VDF(x, t) in the remaining text. Further, given output y of VDF, the prover can supply a proof π to a verifier, who may check the output y = VDF(x) using π in logarithmic time w.r.t. time delay t (i.e., $VDF\_Verify(y, \pi) \overset{?}{=} True$).

Finally, the safety factor A_{max} is defined as the time ratio that the adversary is estimated to run VDF computation faster on proprietary hardware as opposed to a benign VDF computation using commercial hardware (see Drake [17]). CPU overclocking records [18] indicate that $A_{max} = 10$ is a reasonable estimate.

2.4 E-Voting

Typically, e-voting approaches have the setup phase, the registration phase, followed by the voting and tally phases. E-voting approaches include actors such as the election authority, candidates, and participants (i.e., voters). In recent years, many blockchain-based e-voting approaches emerged (e.g., [19], [20], [21], [22], [23]) since blockchains enable not only to instantiate the immutable public bulletin board required for e-voting [24] but also provide other features such as censorship-resistance and correct execution of smart contract code, which are beneficial in this context, e.g., to verify the correctness of submitted (encrypted/blinded) votes and compute the tally in a publicly verifiable fashion. Also, blockchains contribute to end-to-end verifiability [22], [25] as well as universal verifiability [21], [24].

[3] See https://learnmeabitcoin.com/technical/target.

[4] Time locks are computational problems that can only be solved by running a continuous computation for a given amount of time.

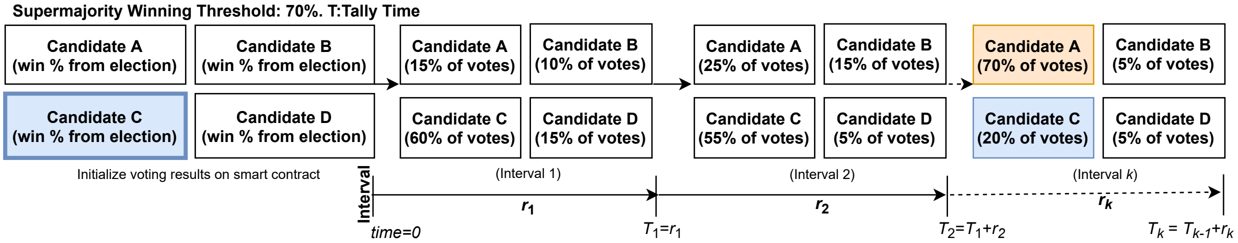

Supermajority Winning Threshold: 70%. T:Tally Time
Candidate A (win % from election)
Candidate B (win % from election)
Candidate C (win % from election)
Candidate D (win % from election)
Initialize voting results on smart contract
Candidate A (15% of votes)
Candidate B (10% of votes)
Candidate C (60% of votes)
Candidate D (15% of votes)
Candidate A (25% of votes)
Candidate B (15% of votes)
Candidate C (55% of votes)
Candidate D (5% of votes)
Candidate A (70% of votes)
Candidate B (5% of votes)
Candidate C (20% of votes)
Candidate D (5% of votes)
Interval
(Interval 1)
(Interval 2)
(Interval k)
r_1
r_2
r_k
time=0
T_1=r_1
T_2=T_1+r_2
T_k = T_{k-1}+r_k

Figure 1. The time between two regular elections is divided into the fixed number ft of intervals (a.k.a., epochs). First, the ratios of votes for all vote choices (i.e., candidates) are initialized from the last election. Next, repeated voting within k epochs results in a winning vote choice transition (from C to A). The new winner A is declared when she obtains a supermajority of total votes (i.e., 70%) at interval k; $k \leq ft$ (see Section 5). Note that $r_1, \ldots, r_k$ are randomized times that determine the length of the intervals $1, ..., k$. The tally is computed at the end of each interval.

2.5 Peak End Effects

In political science, the consequences of peak-end effects on voting have been extensively reviewed. We summarize the results of such studies in the following.

Healy and Lenz [8] studied the bias in voter response to the elections-year economy. It showed that respondents put 75% weight on the elections-year and 25% weight on the year before, whereas the first 2 years of 4 years considered were insignificant. They showed that when the respondents were presented with easy-to-follow information on economic growth for all 4 years, the bias significantly decreased. Wlezien [9] showed that voters weighed their decision not only on the elections-year economic indicators but also on the pre-elections year. The author attaches equal importance to the pre-elections year in influencing voter decisions. Dash and Ferris [7] found the impacts of income growth during the election year as more significant in deciding the incumbents' re-election chances.

Kern and Amri [13] demonstrated that both private and government credit easing was common during the election year to court votes and stimulate credit growth. Li et al. [10] showed the existence of a pattern to political investment cycles – significantly more spending occurred during the election year. Olejnik [11] conducted a study on Polish local government investment expenditure during the election year, indicating a significant increase in highly visible areas such as public infrastructure, tourism, and culture. Guinjoan and Rodon [12] observed a similar pattern of increased spending in highly visible areas such as local festivities during the election year. Voters without expert knowledge in evaluating the investments may base their vote on the visibility of the actions taken. Wenzelburger et al. [14] presented empirical evidence on how governments strategically plan welfare reforms to expand their benefits as the elections approach. Galiani et al. [15] indicated the peak-end pattern of the experiment on cash transfer in exchange for using school and health services by poor households in Honduras. The study showed the increased vote share of the incumbent party in the elections, indicating sensitivity to recent economic activity.

Aguiar-Conraria et al. [26] examined the role of government transparency, i.e., the disclosure of relevant information. For the municipalities whose transparency indexes were high, voters rewarded policies that brought long-term benefits. The results of Carlin et al. [27] showed that restrictions on relevant information distorted the ability of voters to choose in their best interests. On the other hand, transparent governance and independent media allowed voters to hold them accountable.

3 System & Adversary Model

3.1 System Model

Our model has the following main actors and components: i) A *participant* (P) who partakes in governance by casting a vote for her choice or candidate. ii) *Election Authority* (EA) is responsible for validating the eligibility of participants to vote in elections, registering them, and shifting between the phases of the voting. A single EA might be replaced by multiple election authorities (EAs) to improve decentralization. For example, a quorum of $> 2/3^{rd}$ of its EAs must be in agreement to make election decisions. In our case, a single election authority is considered for simplicity. iii) A *smart contract* (SC) collects the votes, acts as a verifier of valid voting, enforces the rules of the election and verifies the tallies of votes. iv) *Bitcoin Puzzle Oracle* (BPO) provides an off-chain data feed from the Bitcoin network and supplies the requested Bitcoin block header (BH) when it is available on the Bitcoin network. v) A *VDF prover* is any benign party in the voting ecosystem who computes the output of VDF and supplies proof of its correctness to SC.

3.2 Adversary Model

The adversary in our voting framework with respect to epoch triggering is a Bitcoin mining adversary Adv_{min}. This adversary is static and has bounded computing power, i.e., it is unable to break used cryptographic primitives under the standard security assumptions. Adv_{min} can mine on the Bitcoin blockchain. Her goal is to find a solution to the Bitcoin PoW puzzle that also triggers the end of the current voting interval, thereby influencing the end time of epoch. Our voting framework uses a function of the Bitcoin block header (BH) inclusive of its PoW solution s, i.e., $f(BH)$ to trigger the end of the current voting interval. Such a manipulation would potentially enable Adv_{min} to prematurely finish the current interval and start the next one (see Section 5.4).

Finally, we assume that EA verifies identities honestly and supply addresses of only verified participants to SC.

4 Solution outline & Design Goals

4.1 Solution Outline

In certain forms of governance (such as with traditional governance), elections are repeated only after many months/years. Here, its participants will have to wait until the next election to change their vote. Due to the fixed time allocated for governance to the winning candidate, it is not possible to change this elected candidate before the allocated time for governance expires. During the time period between 2 consecutive elections, the incumbent may have fallen out of favor to a majority of the participants but current voting frameworks have no option to re-vote until the time to next election has elapsed. This results in limitations on governance since elected candidates or policies are difficult to remove before the next elections. This is an issue we address in our voting framework by breaking it down into smaller repeated epochs (along with thwarting the peak-end-effects in elections).

Our solution presents a 1-out-of-k voting framework repeated over time. When the voting is repeated over a fixed time epoch, its working is similar to voting carried out in a traditional election. However, in our proposed framework, we introduce three main changes. (1) The time interval between consecutive elections is shorter when compared to regularly held elections (repeated after months or years). (2) A single trigger is used to end the current voting epoch and to immediately start the next voting epoch. Hence, there is no time delay between any two consecutive voting epochs. (3) To thwart the peak-end-effect in voting (see Section 2.5), a variable-time epoch voting is used. Here, the start time of a new and upcoming voting epoch is not known in advance and it cannot be conclusively determined by any of the parties in the election.

There exist a few issues related to centralized e-voting, such as censorship and tampering with the results and data (see Section 7.4). For these reasons, our voting framework may be implemented on a blockchain, such as using Hyperledger projects (see Section 7.2). The bitcoin headers from the Bitcoin network are used as an initial source of randomness (see Section 6.3) to trigger the start of the voting epochs. This is unrelated to the blockchain platform on which the voting was implemented. Bitcoin headers were chosen because of the difficulty for the mining adversary to find a suitable nonce to the Bitcoin PoW puzzle within 10 minutes (on average), that would trigger the voting epoch.

The VDF is an additional component used in securing the arbitrary length interval from being maliciously triggered by the mining adversary. The newly mined blocks from the Bitcoin network are sent to a VDF, to further delay the mining adversary from maliciously triggering the voting epoch. Since Bitcoin mining is a lottery, the Bitcoin network accepts the first valid nonce from any miner and appends that mined block to the chain, thereby preventing the malicious miner attack on our voting framework. The focus of our voting framework is in thwarting the peak-end-effect and addressing the security issues due to the introduction of variable time epoch triggers.

4.2 Design Goals

The AoV framework has the following main design goals.

1) **Repeated voting epochs**: Participants are allowed to continuously vote and change elected candidates or policies without waiting for the next election. Participants are permitted to privately change their vote at any point in time, while the effect of their change is considered rightful at the end of each epoch. The duration of such epochs is shorter than the time between the two main elections.
2) **Randomized time epochs**: The end of each epoch is randomized and made unpredictable. In contrast to fixed-length time epochs, the proposed randomized time epochs are used to thwart the peak-end-effect.
3) **Plug & play voting protocols**: The AoV framework is designed to "plug & play" new or existing voting protocols. As a result, AoV inherits the properties of the underlying protocol chosen. However, in the interest of vote confidentiality on a blockchain, we recommend protocols providing secret ballots whose correctness can be publicly verified by SC without leaking any information, e.g., [21], [22], [28], [29]. Also, due to the repetitive nature of AoV, e-voting protocols with expensive on-chain computations and required fault recovery (due to stalling participants) may be less appealing but still acceptable with some limitations, e.g., [19], [20], [23], [30], [31] (see also Section 8.2).

5 Always on Voting Framework

Always-on-Voting (AoV) is a framework for blockchain-based e-voting, in which voting does not end when the votes are tallied and the winners are announced. Instead, participants can continue voting for their previous vote choice or change their vote. A possible outcome of such repetitive voting is transitioning from a previous winning candidate to a new winner. To achieve this, the whole time interval between two regularly scheduled elections is unpredictably divided into several intervals, denoted as voting epochs. Participants may change their vote anytime before the end of a voting epoch (i.e., before a tally of the epoch is computed); however, they do not know beforehand when the end occurs. Any vote choice that transitioned into the supermajority threshold of votes is declared as the new winner of the election, and it remains a winning choice until another vote choice reaches a supermajority threshold.

5.1 Underlying Voting Protocol

AoV provides the option to plug & play any suitable e-voting protocol. To provide the baseline security and privacy of votes (with on-chain verifiability), we assume the voting protocol plugged into AoV allows participants to blind or encrypt their votes whose correctness is verified on-chain by SC. However, AoV does not deal with other features supported by the plugged-in voting protocol (such as end-to-end verifiability [32], coercion-resistance [21], receipt-freeness [24], and fairness [24]).

5.2 Example of Operation

Figure 1 illustrates a scenario with 4 candidates A-D, where C is the present winner of the election. For example, the supermajority threshold of 70% votes is set for future winnings, which is a tunable parameter that may be suitably tailored to the situation. All candidates are initialized to their winning percentages of obtained votes from the last election. Over time, the individual tally is observed to shift

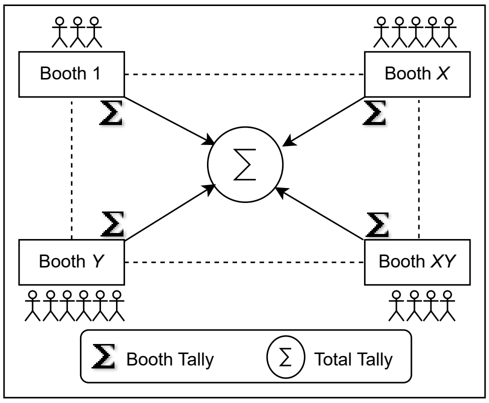


Figure 2. When the tally computation is triggered, each booth computes the sum of all votes cast at the booth (referred to as booth tally). Each booth tally is further summed up to determine the total tally. Pictorially, the booths are numbered 1 to X along the rows and 1 to Y along the columns. There are a total of $X \cdot Y$ booths.

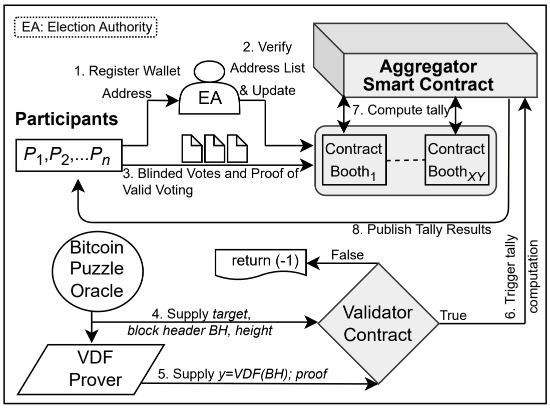

EA: Election Authority
2. Verify Address List & Update
1. Register Wallet Address
EA
Aggregator Smart Contract
Participants
P1,P2,...Pn
7. Compute tally
Contract Booth1
Contract BoothXY
3. Blinded Votes and Proof of Valid Voting
8. Publish Tally Results
6. Trigger tally computation
Bitcoin Puzzle Oracle
return (-1)
False
Validator Contract
True
4. Supply target, block header BH, height
VDF Prover
5. Supply y=VDF(BH); proof

Figure 3. Interaction among participants (Ps), election authority (EA), smart contracts, the Oracle, and VDF prover. (1) Registering wallet addresses of participants and (2) their identity verification are made by the EA. (3) Participants send a blinded vote and its zero-knowledge proof of correctness to their assigned booth contract. The booth contract verifies the validity of the vote. (4) The Bitcoin Puzzle Oracle (BPO) provides the latest Bitcoin block header (BH) and (5) VDF prover sends a proof of sequential work with y (the output of VDF(BH)) to the validator contract. (6) The validator contract finishes the epoch and shifts the state of the elections to the tally upon meeting the required conditions. (7) The aggregator contract is responsible for totaling individual booth tallies and (8) publicly announcing the total tally. The on-chain components of AoV are depicted in gray.

as the supermajority of participants decided to change their vote in favor of another candidate by voting in the epoch intervals. Through k intervals, the winner-ship is seen to transition from candidate C to A. At the k^{th} interval, A obtains the 70% threshold of votes and is declared as the new winner. Note that the supermajority is required only in the voting epochs between two regularly scheduled elections. The regular elections are also executed in AoV, and they repeat every M months/years, while requiring only a majority of votes (i.e., >50%) to declare a winner. Hence, in contrast to existing electoral systems, we only propose changes between regularly scheduled elections and enable new candidates to be added or removed (see in Section 7.6).

Justification for Supermajority. A supermajority of 70% was chosen (see Appendix A for background) to help the incumbent carry out reforms without the risk of losing when there is still sufficient support from participants. On the other hand, the main purpose of this threshold is to block (or repeal) policies that are unpopular or negatively affecting a vast majority of participants. Further, participants may be inclined to vote in favor of the referendum/proposal if it is coming from a leader who won with a super-majority. Additionally, we aim to avoid the quorum paradox (see Appendix B) by setting a minimum participation requirement of 70% from the just concluded main election.

5.3 Overview of AoV Phases

Once the setup phase (that ensures participants agree upon all system parameters) is completed, electronic voting frameworks typically consist of three phases: (1) a registration phase to verify voter credentials and add them to the voting system, (2) a voting phase, in which participants cast their vote via a secret ballot, and (3) a tally phase, where the total votes for each candidate are counted and revealed to participants. The voting protocol plugged-in with the AoV framework may contain additional phases, but we omit them here for brevity.

The architecture of AoV is shown in Figure 3. In AoV, participants (in step 1) register their wallet address[5] with the EA, who then (in step 2) verifies and updates it on the booth smart contract[6]. This is followed by the voting phase (in step 3), where participants publicly cast their secret ballots (i.e, not revealing the vote choice nor identity). The BPO (step 4) supplies the validator contract and VDF prover with the $target$, recent Bitcoin block header BH and its block height. The VDF prover[7] (in step 5) computes and submits $VDF(BH)$ and a proof of sequential work (π) to the validator contract. The validator contract (in step 6) verifies the VDF proof and checks whether the supplied nonce s (included in the block header) is a valid solution to the Bitcoin PoW puzzle of the supplied header. If both verifications pass, the validator contract finalizes the epoch and triggers the tally computation for the epoch. Otherwise, it waits for the next block header submission from the BPO and the proof of sequential work from the VDF prover. When the tally computation is triggered, each booth contract $\{1, 2, ..., X \cdot Y\}$, sums up all its local vote counts and sends them to the aggregator contract (step 7). Then, the aggregator contract totals the votes from each booth contract[8] and publishes the final tally (step 8).

In AoV, the EA is authorized to register/remove participants and candidates in a future interval. Nevertheless, candidates can also be managed by other means, and AoV does not mandate how it should be done (see discussion in Section 7.6) When there are no other changes in the next interval, revoting repeats with step 2 and ends with step 8.

From the initialization of AoV until the next regular elections, the validator smart contract accepts all future Bit-

[5] Refer to Appendix C for a proposed method to improve anonymity for wallet users.

[6] Participants are randomly grouped and assigned to booths $\in \{1, 2, ..., X \cdot Y\}$ (see Figure 2), represented by a booth smart contract.

[7] A VDF prover is any benign user in the voting ecosystem with commercial hardware to evaluate the input of VDF, i.e., $y = \text{VDF}(BH)$ and supply a proof π.

[8] Refer to Appendix E for privacy implications of booth sharding.

coin block headers. The new block headers (as part of their blocks) arriving every 10 minutes on average are appended to the Bitcoin blockchain. The BPO is responsible for timely supplying[9] each new block header to the VDF prover and validator contract. The VDF prover computes the VDF on each of those block headers after they are supplied.

5.4 Calculating the Epoch Tally Time

Due to concerns that Bitcoin nonces are a weak entropy source, additional steps are taken to make it cryptographically secure (see details in Section 6.3). Our notion of randomness relies on Bitcoin Proof-of-Work to generate valid nonces.[10] The validator contract awaits future block headers yet to be mined on the Bitcoin network. When new Bitcoin block headers arrive, they are sent to the validator contract and the VDF prover via the BPO. The VDF ensures that a mining adversary cannot find more than one valid nonce to the block at a given height and test if the nonce is favorable within 10 minutes. The VDF is computed with the block header at the input by the VDF prover, who then submits the VDF output and the proof of sequential work to the validator contract. The choice of VDF depends on its security properties, speed of verification, and a size of the proof [33]. Let BH be the Bitcoin block header. Once VDF prover computes $y = VDF(BH)$, a small proof (π) is used to trivially verify its correctness using $VDF\_Verify(y, \pi)$. Wesolowski's construction [34] is known for its fast verification and a short proof: Let TL be the number of sequential computations. Prover claims

$$y \; = \; BH^{2^{TL}}$$

and computes a proof

$$\pi = BH^{\lfloor \frac{2^{TL}}{B} \rfloor},$$

where B = Blake256(BH || y || TL) hash. Verifier checks whether

$$\pi^{B} \cdot BH^{2^{TL} mod\ B} \overset{?}{=} y.$$

Since we employ VDF, Adv_{min} does not know the value of y before evaluating the VDF and is forced to wait for a given amount of time to see if the output is in her favor (before trying again). However, since Bitcoin mining is a lottery, other miners can solve the puzzle and append a block by propagating the solution to the Bitcoin network, rendering any withheld or attempted solution by the adversary that was not published useless.

5.4.1 Interactions of BPO, VDF Prover, and Validator

Let $TotalTime$ be the time in minutes between 2 regular elections. The BPO (see step 4 in Figure 3) feeds the block header BH of every future Bitcoin block (when it is available) to the validator contract and VDF prover. Further, BPO provides validator contrast also with the value of $target$ when it changes; i.e., every 2016 block (see Section 2.2).

Upon obtaining data from BPO, the VDF prover computes VDF output

$$y \; = \; VDF(BH) \tag{1}$$

with the VDF proof π and sends them to the validator contract (see step 5 in Figure 3). Next, the validator contract verifies the following conditions:

$$VDF\_Verify(y, \pi) \overset{?}{=} True, \tag{2}$$

$$SHA256(BH) < target. \tag{3}$$

The first verification checks whether the VDF output y and supplied proof π (i.e., Equation 2) correspond to the BPO-supplied block header BH. The second verification (i.e., Equation 3) checks whether the nonce received from BPO[11] is a valid solution to the Bitcoin PoW puzzle. Once both checks pass, the validator contract proceeds to compute

$$a = SHA(y), \tag{4}$$

where SHA(.) is SHA-X-256[12] hash. The goal of Equation 4 is to consolidate the entropy by passing it through a compression function that acts as a randomness extractor (see Section 6.3). Using a, the validator contract computes

$$b = a \ (mod\ BHsInInterval), \tag{5}$$

where the expected number of block headers are

$$BHsInInterval \; = \; \frac{IntervalTime}{BlockTime}. \tag{6}$$

and the time interval is found as

$$IntervalTime \; = \; \frac{TotalTime}{ft}. \tag{7}$$

As seen in Figure 1, ft is the number of intervals (epochs) that the total time ($TotalTime$) between 2 regular elections is divided into. $BlockTime$ is the average time of block generation (i.e., 10 minutes in Bitcoin).

The computation of tally for the current interval is triggered when the output of the validator contract is $True$ (see step 6 in Figure 3):

$$VC_{output} = \begin{cases} \text{True}, & \text{if } b = 0 \\ \text{False}, & \text{otherwise.} \end{cases} \tag{8}$$

[9] To respect the finality of the Bitcoin network, we assume that BPO supplies only the block headers that contain at least 6 confirmations on top of them. As a consequence, the probability that such a confirmed block will be reverted is negligible. Note that this does not influence the chances of ADV_{min} to succeed since she is already "delayed" by VDF in finding multiple PoW solutions at the same height; therefore, she prefers to work on top of the chain with her new attempts.

[10] If the nonce overflows, a parameter called *extraNonce* (part of the coinbase transaction) is used to provide miners with the extra entropy needed to solve the PoW puzzle.

[11] We note that the BPO may be replaced by a quorum to improve decentralization. The validator contract will then accept the input from BPO only when 2/3 (and more) of the quorum is in agreement.

[12] X denotes a suitable hash function such as SHA-3, and 256 is the output length in bits.

This article has been accepted for publication in IEEE Transactions on Emerging Topics in Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TETC.2023.3315748

Example: Let $TotalTime$ = 4 years = $525600 \cdot 4$ minutes and $ft = 8$; then $IntervalTime = (525600 \cdot 4)/(8) =$ 262800 minutes $\approx$ 182.5 days and $BHsInInterval$ = $262800/10$ = 26280 blocks. Therefore, the BPO will send on average 26280 block headers (BH values) to the validator contract within an assumed 182.5 days long epoch (assuming 10 minutes block creation interval), i.e., $1/8$ of the total time. We expect the tally will be triggered on average once in every 182.5 days because of the Poisson probability distribution of this event. Therefore, ft expresses the expected number of epochs, while ft might differ across the regular election iterations.

6 Analysis

6.1 Mining Adversary

The goal of Adv_{min} is to find a valid nonce s that solves the Bitcoin puzzle such that b in Equation 5 is 0. When these two conditions are met, and the new epoch is about to start, the validator contract triggers the tally computation of votes. We set the difficulty for the benign VDF prover (with commercial hardware) to take 100 minutes[13] to solve VDF(BH). Based on A_{max} limit, we assume Adv_{min} to take at least 10 minutes to solve the VDF. As a result, Adv_{min} is restricted to a maximum of 1 try (considering 10 minutes as an average Bitcoin block creation time), excluding the Proof-of-Work required to solve the Bitcoin mining puzzle.

However, since a Bitcoin block header is generated on average once every 10 minutes and the benign VDF prover is occupied for 100 minutes, the question is – how many VDF provers are required to prevent the block headers from queuing up? We can see in Table 1 that VDF prover 1 runs a task for time 0-100 minutes, and she picks up the next task to run for time 100-199 minutes. Similarly, all other provers pick up the next task after completing the present one. Hence, 10 VDF provers are sufficient to prevent block headers from queuing up because $A_{max} = 10$.

VDF Prover	Time (minutes)	VDF Prover	Time (minutes)
1	0-100	1	100-199
2	10-110	2	110-209
3	20-120	3	120-219
4	30-130	4	130-229
5	40-140	5	140-239
6	50-150	6	150-249
7	60-160	7	160-259
8	70-170	8	170-269
9	80-180	9	180-279
10	90-190	10	190-289

Table 1
Scheduling 10 VDF provers without queuing. Note the VDF computations on a VDF prover machine are not parallelized. It is the scheduling alone that is in parallel. The start time is based on the job arrival time at the VDF prover, where it will run for 100 minutes. Once completed, it is ready to take on the next job. In column 2, the start times are 10 minutes apart and correspond to the average BTC interblock (job) arrival time. The largest idle time in column 1 is for VDF Prover 10 at 90 minutes, waiting for the job to start. Beyond this, all VDF prover machines are continuously occupied since a new job is available to start immediately after the current job ends.

On the other hand, a benign VDF prover might reduce A_{max} of VDF computation by using specialized hardware instead of commercial hardware (depending on the cost-to-benefit ratio). However, we emphasize that the VDF can be computed only after solving the PoW mining puzzle, which is prohibitively expensive. Moreover, the puzzle difficulty increases proportionally to the mining power of the Bitcoin network. Hence, the proposed serial combination of solving the Bitcoin mining puzzle followed by the computation of VDF output improves the aggregate security against Adv_{min} from choosing a favorable nonce. The estimated requirement of $A_{max} = 10$ might be further increased as more studies to efficiently solve VDFs on ASICs are carried out. However, if A_{max} will increase in the future, our solution can cope with it by employing more VDF provers.

6.2 Implications of VDF Prover Synchronization and Optimizing Frequency of Supplied Block Headers

Several VDF provers are synchronized to supply the VDF proofs to the validator contract in sequence. However, there are no adverse effects when the proofs are generated and supplied out of sequence. The validator smart contract stores the latest *block height* for which the VDF proof was last accepted. It only allows proof verification for stored *block height+1* on the contract and any out-of-order proofs have to be re-sent. Once the order is corrected, a handful of VDF proofs may appear in quick succession at the validator contract. However, the tally for the interval is only triggered when VC_{output} in Equation 8 is *True*.

In terms of gas consumption, it can be costly to process every single Bitcoin block header (supplied to the VDF prover and the validator contract by the BPO). We suggest optimizing this by choosing a coarser time granularity of the block header supply, independent of the Bitcoin block interval (e.g., every x-th block). We modify the example from Section 5.4.1 by considering the processing of every 100^{th} Bitcoin block header.[14] $TotalTime$ = 4 years = $525600 \cdot 4$ minutes and the total number of intervals $ft = 8$. Then, $IntervalTime = (525600 \cdot 4)/(8) = 262800$ minutes and $BHsInInterval = 262800/(10 \cdot 100) = 262.8$. On average, the oracle will send 262.8 block headers (BH values) to the validator contract within 182.5 days instead of the 26280 block headers required in the original example. In this case, we only need 1 VDF prover instead of 10, and it provides similar security guarantees as before.

6.3 Randomness of Bitcoin Nonces & AoV Entropy

We decided to utilize a single public source of randomness instead of a distributed randomness due to the low computation cost and synchronization complexity. Bonneau et al. [35] showed that if the underlying hash function used to solve the Bitcoin PoW puzzle is secure, then each block in the canonical chain has a computational min-entropy of at least d bits, representing the mining difficulty. I.e., d consecutive 0 bits must appear in the hash of the block header[15]. Hence, $\lfloor \frac{d}{2} \rfloor$ near-uniform bits can be securely extracted. Nevertheless, empirical evaluation has shown that

[13] We consider $A_{max} = 10$, i.e., what is solved by a benign VDF prover in 10 units of time, while in the case of Adv_{min} it is in 1 unit.

[14] Note that this would need another condition to be met, i.e., the block height (mod 100) should be equal to 0.

[15] At the time of writing, $d \approx 76$.

This article has been accepted for publication in IEEE Transactions on Emerging Topics in Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TETC.2023.3315748

Bitcoin nonces have visible white spaces (non-uniformity) in its scatter-plot [36]. A possible explanation is that some miners are presetting some of the bits in the 32-bit $nonce$ field and using the $extraNonce$ to solve the PoW puzzle. We use the entire block header as the initial source of entropy instead of the 32-bit nonce alone to avoid such biases. To reduce the probability of Adv_{min} biasing the solution in her favor, the block header is passed through a verifiable delay function (see Equation 1). The output of VDF is hashed (see Equation 4) to consolidate the entropy.

7 Discussion

7.1 Voting in a Referendum

The proposed framework for repetitive voting is also suitable for a referendum, where participants may vote on a proposal. Unlike elections scheduled at regular intervals of time, a referendum may not be necessarily tabled (i.e., put up for voting) more than once. Nevertheless, AoV may be used to make the referendum voting repetitive. Any change to the outcome of a referendum using the AoV framework requires a supermajority unless the proposal is re-tabled through an agreement, in which case normal operations follow.[16] The danger of low participant turnout in-between regularly scheduled votings may be mitigated by setting a minimum threshold on the number of participants (e.g., 70% of the just concluded elections) to overturn a decision. In the case of a referendum, a similar minimum threshold based on previous participant turnouts is recommended. Further, the proposed framework may also be used as an extra tool to record changing public opinion.

7.2 Voting Costs on a Public Permissioned Blockchain & Incentives for the VDF Prover

The expenses imposed by a public permissionless smart contract platform may be high. Furthermore, the transactional throughput of such platforms may be insufficient to cater to a larger number of participants voting in a specified time window. To reduce costs and improve performance, AoV can run on a public permissioned Proof-of-Authority (PoA)[17] blockchain, e.g., using Hyperledger projects (such as safety-favored Besu with BFT), in which all nodes have the same consensus power. Since nodes in PoA protocols "stake" their reputation that is backed by the knowledge of their identities, any misbehavior can lead to loss of the reputation, which is expensive and nodes are thus naturally disincentivized from such misbehaviors. Optionally, smart contract platforms backed by trusted computing that off-chains expensive computations may be used (e.g., Ekiden [37] and TeeChain [38]). Other partially-decentralized second layer solutions (e.g., Plasma[18], Polygon Matic,[19] and Hydra[20] may also be used. Even though these solutions might preserve most of the blockchain features harnessed in e-voting, availability and decentralization may be decreased. The selection depends on the security/performance trade-off (alike the number of full nodes used).

In a permissioned blockchain, only authorized participants are allowed to append transactions to the chain. Unlike a public blockchain, the only costs involved in a permissioned blockchain reside in maintaining the blockchain infrastructure. Consider the following example to put the costs of the PoA blockchain into the real perspective. Let us assume that 4 independent candidates are contesting the election, along with 6 public notaries and 1 EA. Each of them may run a consensus node or rent it out from a cloud for a low fee (e.g., ∼20-50 USD/month). The consensus nodes might be run by EA, public notaries, and some of the participants. The VDF provers may be incentivized through a cryptocurrency[21] treasury [39] to supply cryptocoins for their services (supplying VDF proofs). A fraction of all transaction fees on the blockchain may be sent to a treasury smart contract and claimed by the VDF prover upon verification of a valid VDF proof by the validator contract in Figure 3. This incentive would ensure a consistent supply of the VDF proofs, which is required to maintain the probability of entering a new epoch within each of the predefined intervals.

7.3 Limitations

AoV may be less suitable for voting where a winner is found through set reduction[22]. AoV is suited for elections with 1-out-of-k voting choices, i.e., plurality voting such as first-past-the-post (FPTP) [40]. Also, FPTP is the second most commonly used electoral system in the world. It covers 59/232 countries in national legislature elections and 24/232 countries in presidential elections [41], encompassing billions of voting participants under this electoral system.

7.4 Blockchain in the AoV Framework

In our approach, the blockchain with smart contract platform enables us to enforce the rules of the elections, including the plugged-in e-voting protocol as well as wrapping the AoV framework itself. In the case of plugged-in protocol, the blockchain verifies the correctness of submitted (private) votes, computes the tally in a verifiable fashion (i.e., tallied-as-recorded verifiability), and enables participants to achieve cast-as-intended and recorded-as-cast verifiability, all together providing end-to-end verifiability [22], [25]. Furthermore, in the case of AoV that wraps the plugged-in blockchain-based e-voting protocols, the blockchain enables to re-vote anytime and the correctness of their vote is verified by a smart contract.

7.5 Randomization of the Epoch End Times

The voting intervals in AoV have soft bounds on when the end of an epoch is triggered, i.e., it triggers when a in Equation 4 has a zero remainder when divided by $BHsInInterval$ in Equation 5. In particular, the event of

[16] Some referendums require only a majority while others need a supermajority to arrive at a decision. AoV does not change the voting requirements when it is first tabled or re-tabled.

[17] In a PoA consensus, a strict vetting process is used for machines that are given the right to generate blocks. Vetting is carried out by pre-approved moderators who checks the blocks and its transactions.

[18] See https://plasma.io/plasma.pdf.

[19] See https://github.com/maticnetwork/whitepaper.

[20] See https://hydra.family/head-protocol/.

[21] Funds may also be supplied via central bank digital currency.

[22] In set reduction, candidates are eliminated based on a tally cut-off criteria and revote is carried out on the reduced candidate set.

interval end follows the Poisson distribution. The end of the interval is triggered faster when the value of $IntervalTime$ is smaller.

To the extent, indecisive voters may be prompted and reminded to vote (e.g., by push notifications in a smartphone, or emails). For example, a smartphone or tablet may prompt the user with a push notification at the start of each new epoch. It may also send reminders that they had not already voted in that epoch after a certain elapsed time. If a voter does not have a smartphone, there may be alternative solutions for notification, such as sending an email.

7.6 Adding and Removing Candidates

AoV allows the list of candidates to be updated at any time by the authorized entity (or entities), and thus candidates not willing to participate can be removed in the new epoch. Similarly, new candidates can be added. Such rules may be added to the smart contract based on the agreement between the EA and the candidate (or even in some decentralized fashion). In the context of this work, we abstract from the implementation details of this aspect, and we focus on addressing the continuous voting and peak-end effect.

8 Related Work

8.1 Blockchain-Based E-Voting

Many blockchain-based e-voting protocols and systems are present in the literature, out of which, we are not aware of any continuous e-voting system that addresses the peak-end effect problem. In the following, we extend the categorization of remote blockchain-based e-voting systems proposed by Yu et al. [21] with more examples.

8.1.1 Voting Systems Using Smart Contracts

McCorry et al. [19] proposed OVN, a self-tallying voting protocol that provides vote privacy and supports two vote choices; however, it does not provide robustness (i.e., recovery from stalling participants) and suffers from expensive computation on the smart contract. A similar approach based on the same protocol was proposed by Li et al. [30], who further provided robustness (based on Khander et al. [42]). Seifelnasr et al. [20] increased the scalability of OVN by off-chaining tally computation and registration at EA. Due to the higher costs imposed by storing data on smart contracts, they compute the Merkle tree of voter identities and store only its root hash in the smart contract. Their approach requires only a single honest participant to maintain the protocol's security. Venugopalan et al. [23] proposed BBB-Voting, an approach that on top of OVN features enables $k \geq 2$ vote choices and provides robustness and further cost optimizations. Stančíková and Homoliak [31] proposed SBvote, an approach that extends BBB-Voting by integrity-preserving batching and hierarchical booth aggregation to improve scalability, which supports millions of participants. Yu et al. [21] employed ring signatures to ensure that the vote is from one of the valid choices, and they achieve scalability by linkable ring signature key accumulation. Their approach provided receipt-freeness under the assumption of trusted EA; however, it does not provide end-to-end (E2E) verifiability. Killer et al. [22] presented Provotum, an E2E verifiable remote voting system with 2 vote choices. The authors employed threshold cryptography to achieve robustness using a scheme similar to Shamir secret sharing. Matile et al. [28] proposed a voting system providing cast-as-intended (but neither E2E nor universal) verifiability. Their system uses ElGamal encryption based on DLP with integers modulo p.

8.1.2 Voting Systems Using Cryptocurrency

Zhao and Chan [43] proposed a privacy-preserving voting system with 2 vote choices based on Bitcoin, which uses a lottery-based approach with an off-chain distribution of voters' secret random numbers. Random numbers enable to hide the vote choice and are distributed via ZKP. The authors use deposits to incentivize participants to comply with the protocol. Tarasov and Tewari [44] proposed a conceptual voting system based on Zcash. The voter's anonymity (and thus the privacy of vote) is ensured by the z-address that preserves unlinkability. The correctness of the voting is guaranteed by the trusted EA and the candidates. Liu and Wang [45] proposed a conceptual voting approach that is based on blind signatures with 2 vote choices. They utilized blockchain only for auditable sending of messages among participants and EA.

8.1.3 Commercial Voting Systems with Ballot Box

This category of voting systems usually does not offer the privacy of votes, instead relies on the unlinkability of blockchain wallet addresses with participant identities; therefore, one may use VPNs and/or anonymization services. Examples are FollowMyVote[23], Tivi,[24], and Agora,[25] which use blockchain only for recording votes into a *ballot box*. In contrast to the previous solutions, NetVote[26] is a solution addressing privacy by trusting EA to reveal the encryption key of ballots after the elections end.

8.2 Plug-able Protocols into AoV

We reviewed several blockchain-based e-voting approaches, out of which, the smart contract-based ones (see Section 8.1.1) are suitable to be plugged into AoV. The most convenient approaches to be plugged in AoV are the ones that allow re-voting within an epoch (i.e., participants can change their votes), such as [21], [22], [28], [29]. However, approaches using distributed MPC keys, a.k.a., self-tallying protocols [24] such as [19], [20], [23], [30], [31] have a limitation in terms of AoV despite having other benefits (e.g., perfect ballot secrecy, fairness). In particular, they can be applied in AoV with a limitation of a single vote within an epoch. They do not enable re-voting within an epoch since they provide the privacy of vote only with a single cast of a vote (i.e., requirements on the shared MPC key).

9 Conclusions

We reviewed the works in political science that motivated an engineering solution to the peak-end effect problem in

[23] See https://followmyvote.com/.
[24] See https://tivi.io/tivi/.
[25] See https://www.agora.vote/.
[26] See https://github.com/netvote/elections-solidity.

voting and voter manipulation. We showed that existing voting systems provide little to no recourse with changing the elected candidates (until the next main elections) – even when they lost the support of a majority of voters. Therefore, we proposed the Always-on-Voting (AoV) framework that allows participants to change their vote between two main elections and thwart the peak-end effects. To achieve this in unbiased fashion, we divided the time between two main elections into a few shorter epochs whose ends were made unpredictable, and tallying the votes at the end of each epoch. The AoV framework used verifiable delay functions and Bitcoin block headers as the source of randomness to thwart the mining adversary who intends to bias the ends of epochs. AoV recommended a public permissioned blockchain to ensure its security properties and save costs. It can be integrated with various existing blockchain-based e-voting solutions. Finally, we proposed a supermajority requirement to elicit a change reflecting the current result of the elections, which helps to maintain the stability of the existing system.

Acknowledgment

The authors would like to thank Paweł Szałachowski, Stefanos Leonardos, Daniel Reijsbergen, Mike Rosing, and Balamurali B. T. for their comments on an earlier version of the paper. Ivana Stančíková and Ivan Homoliak were supported by the DIAS project, which has received funding from the ECSEL Joint Undertaking under grant agreement No 8A21012, 101007273. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of their employer(s).

References

[1] C. Blum and C. I. Zuber, "Liquid democracy: Potentials, problems, and perspectives," *Journal of Political Philosophy*, vol. 24, no. 2, pp. 162–182, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/jopp.12065

[2] R. Epstein, R. E. Robertson, D. Lazer, and C. Wilson, "Suppressing the search engine manipulation effect (SEME)," *PACMHCI*, vol. 1, no. CSCW, pp. 42:1–42:22, 2017.

[3] R. S. Goldzweig, I. Kirova, B. Lupion, M. Meyer-Resende, and S. Morgan, "Social media monitoring during elections: Cases and best practice to inform electoral observation missions," in *Democracy Reporting International*, R. Taylor, Ed. Open Society Foundations, 2019.

[4] F. J. Z. Borgesius, J. Möller, S. Kruikemeier, R. Ó-Fathaigh, K. Irion, T. Dobber, B. Bodo, and C. de Vreese, "Online political microtargeting: Promises and threats for democracy," in *Proceedings of the 22Nd Annual International Cryptology Conference on Advances in Cryptology*, ser. Utrecht Law Review, 2018, pp. 82–96. [Online]. Available: https://ssrn.com/abstract=3128787

[5] D. Kahneman, B. L. Fredrickson, C. A. Schreiber, and D. A. Redelmeier, "When more pain is preferred to less: Adding a better end," *Psychological Science*, vol. 4, no. 6, pp. 401–405, 1993. [Online]. Available: https://doi.org/10.1111/j.1467-9280.1993.tb00589.x

[6] Z. Carmon and D. Kahneman, "The experienced utility of queuing: Experience profiles and retrospective evaluations of simulated queues," Ph.D. dissertation, 01 1996.

[7] B. B. Dash and J. S. Ferris, "Economic Performance and Electoral Volatility: Testing the Economic Voting Hypothesis on Indian States, 1957–2013," Carleton University, Department of Economics, Carleton Economic Papers 18-07, Jun. 2018. [Online]. Available: https://ideas.repec.org/p/car/carecp/18-07.html

[8] A. Healy and G. S. Lenz, "Substituting the end for the whole: Why voters respond primarily to the election-year economy," *American Journal of Political Science*, vol. 58, no. 1, pp. 31–47, 2014. [Online]. Available: http://www.jstor.org/stable/24363467

[9] C. Wlezien, "The myopic voter? the economy and us presidential elections," *Electoral Studies*, vol. 39, pp. 195–204, 2015. [Online]. Available: https://doi.org/10.1016/j.electstud.2015.03.010

[10] Q. Li, C. Lin, and L. Xu, "Political investment cycles of state-owned enterprises," *The Review of Financial Studies*, vol. 33, no. 7, pp. 3088–3129, 09 2019.

[11] Łukasz Wiktor Olejnik, "Cycles in a cycle: investment expenditures and their composition during the political budgetary cycle," *Local Government Studies*, vol. 0, no. 0, pp. 1–32, 2021. [Online]. Available: https://doi.org/10.1080/03003930.2020.1851207

[12] M. Guinjoan and T. Rodon, "Let's party! the impact of local festivities on the incumbent's electoral support," *Local Government Studies*, vol. 0, no. 0, pp. 1–23, 2020. [Online]. Available: https://doi.org/10.1080/03003930.2020.1771308

[13] A. Kern and P. Amri, "Political credit cycles," *Economics & Politics*, vol. 33, no. 1, pp. 76–108, 2021. [Online]. Available: https://doi.org/10.1111/ecpo.12158

[14] G. Wenzelburger, C. Jensen, S. Lee, and C. Arndt, "How governments strategically time welfare state reform legislation: empirical evidence from five european countries," *West European Politics*, vol. 43, no. 6, pp. 1285–1314, 2020.

[15] S. Galiani, N. Hajj, P. J. McEwan, P. Ibarrarán, and N. Krishnaswamy, "Voter response to peak and end transfers: Evidence from a conditional cash transfer experiment," *American Economic Journal: Economic Policy*, vol. 11, no. 3, pp. 232–60, August 2019. [Online]. Available: https://www.aeaweb.org/articles?id=10.1257/pol.20170448

[16] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," Cryptology ePrint Archive, Report 2018/601, 2018, https://eprint.iacr.org/2018/601.

[17] J. Drake, "Minimal vdf randomness beacon," https://ethresear.ch/t/minimal-vdf-randomness-beacon/3566, 2018.

[18] D. SAMUEL and D. FRANCK, "Cpu-z oc world records. highest overclocking of all times." https://valid.x86.fr/records.php, 2020.

[19] P. McCorry, S. F. Shahandashti, and F. Hao, "A smart contract for boardroom voting with maximum voter privacy," in *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers*, 2017, pp. 357–375.

[20] M. Seifelnasr, H. S. Galal, and A. M. Youssef, "Scalable open-vote network on ethereum," in *Financial Cryptography and Data Security - FC 2020 International Workshops, AsiaUSEC, CoDeFi, VOTING, and WTSC, Kota Kinabalu, Malaysia, February 14, 2020, Revised Selected Papers*, ser. Lecture Notes in Computer Science, M. Bernhard, A. Bracciali, L. J. Camp, S. Matsuo, A. Maurushat, P. B. Rønne, and M. Sala, Eds., vol. 12063. Springer, 2020, pp. 436–450.

[21] B. Yu, J. K. Liu, A. Sakzad, S. Nepal, R. Steinfeld, P. Rimba, and M. H. Au, "Platform-independent secure blockchain-based voting system," in *International Conference on Information Security*. Springer, 2018, pp. 369–386.

[22] C. Killer, B. Rodrigues, E. J. Scheid, M. Franco, M. Eck, N. Zaugg, A. Scheitlin, and B. Stiller, "Provotum: A blockchain-based and end-to-end verifiable remote electronic voting system," in *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, 2020, pp. 172–183.

[23] S. Venugopalan, I. Homoliak, Z. Li, and P. Szalachowski, "BBB-Voting: 1-out-of-k Blockchain-Based Boardroom Voting," 2021.

[24] A. Kiayias and M. Yung, "Self-tallying elections and perfect ballot secrecy," in *Public Key Cryptography*, D. Naccache and P. Paillier, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 141–158.

[25] J. Benaloh, R. Rivest, P. Y. Ryan, P. Stark, V. Teague, and P. Vora, "End-to-end verifiability," *arXiv preprint arXiv:1504.03778*, 2015.

[26] L. Aguiar-Conraria, P. C. Magalhães, and F. J. Veiga, "Transparency, Policy Outcomes, and Incumbent Support," *Kyklos*, vol. 72, no. 3, pp. 357–380, August 2019.

[27] R. E. Carlin, T. Hellwig, G. J. Love, C. Martínez-Gallardo, and M. M. Singer, "When does the public get it right? the information environment and the accuracy of economic sentiment," *Comparative Political Studies*, vol. 0, no. 0, p. 0010414021989758, 2021.

[28] R. Matile, B. Rodrigues, E. J. Scheid, and B. Stiller, "Caiv: Cast-as-

intended verifiability in blockchain-based voting," in *ICBC 2019, Seoul, Korea (South), May 14-17, 2019*, 2019, pp. 24–28.

[29] O. Baudron, P.-A. Fouque, D. Pointcheval, J. Stern, and G. Poupard, "Practical multi-candidate election system," in *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, 2001, pp. 274–283.

[30] Y. Li, W. Susilo, G. Yang, Y. Yu, D. Liu, X. Du, and M. Guizani, "A blockchain-based self-tallying voting protocol in decentralized iot," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2020.

[31] I. Stančíková and I. Homoliak, "Sbvote: Scalable self-tallying blockchain-based voting," in *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, 2023.

[32] H. Jonker, S. Mauw, and J. Pang, "Privacy and verifiability in voting systems: Methods, developments and trends," *Comput. Sci. Rev.*, vol. 10, pp. 1–30, 2013. [Online]. Available: https://doi.org/10.1016/j.cosrev.2013.08.002

[33] K. Pietrzak, "Simple verifiable delay functions," in *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, ser. LIPIcs, A. Blum, Ed., vol. 124. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 60:1–60:15. [Online]. Available: https://doi.org/10.4230/LIPIcs.ITCS.2019.60

[34] B. Wesolowski, "Efficient verifiable delay functions," *J. Cryptol.*, vol. 33, no. 4, pp. 2113–2147, 2020. [Online]. Available: https://doi.org/10.1007/s00145-020-09364-x

[35] J. Bonneau, J. Clark, and S. Goldfeder, "On Bitcoin as a public randomness source," *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 1015, 2015. [Online]. Available: http://eprint.iacr.org/2015/1015

[36] Bitmex_Reserach, "The mystery of the bitcoin nonce pattern," https://blog.bitmex.com/the-mystery-of-the-bitcoin-nonce-pattern/, 2019.

[37] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 185–200.

[38] J. Lind, O. Naor, I. Eyal, F. Kelbert, E. G. Sirer, and P. Pietzuch, "Teechain: a secure payment network with asynchronous blockchain access," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 63–79.

[39] B. Zhang, R. Oliynykov, and H. Balogun, "A treasury system for cryptocurrencies: Enabling better collaborative intelligence," in *NDSS 2019*, 2019.

[40] G. Harewood, "Electoral reform in canada," 2002. [Online]. Available: https://www.securewebexchange.com/metamodequadrant.com/Electoral_Reform.html

[41] IDEA, "International institute for democracy and electoral assistance," 2021, https://www.idea.int/data-tools/world-view/44.

[42] D. Khader, B. Smyth, P. Y. A. Ryan, and F. Hao, "A fair and robust voting system by broadcast," in *(EVOTE 2012), July 11-14, 2012, Castle Hofen, Bregenz, Austria*, 2012, pp. 285–299. [Online]. Available: https://dl.gi.de/20.500.12116/18220

[43] Z. Zhao and T.-H. H. Chan, "How to vote privately using Bitcoin," in *International Conference on Information and Communications Security*. Springer, 2015, pp. 82–96.

[44] P. Tarasov and H. Tewari, "Internet voting using Zcash," *Cryptology ePrint Archive*, 2017.

[45] Y. Liu and Q. Wang, "An e-voting protocol based on blockchain," *Cryptology ePrint Archive*, 2017.

The authors are Sarad Venugopalan, Ivana Stančíková and Ivan Homoliak – biography not provided by the author's choice.

SmartOTPs: An Air-Gapped 2-Factor Authentication for Smart-Contract Wallets

Ivan Homoliak
FIT, Brno University of Technology
Singapore University of Technology and Design
ihomoliak@fit.vutbr.cz

Dominik Breitenbacher
FIT, Brno University of Technology
dbreitenbacher@gmail.com

Ondrej Hujnak
FIT, Brno University of Technology
ihujnak@fit.vutbr.cz

Pieter Hartel
University of Twente
pieter.hartel@utwente.nl

Alexander Binder
Singapore University of Technology and Design
alexander_binder@sutd.edu.sg

Pawel Szalachowski
Singapore University of Technology and Design
pawel@sutd.edu.sg

ABSTRACT

With the recent rise of cryptocurrencies' popularity, the security and management of crypto-tokens have become critical. We have witnessed many attacks on users and providers, which have resulted in significant financial losses. To remedy these issues, several wallet solutions have been proposed. However, these solutions often lack either essential security features, usability, or do not allow users to customize their spending rules.

In this paper, we propose SmartOTPs, a smart-contract wallet framework that gives a flexible, usable, and secure way of managing crypto-tokens in a self-sovereign fashion. The proposed framework consists of four components (i.e., an authenticator, a client, a hardware wallet, and a smart contract), and it provides 2-factor authentication (2FA) performed in two stages of interaction with the blockchain. To the best of our knowledge, our framework is the first one that utilizes one-time passwords (OTPs) in the setting of the public blockchain. In SmartOTPs, the OTPs are aggregated by a Merkle tree and hash chains whereby for each authentication only a short OTP (e.g., 16B-long) is transferred from the authenticator to the client. Such a novel setting enables us to make a fully air-gapped authenticator by utilizing small QR codes or a few mnemonic words, while additionally offering resilience against quantum cryptanalysis. We have made a proof-of-concept based on the Ethereum platform. Our cost analysis shows that the average cost of a transfer operation is comparable to existing 2FA solutions using smart contracts with multi-signatures.

ACM Reference Format:
Ivan Homoliak, Dominik Breitenbacher, Ondrej Hujnak, Pieter Hartel, Alexander Binder, and Pawel Szalachowski. 2020. SmartOTPs: An Air-Gapped 2-Factor Authentication for Smart-Contract Wallets. In *2nd ACM Conference on Advances in Financial Technologies (AFT '20), October 21–23, 2020, New York, NY, USA.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3419614.3423257

1 INTRODUCTION

The success of cryptocurrencies has surpassed all expectations resulting in various open and decentralized platforms that allow users to conduct monetary transfers, write smart contracts, and participate in predictive markets. Cryptocurrencies introduce their own crypto-tokens, which can be transferred in transactions authenticated by private keys that belong to crypto-token owners. These private keys are managed by a wallet software that gives users an interface to interact with the cryptocurrency. There are many cases of stolen keys that were secured by various means [9, 16, 18, 26]. Such cases have brought the attention of the research community to the security issues related to key management in cryptocurrencies [14, 32, 34]. According to the previous work [14, 32], there are a few categories of key management approaches.

In password-protected wallets, private keys are encrypted with selected passwords. Unfortunately, users often choose weak passwords that can be brute-forced if stolen by malware [1]; optionally, such malware may use a keylogger for capturing a passphrase [14, 65]. Another similar option is to use password-derived wallets that generate keys based on the provided password. However, they also suffer from the possibility of weak passwords [26]. Hardware wallets are a category that promises the provision of better security by introducing devices that enable only the signing of transactions, without revealing the private keys stored on the device. However, these wallets do not provide protection from an attacker with full access to the device [29, 44, 45], and more importantly, wallets that do not have a secure channel for informing the user about the details of a transaction being signed (e.g., [48]) may be exploited by malware targeting IPC mechanisms [15].

A popular option for storing private keys is to deposit them into server-side hosted (i.e., custodial) wallets and currency-exchange services [10, 20, 50, 61, 62, 66]. In contrast to the previous categories, server-side wallets imply trust in a provider, which is a potential risk of this category. Due to many cases of compromising server-side wallets [2, 9, 53, 68, 77] or fraudulent currency-exchange operators [76], client-side hosted wallets have started to proliferate. In such wallets, the main functionality, including the storage of private keys, has moved to the user side [17, 19, 21, 40, 57]; hence,

AFT '20, October 21–23, 2020, New York, NY, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8139-0/20/10…$15.00
https://doi.org/10.1145/3419614.3423257

trust in the provider is reduced but the users still depend on the provider's infrastructure.

To increase security of former wallet categories, multi-factor authentication (MFA) is often used, which enables spending crypto-tokens only when a number of secrets are used together. However, we emphasize that different security implications stem from the multi-factor authentication made *against a centralized party* (e.g., using Google Authenticator) and *against the blockchain* itself. In the former, the authentication factor is only as secure as the centralized party, while the latter provides stronger security that depends on the assumption of an honest majority of decentralized consensus nodes (i.e., miners) and security of cryptographic primitives used. Wallets from a split control category [32] provide MFA against the blockchain. This can be achieved by threshold cryptography wallets [34, 56], multi-signature wallets [6, 25, 30, 74], and state-aware smart-contract wallets [22, 72, 75]. The last class of wallets is of our concern, as spending rules and security features can be encoded in a smart contract.

Although there are several smart-contract wallets using MFA against the blockchain [22, 75], to the best of our knowledge, none of them provide an air-gapped authentication in the form of short OTPs similar to Google Authenticator.

Proposed Approach. In this paper, we propose SmartOTPs, a framework for smart-contract cryptocurrency wallets, which provides 2FA against data stored on the blockchain. The first factor is represented by the user's private key and the second factor by OTPs. To produce OTPs, the authenticator device of SmartOTPs utilizes hash-based cryptographic constructs, namely a pseudo-random function, a Merkle tree, and hash chains. We propose a novel combination of these elements that minimizes the amount of data transferred from the authenticator, which enables us to implement the authenticator in a fully air-gapped setting, not requiring any USB or another connection. SmartOTPs belongs to the category of state-aware smart contract wallets, and it provides protection against the attacker that possesses the user's private key *or* the user's authenticator *or* the attacker that tampers with the client.

Contributions. Our main contributions are as follows:

- We show that standard 2FA methods against the blockchain do not meet either the security or usability requirements for an air-gapped setting (see Section 3.2).
- We propose SmartOTPs, a smart-contract wallet framework that provides 2FA against the blockchain while using short OTPs serving as the second factor (see Section 4). OTPs are managed in a novel way, enabling us to make an authenticator device fully air-gapped.
- To increase the number of OTPs, we resolve the time-space trade-off at the client by combining hash chains with Merkle trees in a novel way (see Section 4.4).
- We implement and evaluate our approach (including hardware version of the authenticator), and we provide the source code of our solution (see Section 6).

2 BACKGROUND AND PRELIMINARIES

We assume a generic cryptocurrency of which the blocks of records are stored in an ever-growing public distributed ledger called a *blockchain*, which is by design resistant to modifications. In a blockchain, blocks are linked using a cryptographic hash function, and each new block has to be agreed upon by participants running a consensus protocol (i.e., *miners*). Each block may contain orders transferring crypto-tokens, application codes written in a platform-supported language, and the execution orders of such applications. These application codes are referred to as *smart contracts* and can encode arbitrary processing logic (e.g., agreements). Interactions between clients and the cryptocurrency system are based on messages called *transactions*, which can contain either orders transferring crypto-tokens or calls of smart contract functions. All transactions sent to a blockchain are validated by miners who replicate the state of the blockchain.

Merkle Tree. A Merkle tree is a data structure based on the binary tree in which every leaf node contains a hash of a single data block, while every non-leaf node contains a hash of its concatenated children. A Merkle tree enables efficient verification as to whether some data are associated with a leaf node by comparing the expected root hash of a tree with the one computed from a hash of the data in the query and the remaining nodes required to reconstruct the root hash (i.e., *proof* or *authentication path*). The reconstruction of the root hash has logarithmic time complexity, which makes the Merkle tree an efficient scheme for membership verification.

2.1 Notation

By the term *operation* we refer to an action with a smart-contract wallet using SmartOTPs, which may involve, for instance, a transfer of crypto-tokens or a change of daily spending limits. Then, we use the term *transfer* for the indication of transferring crypto-tokens. By $\{msg\}_{\mathbb{U}}$ we denote the message *msg* digitally signed by $\mathbb{U}$, and by $msg.\sigma$ we refer to the signature; $\mathcal{RO}$ is the random oracle; $h(.)$: stands for a cryptographic hash function; $h^i(.)$ substitutes i-times chained function $h(.)$, e.g., $h^2(.) \equiv h(h(.))$; $\|$ is the string concatenation; $h^i_{\mathcal{D}}(.)$ substitutes i-times chained function $h(.)$ with embedded domain separation, e.g., $h^2_{\mathcal{D}}(.) = h(2 \,||\, h(1 \,||\, .))$; $F_k(.) \equiv h(k \,\|\, .)$ denotes a pseudo-random function that is parametrized by a secret seed k; % represents modulo operation over integers; $\Sigma.\{KeyGen, Verify, Sign\}$ represents a signature scheme of the blockchain platform; $SK_{\mathbb{U}}$, $PK_{\mathbb{U}}$ is the private/public key-pair of $\mathbb{U}$, under Σ, and $a \mid b$ represents bitwise OR of arguments a and b.

3 PROBLEM DEFINITION

The main goal of this work is to propose a cryptocurrency wallet framework that provides a secure and usable way of managing crypto-tokens. In particular, we aim to achieve:

Self-Sovereignty: ensures that the user does not depend on the 3rd party's infrastructure, and the user does not share his secrets with anybody. Self-sovereign (i.e., non-custodial) wallets do not pose a single point of failure in contrast to server-side (i.e., custodial) wallets, which when compromised, resulted in huge financial loses [2, 9, 53, 68, 77].

Security: the insufficient security level of some self-sovereign wallets has caused significant financial losses for individuals and companies [16, 18, 26, 60]. We argue that wallets should be designed with security in mind and in particular, we point out 2FA solutions, which have successfully contributed to the security of other environments [3, 69]. Our motivation is to provide a cheap security extension of the hardware wallets (i.e., the first factor) by using OTPs as the second factor in a fashion similar to Google Authenticator.

3.1 Threat Model

For a generic cryptocurrency described in Section 2, we assume an adversary $\mathcal{A}$ whose goal is to conduct unauthorized operations on the user's behalf or render the user's wallet unusable. $\mathcal{A}$ is able to eavesdrop on the network traffic as well as to participate in the underlying consensus protocol. However, $\mathcal{A}$ is unable to take over the cryptocurrency platform nor to break the used cryptographic primitives. We further assume that $\mathcal{A}$ is able to intercept and "override" the user's transactions, e.g., by launching a man-in-the-middle (MITM) attack or by creating a conflicting malicious transaction with a higher fee, which will incentivize miners to include $\mathcal{A}$'s transaction and discard the user's one; this attack is also referred to as *transaction front-running*. We assume three types of exclusively occurring attackers, each targeting one of the three components of our framework: (1) $\mathcal{A}$ with access to the user's private key hardware wallet, (2) $\mathcal{A}$ that tampers with the client, and for completeness we also assume (3) $\mathcal{A}$ with access to the authenticator. Next, we assume that the legitimate user correctly executes the proposed protocols and $h(.)$ is an instantiation of $\mathcal{RO}$.

3.2 Design Space

There are many types of wallets with different properties. In our context, to achieve self-sovereignty we identify smart-contract wallets as a promising category. These wallets manage crypto-tokens by the functionality of smart contracts, enabling users to have customized control over their wallets. The advantages of these solutions are that spending rules can be explicitly specified and then enforced by the cryptocurrency platform itself. Therefore, using this approach, it is possible to build a flexible wallet with features such as daily spending limits or transfer limits.

General OTPs. With spending rules encoded in a smart contract, it is feasible to design custom security features, such as OTP-based authentication serving as the second factor. In such a setting, the authenticator produces OTPs to authenticate transactions in the smart contract. However, in contrast to digital signatures, OTPs do not provide non-repudiation of data present in a transaction with an OTP; moreover, they can be intercepted and misused by the front-running or the MITM attacks. To overcome this limitation, we argue that a two-stage protocol $\Pi_O^{<G>}$ must be employed, enabling secure utilization of general OTPs in the context of blockchains. In the first stage of $\Pi_O^{<G>}$, an operation O, signed by the user $\mathbb{U}$, is submitted to the blockchain, where it obtains an identifier i. Then, in the second stage, O_i is executed on the blockchain upon the submission of OTP_i that is unambiguously associated with the operation initiated in the first stage.

Requirements of General and Air-Gapped OTPs. Based on the above, we define the necessary security requirements of general OTPs used in the blockchain as follows:

(1) **Authenticity:** each OTP must be associated only with a unique authenticator instance.

(2) **Linkage:** each OTP_i must be linked with exactly a single operation O_i, ensuring that OTP_i cannot be misused for the authentication of $O_j, i \neq j$.

(3) **Independence:** OTP_i linked with the operation O_i cannot be derived from OTP_j of an operation O_j, where $i \neq j$, or an arbitrary set of other OTPs.

Nevertheless, in the air-gapped setting (important for a high usability and security), one more requirement comes into play: **the short length of OTPs**. Short OTPs allow the users to use a relatively small number of mnemonic words or a small QR code to transfer an OTP in an air-gapped fashion. This requirement is of high importance especially in the case when the authenticator is implemented as a resource-constrained embedded device with a small display (e.g., credit-card-shaped wallet, such as CoolBitX [24]).

Analysis of Existing Solutions. We argue that not all solutions meet the requirements of air-gapped OTPs. Asymmetric cryptography primitives such as digital signatures or zero-knowledge proofs are inadequate in this setting, despite meeting all general OTP requirements. State-of-the-art signature schemes with reasonable performance overhead [8, 41] and short signature size produce a 48B-64B long output. The BLS signatures [12] go even beyond the previous constructs and might produce signatures of size 32B. Nevertheless, BLS signatures are unattractive in the setting of the smart contract platforms that put high execution costs for BLS signature verification, which is ~33 times more expensive than in the case of ECDSA with the equivalent security level [13]. Hence, we assume 48B as the minimal feasible OTP size for assymetric cryptography.

However, transferring even 48B in a fully air-gapped environment by transcription of mnemonic words [59] would lack usability for regular users – considering study from Dhakal et al. [27], transcription of 36 English words takes 42s on average, which is much longer than users are willing to "sacrifice." We note that the situation is better with QR code, but on the other hand it has two limitations: (1) when the authenticator is implemented as a simple embedded device, its display might be unable to fit a requested QR code with sufficient scanning properties (to preserve the maximal scanning distance of QR code, the "denser" QR code must be displayed in a larger image [67]) and (2) occasionally, the users might not have a camera in their devices, thus, they can proceed only with a fallback method that uses mnemonics. Finally, most of the currently deployed asymmetric constructions are vulnerable to quantum computing [7].

The problem of long signatures also exists in hash-based signature constructs [28, 46, 51]. Lamport-Diffie one-time signatures (LD-OTS) [46] produce an output of length $2|h(.)|^2$, which, for example in the case of $|h(.)| = 16B$ yields $4kB$-long signatures. The signature size of LD-OTS can be reduced by using one string of one-time key for simultaneous signing of several bits in the message digest (i.e., Winternitz one-time signatures (W-OTS) [28]), but at the expense of exponentially increased number of hash computations (in the number of encoded bits) during a signature generation and verification. The extreme case minimizing the size of W-OTS to $|h(.)|$ (for simplicity omitting checksum) would require $2^{|h(.)|}$ hash computations for signature generation, which is unfeasible.

Approaches based on symmetric cryptography primitives produce much shorter outputs, but it is challenging to implement them with smart-contract wallets. Widely used one-time passwords like HOTP [54] or TOTP [55] require the user to share a secret key k with the authentication server. Then, with each authentication request the user proves that he possesses k by returning the output of an $F_k(.)$ computed with a nonce (i.e., HOTP) or the current timestamp (i.e., TOTP). This approach is insecure in the setting of the blockchain, as the user would have to share the secret k with a smart-contract wallet, making k publicly visible.

A solution that does not publicly disclose secret information and, at the same time, provides short enough OTPs (e.g., $16B \simeq 12$ mnemonic words $\simeq$ QR code v1), can be implemented by Lamport's hash chains [47] or other single hash-chain-based constructs, such as T/Key [43]. A hash chain enables the production of many OTPs by the consecutive execution of a hash function, starting from k that represents a secret key of the authenticator. Upon the initialization, a smart contract is preloaded with the last generated value $h^n(k)$. When the user wants to authenticate the ith operation, he sends the $h^{n-i}(k)$ to the smart contract in the second stage of $\Pi_O^{<G>}$. The smart contract then computes $h(.)$ consecutively i times and checks to ascertain whether the obtained value equals the stored value. However, the main drawback of this solution is that *each OTP can be trivially derived from any previous one*, and thereby this scheme does not meet the requirement of OTPs on independence. To detail an attack misusing this flaw, assume the MITM attacker possessing $SK_{\mathbb{U}}$ (i.e., the first factor) is able to initiate operations in the first stage of $\Pi_O^{<G>}$. The attacker $\mathcal{A}$ initiates operation O_i and waits for $\mathbb{U}$ to initiate and confirm an arbitrary follow-up operation $O_j, j > i$. When $\mathbb{U}$ sends OTP_j in the second stage of $\Pi_O^{<G>}$, $\mathcal{A}$ intercepts and "front-runs" the user's transaction by a malicious transaction with OTP_i computed as $h^{j-i}(OTP_j)$. Although one may argue that this scheme can be hardened by a modification denying to confirm older operations than the last initiated one, it would bring a race condition issue in which $\mathcal{A}$ might keep initiating operations in the first stage of $\Pi_O^{<G>}$ each time he intercepts a confirmation transaction from $\mathbb{U}$, causing the DoS attack on the wallet.

4 PROPOSED APPROACH

For a cryptocurrency described in Section 2, we propose SmartOTPs, a 2FA against the blockchain, which consists of: (1) a client $\mathbb{C}$, (2) a private key hardware wallet $\mathbb{W}$ equipped with a display, (3) a smart-contract $\mathbb{S}$, and (4) an air-gapped authenticator $\mathbb{A}$ that might be implemented as an embedded device with limited resources. First, we explain the key idea of our approach, which enables us to construct $\mathbb{A}$ as a fully air-gapped device. Then, we present the base version of SmartOTPs, and finally, we describe modifications.

4.1 Design of an Air-Gapped Authenticator

In our approach, OTPs are generated by a pseudo-random function $F_k(.)$ and then aggregated by a Merkle tree, providing a single value, the root hash ($\mathcal{R}$). $\mathcal{R}$ is stored at $\mathbb{S}$ and serves as a PK for OTPs. Assuming the two stage protocol $\Pi_O^{<G>}$ (further denoted as Π_O), the user $\mathbb{U}$ might confirm the initiated operation O_{opID} by a corresponding OTP_{opID} (provided by $\mathbb{A}$) in the second stage of Π_O, whereby $\mathbb{S}$ verifies the correctness of OTP_{opID} with use of $\mathcal{R}$. A challenge of such an approach is the size of an OTP.

Algorithm 1: Smart contract $\mathbb{S}$ with 2FA

```
▷ VARIABLES AND FUNCTIONS OF ENVIRONMENT:
  tx: a current transaction processed by $\mathbb{S}$,
  balance: the current balance of a contract,
  transfer(r, v): transfer v crypto-tokens from a smart contract to r,
▷ DECLARATION OF TYPES:
  Operation { addr, param, pending, type ∈ {TRANSFER, . . .} }
▷ DECLARATION OF FUNCTIONS:
function constructor(root, pk) public
    operations ← [];                                  ▷ An append-only list
    $PK_{\mathbb{U}}$ ← pk,  $\mathcal{R}$ ← root,  nextOpID ← 0;
    return $\mathbb{S}^{ID}$;                        ▷ Computed by a blockchain platform.
function initOp(a, p, type) public
    assert $\Sigma.verify(tx.\sigma, PK_{\mathbb{U}})$;     ▷ 1st factor of 2FA
    opID ← nextOpID++;
    operations[opID] ← new Operation(a, p, true, type);
function confirmOp(otp, $\pi$, opID) public
    assert operations[opID].pending;
    verifyOTP(otp, $\pi$, opID);                      ▷ 2nd factor of 2FA
    execOp(operations[opID]);
    operations[opID].pending ← false;
function verifyOTP(otp, $\pi_{opID}$, opID) private
    assert deriveRootHash(otp, $\pi_{opID}$, opID) = $\mathcal{R}$;
function execOp(oper) private
    if TRANSFER = oper.type then
        assert oper.param ≤ balance;
        transfer(oper.addr, oper.param);
```

4.1.1 ***From Straw-Man to the Base Version.***

Using the straw-man version, a 2FA requires $\mathbb{A}$ to provide an OTP and its proof. However, in such a straw-man version, the user $\mathbb{U}$ has to transfer $\frac{(S+S\times H)}{8}$ bytes from $\mathbb{A}$ each time he confirms an operation, where S represents the bit-length of an OTP as well as the output of $h(.)$, and H represents the height of a Merkle tree with N leaves; hence $H = log_2(N)$. For example, if $S = 256$ and $H = 10$, then $\mathbb{U}$ would have to transfer 352B each time he confirms an operation, which has very low usability in an air-gapped setting utilizing transcription of mnemonic words [59] (i.e., 264 words) or scanning of several QR codes (e.g., 21 QR codes v1) displayed on an embedded device with a small display. Even further reduction of S to 128 bits would not help to resolve this issue, as the amount of user transferred data would be equal to 176B $\simeq$ 132 mnemonic words $\simeq$ 11 QR codes v1.

We make the observation that it is possible to decouple providing OTPs from providing their proofs. The only data that need to be kept secret are OTPs, while any node of a Merkle tree may potentially be disclosed – no OTP can be derived from these nodes. Therefore, we propose providing OTPs by $\mathbb{A}$, while their proofs can be constructed at $\mathbb{C}$ from stored hashes of OTPs. This modification enables us to fetch the nodes of the proof from the storage of $\mathbb{C}$, while $\mathbb{U}$ has to transfer only the OTP itself from $\mathbb{A}$ when confirming an operation (i.e, $S = 128 \simeq 12$ mnemonic words by default).

4.2 Base Version

4.2.1 ***Secure Bootstrapping.***

As common in other schemes and protocols, by default, we assume a secure environment for bootstrapping protocol Π_B^S (see Figure 1 and Appendix A.5), which means that $\mathbb{C}$ is trusted and cannot be compromised during execution of Π_B^S. First, $\mathbb{A}$ generates a secret seed k, which is stored as a recovery phrase by $\mathbb{U}$. $\mathbb{W}$ generates a key-pair $SK_{\mathbb{U}}, PK_{\mathbb{U}} \leftarrow \Sigma.KeyGen()$. Next, $\mathbb{U}$ transfers k from $\mathbb{A}$ to $\mathbb{C}$ in an air-gapped manner (i.e., transcribing a few mnemonic words or scanning a QR code).

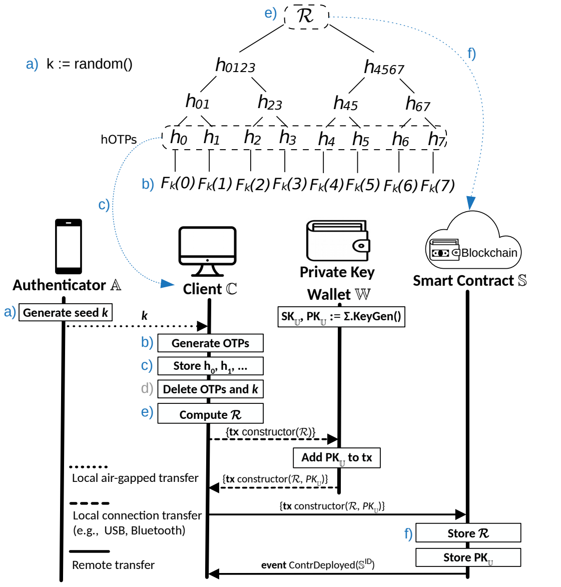

e) $\mathcal{R}$
f)
a) k := random()
h_{0123}
h_{4567}
h_{01}
h_{23}
h_{45}
h_{67}
hOTPs
h_0 h_1 h_2 h_3 h_4 h_5 h_6 h_7
b) $F_k(0)$ $F_k(1)$ $F_k(2)$ $F_k(3)$ $F_k(4)$ $F_k(5)$ $F_k(6)$ $F_k(7)$
c)
Authenticator $\mathbb{A}$
Client $\mathbb{C}$
Private Key Wallet $\mathbb{W}$
Blockchain
Smart Contract $\mathbb{S}$
a) Generate seed k
k
$SK_{\mathbb{U}}, PK_{\mathbb{U}} := \Sigma.KeyGen()$
b) Generate OTPs
c) Store $h_0, h_1, \ldots$
d) Delete OTPs and k
e) Compute $\mathcal{R}$
{tx constructor($\mathcal{R}$)}
Add $PK_{\mathbb{U}}$ to tx
{tx constructor($\mathcal{R}$, $PK_{\mathbb{U}}$)}
{tx constructor($\mathcal{R}$, $PK_{\mathbb{U}}$)}
Local air-gapped transfer
Local connection transfer (e.g., USB, Bluetooth)
Remote transfer
f) Store $\mathcal{R}$
Store $PK_{\mathbb{U}}$
event ContrDeployed($\mathbb{S}^{ID}$)

Figure 1: Bootstrapping of SmartOTPs in a secure environment (Π_B^S).

Then, $\mathbb{C}$ generates OTPs by computing $F_k(i) \mid i \in \{0, 1, \ldots, N-1\}$, where N is the number of leaves (equal to the number of OTPs in the base version). Next, $\mathbb{C}$ computes and stores the leaves of the tree – i.e., the hashes of the OTPs (i.e., *hOTPs*), which do not contain any confidential data.[1] After this step, k and the OTPs are deleted from $\mathbb{C}$, and $\mathbb{C}$ computes $\mathcal{R}$ from the stored hashes of the OTPs. Then, $\mathbb{C}$ creates a transaction containing constructor of $\mathbb{S}$ (see Algorithm 1) with $\mathcal{R}$ as the argument and passes it to $\mathbb{W}$ for appending $PK_{\mathbb{U}}$. Finally, $\mathbb{C}$ sends the transaction with the constructor to the blockchain where the deployment of $\mathbb{S}$ is made. In the constructor, $\mathcal{R}$ with $PK_{\mathbb{U}}$ are stored and ID of $\mathbb{S}$ (i.e., $\mathbb{S}^{ID}$) is assigned by a blockchain platform and returned in a response.[2] Storing $\mathcal{R}$ and $PK_{\mathbb{U}}$ binds an instance of $\mathbb{S}$ with the user's authenticator $\mathbb{A}$ and the user's private key wallet $\mathbb{W}$, respectively. In detail, $PK_{\mathbb{U}}$ enables $\mathbb{S}$ to verify whether an arbitrary transaction was signed by the user who created $\mathbb{S}$, while $\mathcal{R}$ enables the verification whether the given OTP was produced by the user's $\mathbb{A}$.

4.2.2 ***Operation Execution.*** When the wallet framework is initialized, it is ready for executing operations by a two-stage protocol Π_O (see Figure 2 and Appendix A.5):

(1) **Initialization Stage**. When $\mathbb{U}$ decides to execute an operation with SmartOTPs, he enters the details of the operation into $\mathbb{C}$ that creates a transaction calling *initOp()*, which is provided with operation-specific parameters – the type of operation (e.g., transfer), a numerical parameter (e.g., amount or daily limit), and an address parameter (e.g., recipient). Then, $\mathbb{C}$ sends this transaction to $\mathbb{W}$, which displays the details of the transaction and prompts $\mathbb{U}$ to confirm signing by a hardware button. Upon confirmation, $\mathbb{W}$ signs the transaction by $SK_{\mathbb{U}}$ and sends it back to $\mathbb{C}$. $\mathbb{C}$ forwards the transaction to $\mathbb{S}$. In the function *initOp()*, $\mathbb{S}$ verifies whether the signature was created by $\mathbb{U}$ (the first factor), stores the parameters of the operation, and then assigns a sequential ID (i.e., *opID*) to the initiated operation. In the response from $\mathbb{S}$, $\mathbb{C}$ is provided with an *opID*.

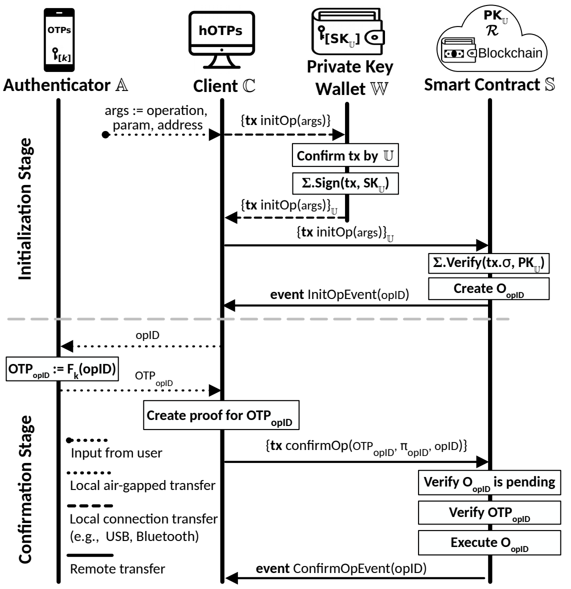

OTPs
[k]
hOTPs
[$SK_{\mathbb{U}}$]
$PK_{\mathbb{U}}$
$\mathcal{R}$
Blockchain
Authenticator $\mathbb{A}$
Client $\mathbb{C}$
Private Key Wallet $\mathbb{W}$
Smart Contract $\mathbb{S}$
Initialization Stage
args := operation, param, address
{tx initOp(args)}
Confirm tx by $\mathbb{U}$
$\Sigma.Sign(tx, SK_{\mathbb{U}})$
$\{tx\ initOp(args)\}_{\mathbb{U}}$
$\{tx\ initOp(args)\}_{\mathbb{U}}$
$\Sigma.Verify(tx.\sigma, PK_{\mathbb{U}})$
Create O_{opID}
event InitOpEvent(opID)
opID
$OTP_{opID} := F_k(opID)$
OTP_{opID}
Confirmation Stage
Create proof for OTP_{opID}
{tx confirmOp(OTP_{opID}, π_{opID}, opID)}
Input from user
Local air-gapped transfer
Local connection transfer (e.g., USB, Bluetooth)
Remote transfer
Verify O_{opID} is pending
Verify OTP_{opID}
Execute O_{opID}
event ConfirmOpEvent(opID)

Figure 2: Execution of an operation (Π_O).

(2) **Confirmation Stage**. After the transaction (that initiated the operation) is persisted on the blockchain, $\mathbb{U}$ proceeds to the second stage of Π_O. $\mathbb{U}$ enters *opID* to $\mathbb{A}$, which, in turn, computes and displays OTP_{opID} as $F_k(opID)$. Storing *hOTPs* computed from OTPs at $\mathbb{C}$ enables $\mathbb{U}$ to transfer only the displayed OTP from $\mathbb{A}$ to $\mathbb{C}$, which can be accomplished in an air-gapped manner. Considering the mnemonic implementation [59], this means an air-gapped transfer of 12 words in the case of $O = 16$B. Then, $\mathbb{C}$ computes and appends the corresponding proof π_{opID} to the OTP. The proof of the OTP is computed from stored *hOTPs* in the $\mathbb{C}$'s storage (or directly fetched from the storage if $\mathbb{C}$ stores all nodes of the Merkle tree). Next, $\mathbb{C}$ sends a transaction with OTP_{opID} and its proof π_{opID} to the blockchain, calling the function *confirmOp()* of $\mathbb{S}$, which handles the second factor. This function verifies the authenticity of the OTP (i.e., the first requirement of OTPs) and its association with the requested operation (i.e., the second requirement of OTPs), which together implies the correctness of the provided OTP.[3] In detail, upon calling the *confirmOp()* function with *opID*, OTP_{opID}, and π_{opID} as the arguments, $\mathbb{S}$ reconstructs the root hash from the provided arguments by the function *deriveRootHash()* that is presented in Appendix A.2.[4] If the reconstructed value matches the stored value $\mathcal{R}$, the operation is executed (e.g., crypto-tokens are transferred).

[1] To improve performance during provisioning of proofs, $\mathbb{C}$ might additionally store non-leaf nodes, increasing the requirement on $\mathbb{C}$'s storage 2x.

[2] Note that $\mathbb{S}^{ID}$ represents a public identification of $\mathbb{S}$, which serves as a destination for sending crypto-tokens to $\mathbb{S}$ by any party.

[3] Note that SmartOTPs meet the third requirement of OTPs by the design.

[4] Note that this algorithm contains, not yet described, improvements.

In the following, we present extensions of SmartOTPs, improving its efficiency and usability, and introducing new features.

4.3 Bootstrapping in an Insecure Environment

The main advantage of Π_B^S described above is its high usability, requiring only an air-gapped transfer of k and connected $\mathbb{W}$. However, Π_B^S is not resistant against $\mathcal{A}$ tampering with $\mathbb{C}$; $\mathcal{A}$ might intercept k or forge $\mathcal{R}$ for $\mathcal{R}'$. Similarly, $\mathcal{A}$ might forge $PK_{\mathbb{U}}$ for $PK_{\mathcal{A}}$, while staying unnoticeable for $\mathbb{U}$ who expects that $\mathbb{S}^{ID}$ obtained is correct. Therefore, we propose an alternative bootstrapping protocol Π_B^I (see Appendix A.5), assuming that $\mathcal{A}$ can tamper with $\mathbb{C}$ during bootstrapping. In this protocol, first we protect SmartOTPs from the interception of k and then from forging $\mathcal{R}$ and $PK_{\mathbb{U}}$.

To avoid the interception of k, instead of transferring k, $\mathbb{U}$ performs a transfer of all leaves of the Merkle tree (i.e., $hOTPs$) from $\mathbb{A}$ to $\mathbb{C}$, which can be achieved with a microSD card. Note that the leaves are hashes of OTPs, hence they do not contain any confidential data. Next, to protect SmartOTPs from forging of $PK_{\mathbb{U}}$ and $\mathcal{R}$, we require a deterministic computation of $\mathbb{S}^{ID}$ by a blockchain platform using $PK_{\mathbb{U}}$ and $\mathcal{R}$, hence $\mathbb{S}^{ID}$ can be computed and displayed together with $\mathcal{R}$ in $\mathbb{W}$ before the deployment of $\mathbb{S}$. In detail, $\mathbb{S}^{ID}$ is computed as $h(PK_{\mathbb{U}} \parallel \mathcal{R})$, thus each pair consisting of a public key and a root hash maps to the only $\mathbb{S}^{ID}$. However, even with this modification, $\mathcal{R}$ can still be forged by $\mathbb{C}$. Therefore, when transaction with the constructor is sent to $\mathbb{W}$, $\mathbb{U}$ has to compare $\mathcal{R}$ displayed at $\mathbb{W}$ with the one computed and displayed by $\mathbb{A}$. In the case of equality, $\mathbb{U}$ records $\mathbb{S}^{ID}$ displayed in $\mathbb{W}$.

4.4 Increasing the Number of OTPs

A small number of OTPs can have negative usability and security implications. First, users executing many transactions[5] would need to create new OTPs often, and thus change their addresses. Second, an attacker possessing $SK_{\mathbb{U}}$ can flood $\mathbb{S}$ with initialized operations, rendering all the OTPs unusable. Therefore, we need to increase the number of OTPs to make the attack unfeasible. However, increasing the number of OTPs linearly increases the amount of data that $\mathbb{C}$ needs to preserve in its storage. For example, if the number of OTPs is 2^{20}, then $\mathbb{C}$ has to store $33.6MB$ of data (considering $S = 16B$ and $\mathbb{C}$ storing all leaves), which is feasible even on storage-limited devices. However, e.g., for 2^{32} OTPs, $\mathbb{C}$ needs to store $137.4GB$ of data, which might be infeasible even on PCs, especially when $\mathbb{C}$ handles multiple instances of SmartOTPs.

To resolve this issue, we modify the base approach by applying a time-space trade-off [38] for OTPs. Namely, we introduce hash chains of which last items are aggregated by the Merkle tree. With such a construction, OTPs can be encoded as elements of chains and revealed layer by layer in the reverse order of creating the chains. This allows multiplication of the number of OTPs by the chain length without increasing the $\mathbb{C}$'s storage but imposing a larger number of hash computations on $\mathbb{S}$ and $\mathbb{A}$. Nonetheless, smart contract platforms set only a low execution cost for $h(.)$.

An illustration of this construction is presented in the bottom left part of Figure 3.[6] A hash chain of length P is built from each OTP assumed so far. Then, the last items of all hash chains are used as the first iteration layer, which provides $\frac{N}{P}$ OTPs.[7] Similarly, the penultimate items of all the hash chains are used as the second iteration layer, etc., until the last iteration layer consisting of the first items of hash chains (i.e., outputs of $F_k(.)$) has been reached (see the middle part of Figure 3). We emphasize that introducing hash chains may cause a violation of the requirement on the independence of OTPs if implemented incorrectly; i.e., OTPs from upper iteration layers can be derived from lower layers. Therefore, to enforce this requirement, we invalidate all the OTPs of all the previous iteration layers by a sliding window at $\mathbb{S}$.

Furthermore, if a hash chain were to use the same hash function throughout the entire chain, it would be vulnerable to birthday attacks [39]. To harden a hash chain against a birthday attack, a *domain separation* proposed by Leighton and Micali [49] can be used: a different hash function is applied in each step of a hash chain. Note that without domain separation, inverting the ith iterate of $h(.)$ is i times easier than inverting a single hash function (see the proof in [37]). Therefore, we use a different hash function for all but the last iteration layer $1 \leq i < P$ as follows:

$$h_{\mathcal{D}[i]}(x) \quad = \quad h(P - i + 1 \,||\, x), \tag{1}$$

where x represents the OTP from the next iteration layer.

Although domain separation hardens a single hash chain against the birthday attack, this attack is still possible within the current iteration layer, which is an inevitable consequence of using multiple hash chains. Therefore, the number of leaves $\mathcal{L}$ (i.e., N/P) is the parameter that must be considered when quantifying the security level of our scheme (see Section 5).

With this improvement, $\mathbb{A}$ is updated to provide OTPs by

$$getOTP(i) \quad = \quad h_{\mathcal{D}}^{\alpha(i)}\Bigg(F_k\Big(\beta(i)\Big)\Bigg), \tag{2}$$

where i is the operation ID, $\alpha(i)$ determines the index in a hash chain, and $\beta(i)$ determines the index in the last iteration layer of OTPs. We provide concrete expressions for $\alpha(i)$ and $\beta(i)$ in Equation 4, which involves all proposed improvements and optimizations. A derivation of $\mathcal{R}$ from the OTP at $\mathbb{S}$ needs to be updated as well (see Algorithm 6 in Appendix). In detail, $\mathbb{S}$ executes $P - \alpha(i) - 1 = \left\lfloor \frac{iP}{N} \right\rfloor$ hash computations, which is a complementary number to the number of hash computations at $\mathbb{A}$ with regard to P. Also, $\mathbb{C}$ has to be modified, requiring computation of a proof to use the leaf index relative to the current iteration layer of OTPs (i.e., $i \ \% \ \frac{N}{P}$).

With this improvement, given the number of leaves equal to 2^{20} and $P = 2^{12}$, $\mathbb{C}$ stores only $33.6MB$ of data and it has 2^{32} OTPs available. On the other hand, this modification implies, on average, the execution of additional $P/2$ hash computations at $\mathbb{S}$, imposing additional costs. However, our experiments show the benefits of this approach (see Section 6.1).

4.5 Depletion of OTPs

Even with the previous modification, the number of OTPs remains bounded, therefore they may be depleted. We propose handling of depleted OTPs by a special operation that replaces the current tree with a new one. To introduce a new tree securely, we propose updating $\mathcal{R}$ value while using the last OTP of the current tree for

[5] E.g., several smart contracts in Ethereum have over 2^{20} transactions made.

[6] Note that this figure contains further, not yet described, improvements.

[7] For simplicity, we assume that $GCD(N, P) = P$.

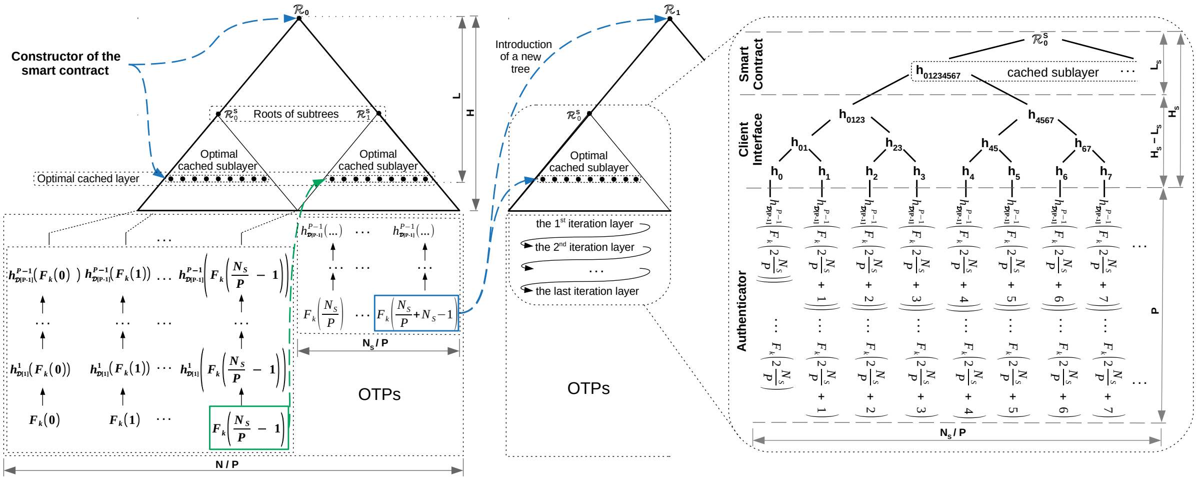


Figure 3: An overview of our approach and its improvements.

confirmation. Nevertheless, for this purpose we cannot use Π_O consisting of two stages, as $\mathcal{A}$ possessing $SK_{\mathbb{U}}$ could be "faster" than the user and might initialize the last operation and thus block all the user's funds. If we were to allow repeated initialization of this operation, then we would create a race condition issue.

To avoid this race condition issue, we propose a protocol Π_{NR} that replaces $\mathcal{R}$ during three stages of interaction with the blockchain, which requires two append-only lists L_1 and L_2 (see Algorithm 2):

(1) $\mathbb{U}$ enters OTP_{N-1} to $\mathbb{C}$. $\mathbb{C}$ sends $h(OTP_{N-1} \parallel \mathcal{R}^{new})$ to $\mathbb{S}$, which appends it to L_1.
(2) $\mathbb{C}$ sends $\mathcal{R}^{new}$ to $\mathbb{S}$, which appends it to L_2.
(3) $\mathbb{C}$ passes OTP_{N-1} with π_{N-1} to $\mathbb{S}$, where the first matching entries of L_1 and L_2 are located to perform the introduction of $\mathcal{R}^{new}$. Finally, the lists are cleared for future updates.

Locating the first entries in the lists relies on the append-only feature of lists, hence no $\mathcal{A}$ can make the first valid pair of entries in the lists. Similarly as in Π_B, we propose two variants of Π_{NR} intended for secure (i.e., Π^{S}_{NR}) and insecure environment (i.e., Π^{I}_{NR}). In Π^{I}_{NR} (see Appendix A.5), $\mathbb{A}$ must compute and display $h(OTP_{N-1} \parallel \mathcal{R}^{new})$ and $\mathcal{R}^{new}$ to enable protection against $\mathcal{A}$ that tampers with $\mathbb{C}$. Hence, $\mathbb{U}$ can verify the equality of items displayed at $\mathbb{W}$ with the ones displayed at $\mathbb{A}$ during the first and the second stage of Π^{I}_{NR}, preventing $\mathcal{A}$ from forging the tree. To adapt this improvement at $\mathbb{C}$, $\mathbb{C}$ needs to store all nodes of the new tree. Therefore, $\mathbb{U}$ provides $\mathbb{C}$ with all nodes of the new tree, transferred from $\mathbb{A}$ on a microSD card. In the case of Π^{S}_{NR}, the nodes of the new tree are transferred by a transcription of k from $\mathbb{A}$ to $\mathbb{C}$ and no values are displayed at $\mathbb{W}$ and $\mathbb{A}$ for $\mathbb{U}$'s verification.

Algorithm 2: Introduction of a new $\mathcal{R}$ in $\mathbb{S}$

```
$L_1 \leftarrow []$;                         ▷ Items have form $< h(\mathcal{R}^{new} \parallel OTP) >$
$L_2 \leftarrow []$;                         ▷ Items have form $< \mathcal{R}^{new} >$
function 1_$newRootHash$(hRootAndOTP) public
    assert $\Sigma.verify(tx.\sigma, PK_{\mathbb{U}})$;
    assert $nextOpID \% N = N - 1$;          ▷ The last oper. of tree
    $L_1$.append(hRootAndOTP);
function 2_$newRootHash$($\mathcal{R}^{new}$) public
    assert $\Sigma.verify(tx.\sigma, PK_{\mathbb{U}})$;
    assert nextOpID % $N = N - 1$;           ▷ The last oper. of tree
    $L_2$.append($\mathcal{R}^{new}$);
function 3_$newRootHash$(otp, $\pi$) public
    assert nextOpID % $N = N - 1$;           ▷ The last oper. of tree
    verifyOTP(otp, $\pi$, nextOpID);
    if $L_1.len > LEN_{MAX}$ | $L_2.len > LEN_{MAX}$ then
        $L_1, L_2 \leftarrow [], []$;
        return;                              ▷ To avoid $\mathcal{A}$ DoS-ing $\mathbb{S}$ by gas depletion.
    for {$i \leftarrow 0$; $i < L_2.len$; $i$++} do
        for {$j \leftarrow 0$; $j < L_1.len$; $j$++} do
            if $h(L_2[i] \parallel otp) = L_1[j]$ then
                $\mathcal{R} \leftarrow L_2[i]$;
                $L_1, L_2 \leftarrow [], []$;
                nextOpID++;
```

4.6 Cost & Security Optimizations

4.6.1 ***Caching in the Smart Contract.*** With a high Merkle tree, the reconstruction of $\mathcal{R}$ from a leaf node may be costly. Although the number of hash computations stemming from the Merkle tree is logarithmic in the number of leaves, the cost imposed on the blockchain platform may be significant for higher trees. We propose to reduce this cost by caching an arbitrary tree layer of depth L at $\mathbb{S}$ and do proof verifications against a cached layer. Hence, every call of *deriveRootHash()* will execute L fewer hash computations in contrast to the version that reconstructs $\mathcal{R}$, while $\mathbb{C}$ will transfer by L fewer elements in the proof.

The minimal operational cost can be achieved by directly caching leaves of the tree, which accounts only for hash computations coming from hash chains, not a Merkle tree. However, storing such a high amount of cached data on the blockchain is too expensive. Therefore, this cost optimization must be viewed as a trade-off between the depth L of the cached layer and the price required for the storage of such a cached layer on the blockchain (see Section 6.1).

We depict this modification in the left part of Figure 3, and we show that an optimal caching layer can be further partitioned into caching sublayers of subtrees (introduced later). To enable this optimization, the cached layer of the Merkle tree must be stored in the constructor of $\mathbb{S}$. From that moment, the cached layer replaces the functionality of $\mathcal{R}$, reducing the size of proofs. During the confirmation stage of Π_O, an OTP and its proof are used for the reconstruction of a particular node in the cached layer, instead of $\mathcal{R}$. Then the reconstructed value is compared with an expected node

Algorithm 3: Introduction of the next subtree at $\mathbb{S}$

```
currentSubLayer[];                                  ▷ Adjusted in the constructor
function nextSubtree(nextSubLayer, otp, $\pi_{otp}$, $\pi_{sr}$) public
    assert nextOpID % $N \neq N - 1$;                  ▷ Not the last op. of parent
    assert nextOpID % $N_S = N_S - 1$;                 ▷ The last op. of subtree
    assert currentSubLayer.len = nextSubLayer.len;
    assert deriveRootHash(otp, $\pi_{otp}$, nextOpID) = $\mathcal{R}$;
    currentSubLayer ← nextSubLayer;
    $\mathcal{R}^s$ ← reduceMT(currentSubLayer, currentSubLayer.len);
    assert subtreeConsistency($\mathcal{R}^s$, $\pi_{sr}$, $\mathcal{R}$);
    nextOpID++;                      ▷ Accounts for this introduction of a subtree
```

of the cached layer. The index of an expected node is computed as

$$idxInCache(i) \quad = \quad \left\lfloor \left(i \ \% \ \frac{N}{P}\right) \ / \ 2^{H-L} \right\rfloor, \tag{3}$$

where i is the ID of an operation.

4.6.2 ***Partitioning to Subtrees.*** The caching of the optimal layer minimizes the operational costs of SmartOTPs, but on the other hand, it requires prepayment for storing the cache on the blockchain. If the cached layer were to contain a high number of nodes, then the initial deployment cost could be prohibitively high, and moreover, the user might not deplete all the prepaid OTPs. On top of that, after revealing the first iteration layer of OTPs, the security of our scheme described so far is decreased by $log_2(N/P)$ bits due to the birthday attack (see Section 5) on OTPs. Hence, bigger trees suffer from higher security loss than smaller trees.

To overcome the prepayment issue and to mitigate the birthday attack, we propose partitioning an optimal cached layer to smaller groups having the same size, forming sublayers that belong to subtrees (see the left part of Figure 3). The obtained security loss is $log_2(N_S/P)$, $N_S \ll N$.

Starting with the deployment of $\mathbb{S}$, the cached sublayer of the first subtree and the "parent" root hash (i.e., $\mathcal{R}$) are passed to the constructor; the cached sublayer is stored on the blockchain and its consistency against $\mathcal{R}$ is verified. Then during the operational stage of Π_O, when confirmation of operation is performed, the passed OTP is verified against an expected node in the cached sublayer of the current subtree, saving costs for not doing verification against $\mathcal{R}$ (see Algorithm 5 in Appendix).

If the last OTP of the current subtree is reached, then no operation other than the introduction of the next subtree can be initialized (see the green dashed arrow in Figure 3). We propose a protocol Π_{ST} for the introduction of the next subtree (see Appendix A.5 for the detailed description). Namely, $\mathbb{C}$ introduces the next subtree in a single step by calling a function *nextSubtree()* of $\mathbb{S}$ with the arguments containing: (1) the last OTP of the current subtree $OTP_{(N_S-1)+\delta N_S}$, $\delta \in \{1, \ldots, N/N_S - 1\}$, (2) its proof π_{otp}, (3) the cached sublayer of the next subtree, and (4) the proof π_{sr} of the next subtree's root; all items but OTP are computed by $\mathbb{C}$. The pseudo-code of the next subtree introduction at $\mathbb{S}$ is shown in Algorithm 3. The current subtree's cached sublayer is replaced by the new one, which is verified by the function *subtreeConsistency*() against $\mathcal{R}$ with the use of the passed proof π_{sr} of the new subtree's root hash $\mathcal{R}^s$. Note that introducing a new subtree invalidates all initialized yet to be confirmed operations of the previous subtree.

At $\mathbb{A}$, this improvement requires accommodating the iteration over layers of hash chains in shorter periods. Hence, $\mathbb{A}$ provides OTPs by Equation 2 with the following expressions:

$$\begin{aligned} \alpha(i) = & \quad P - \left\lfloor \frac{(i \ \% \ N_S)P}{N_S} \right\rfloor - 1, \\ \beta(i) = & \quad \left\lfloor \frac{i}{N_S} \right\rfloor \frac{N_S}{P} + \left(i \ \% \ \frac{N_S}{P}\right), \end{aligned} \tag{4}$$

where i is an operation ID and N_S is the number of OTPs provided by a single subtree. We remark, that due to this optimization, the update of a new parent root $\mathcal{R}$ as well as the constructor of $\mathbb{S}$ requires, additionally to Algorithm 2 and Algorithm 1, the introduction of a cached sublayer of the first subtree (omitted here for simplicity).

5 SECURITY ANALYSIS

We analyze the security of SmartOTPs and its resilience to attacker models under the assumption of random oracle model $\mathcal{RO}$.

5.1 Security of OTPs

OTPs in our scheme are related to two cryptographic constructs: a list of hash chains and the Merkle tree aggregating their last values. In this subsection, we assume an adversary $\mathcal{A}$ who is trying to invert OTPs, and we give a concrete expressions for security of our scheme. Since we employ the hash domain separation technique [49] for hash chains, each hash execution can be seen as an execution of an independent hash function. For such a construction, Kogan et al. give the following upper bound (see Theorem 4.6 in [43]) on the advantage of $\mathcal{A}$ breaking a chain:

$$Pr[\mathcal{A} \ breaks \ a \ chain] \leq \frac{2Q + 2P + 1}{2^S}, \tag{5}$$

where Q is the number of queries that $\mathcal{A}$ can make to $h(.)$, P is the chain length, and S is the bit-length of OTPs (and the output of $h(.)$). Kogan et al. [43] proved that inverting a hash chain hardened by the domain separation imposes a loss of security equal to the factor of 2. Therefore, to make a hardened hash chain as secure as λ*-bit* $\mathcal{RO}$, it is enough to set $S = \lambda + 2$. E.g., to achieve 128-bit security, S should be equal to 130.

SmartOTPs without Subtrees. This scheme (see Section 4.6) uses a Merkle tree that aggregates $\mathcal{L} = \frac{N}{P}$ hash chains, where the chains are created independently of each other; they have the same length and the same number of OTPs. $\mathcal{A}$ can win by inverting any of the chains; hence, the probability that this scheme is secure is

$$Pr[Scheme \ is \ secure] = \left(1 - \frac{2Q + 2P + 1}{2^S}\right)^{\mathcal{L}}. \tag{6}$$

We can apply the alternative form of Bernoulli's inequality $(1 - x)^{\mathcal{L}} \geq 1 - x\mathcal{L}$, where $\mathcal{L} \geq 1$ and $0 \leq x \leq 1$ must hold. In our case, the input conditions hold since the number of hash chains is always greater than one and the probability that $\mathcal{A}$ breaks a single chain from Equation 5 fits the range of x (i.e., $0 \leq \frac{2Q+2P+1}{2^S} \leq 1$). Hence, we lower-bound the probability from Equation 6 as follows:

$$Pr[Scheme \ is \ secure] \geq 1 - \frac{\mathcal{L}(2Q + 2P + 1)}{2^S}. \tag{7}$$

Corollary 5.1. *To make SmartOTPs without partitioning into subtrees as secure as* λ*-bit* $\mathcal{RO}$*, it is enough to set* $S = \lambda + 2 + log_2(\mathcal{L})$.

For example, to achieve 128-bit security with $\mathcal{L} = 64$ and $P \geq 1$, S should be equal to 136, and thus an OTP can be transferred by one QR code v1 or 13 mnemonic words.

Full SmartOTPs. The full SmartOTPs scheme contains partitioning into subtrees, in which all leaves of the next subtree "are visible" only after depleting OTPs of the current subtree (and using OTPs from the 1st iteration layer of the next subtree). This improves the security of our scheme under the assumption that $\mathbb{C}$'s storage is not compromised by $\mathcal{A}$, which is true for $\mathcal{A}$ that possesses $PK_{\mathbb{U}}$ or $\mathbb{A}$. Therefore, we replace $\mathcal{L}$ in Equation 7 for $\mathcal{L}_{\mathcal{S}} = \frac{N_S}{P}$, $N_S \ll N$.

Corollary 5.2. *To make the full scheme of SmartOTPs as secure as λ-bit $\mathcal{RO}$, it is enough to set $S = \lambda + 2 + log_2(\mathcal{L}_{\mathcal{S}})$.*

Therefore, to achieve 128-bit security with $\mathcal{L} = \frac{N}{N_S}\mathcal{L}_{\mathcal{S}}$, $\mathcal{L}_{\mathcal{S}} = 64$, and $P \geq 1$, S should be equal to 136, and thus an OTP can be transferred by a QR code v1 or 13 mnemonic words. To achieve the same security with $\mathcal{L}_{\mathcal{S}} = 1024$, we need to set $S = 140$, and thus an OTP can be transferred in a QR code v2 or 13 mnemonic words.

5.2 The Attacker Possessing $SK_{\mathbb{U}}$

Theorem 5.3. *$\mathcal{A}$ with access to $SK_{\mathbb{U}}$ is able to initiate operations by Π_O but is unable to confirm them.*

Justification. The security of Π_O is achieved by meeting all requirements on general OTPs (see Section 3.2). In detail, the requirement on the *independence* of two different OTPs is satisfied by the definition of $F_k(.) \equiv h(k \parallel .)$, where $h(.)$ is instantiated by $\mathcal{RO}$. This is applicable when $P = 1$. However, if $P > 1$, then items in previous iteration layers of OTPs can be computed from the next ones. Therefore, to enforce this requirement, we employ an explicit invalidation of OTPs belonging to all previous iteration layers by a sliding window at $\mathbb{S}$ (see Section 4.4). The requirement on the *linkage* of each OTP_i with operation O_i is satisfied due to (1) $\mathcal{RO}$ used for instantiation of $h(.)$ and (2) by the definition of the Merkle tree, preserving the order of its aggregated leaves. By meeting these requirements, $\mathcal{A}$ is able to initiate an operation O_j in the first stage of Π_O but is unable to use an OTP_i intercepted in the second stage of Π_O to confirm O_j, where $j \neq i$. Finally, the requirement on the *authenticity* of OTPs is ensured by a random generation of k and by anchoring $\mathcal{R}$ associated with k at the constructor of $\mathbb{S}$. □

Theorem 5.4. *Assuming $\delta \in \{0, \ldots, \frac{N}{N_S} - 2\}$, $\mathcal{A}$ with access to $SK_{\mathbb{U}}$ is unable to deplete all OTPs or misuse a stolen OTP that introduces the $(\delta + 1)$th subtree by Π_{ST}.*

Justification. When all but one OTPs of the δth subtree are depleted, the last remaining operation $O_{(N_S-1)+\delta N_S}$, $\delta \in \{0, \ldots, \frac{N}{N_S} - 2\}$ is enforced by $\mathbb{S}$ to be the introduction of the next subtree. This operation is executed in a single transaction calling the function *nextSubtree*() of $\mathbb{S}$ (see Algorithm 3) requiring the corresponding $OTP_{(N_S-1)+\delta N_S}$ that is under control of $\mathbb{U}$; hence $\mathcal{A}$ cannot execute the function to proceed with a further depletion of OTPs in the $(\delta + 1)th$ subtree. If $\mathcal{A}$ were to intercept $OTP_{(N_S-1)+\delta N_S}$ during the execution of Π_{ST} by $\mathbb{U}$, he could use the intercepted OTP only for the introduction of the next valid subtree since the function *nextSubtree*() also checks a valid cached sublayer of the $(\delta + 1)$th subtree against the parent root hash $\mathcal{R}$. □

Theorem 5.5. *Assuming $\delta = \frac{N}{N_S} - 1$, $\mathcal{A}$ with access to $SK_{\mathbb{U}}$ is neither able to deplete all OTPs nor introduce a new parent tree nor render SmartOTPs unusable.*

Justification. In contrast to the adjustment of the next subtree, the situation here is more difficult to handle, since the new parent tree cannot be verified at $\mathbb{S}$ against any paramount field. If we were to use Π_O while constraining to the last initialized operation $O_{(N-1)+\eta N}$, $\eta \in \{0, 1, \ldots\}$ of the parent tree, then $\mathcal{A}$ could render SmartOTPs unusable by submitting an arbitrary $\mathcal{R}$ in *initOp*(), thus blocking all the funds of the user. If we were to allow repeated initialization of this operation, then we would create a race condition issue. Therefore, this operation needs to be handled outside of the protocol Π_O, using two unlimited append-only lists L_1 and L_2 that are manipulated in three stages of interaction with the blockchain (see Section 4.5). In the first stage, $h(\mathcal{R}^{new} \parallel OTP_{(N-1)+\eta N})$ is appended to L_1, hence $\mathcal{A}$ cannot extract the value of OTP. In the second stage, $\mathcal{R}^{new}$ is appended to L_2, and finally, in the third stage, the user reveals the OTP for confirmation of the first matching entries in both lists. Although $\mathcal{A}$ might use an intercepted OTP from the third stage for appending malicious arguments into L_1 and L_2, when he proceeds to the third stage and submits the intercepted OTP to $\mathbb{S}$, the user's entries will match as the first ones. □

5.3 The Attacker that Tampers with the Client

Theorem 5.6. *If $\mathbb{C}$ is tampered with after Π_B, $\mathbb{U}$ can detect such a situation and prevent any malicious operation from being initialized.*

Justification. If we were to assume that $\mathbb{W}$ is implemented as a software wallet (or hardware wallet without a display), then $\mathcal{A}$ tampering with $\mathbb{C}$ might also tamper with the $\mathbb{W}$'s software running on the same machine. This would in turn enable a malicious operation to be initialized and further confirmed by $\mathbb{U}$, since $\mathbb{U}$ would be presented with a legitimate data in $\mathbb{C}$ and $\mathbb{W}$, while the transactions would contain malicious data. Therefore, we require that $\mathbb{W}$ is implemented as a hardware wallet with a display, which exposes only signing capabilities, while $SK_{\mathbb{U}}$ never leaves the device (e.g., [11, 31, 42, 73]). Due to it, $\mathbb{U}$ can verify the details of a transaction being signed in $\mathbb{W}$ and confirm signing only if the details match the information shown in $\mathbb{C}$ (for Π_O) or $\mathbb{A}$ (for $\Pi_{NR}^{\mathcal{I}}$). We refer the reader to the work of Arapinis et al. [5] for the security analysis of hardware wallets with displays. □

Theorem 5.7. *If $\mathbb{C}$ is tampered with during an execution of $\Pi_B^{\mathcal{I}}$, $\mathcal{A}$ can neither intercept k nor forge $\mathcal{R}$ nor forge $PK_{\mathbb{U}}$.*

Justification. When the protocol $\Pi_B^{\mathcal{I}}$ is used, instead of an air-gapped transfer of k from $\mathbb{A}$ to $\mathbb{C}$, $\mathbb{U}$ transfers leaves of the Merkle tree by microSD card. The leaves represent hashes of OTPs in the base version or the hashes of the last items of hash chains in the full version of SmartOTPs. In both versions, the transferred data do not contain any secrets, hence $\mathcal{A}$ cannot take advantage of intercepting them. The next option that $\mathcal{A}$ may seek for is to forge $\mathcal{R}$ for $\mathcal{R}'$ and $PK_{\mathbb{U}}$ for $PK_{\mathcal{A}}$, which results in different $\mathbb{S}^{ID}$ than in the case of $\mathcal{R}$ and $PK_{\mathbb{U}}$, since $\mathbb{S}^{ID}$ is computed as $h(PK_{\mathbb{U}} \parallel \mathcal{R})$. While $PK_{\mathbb{U}}$ is stored at $\mathbb{W}$, the authenticity of $\mathcal{R}$ needs to be verified by $\mathbb{U}$ who compares displays of $\mathbb{A}$ and $\mathbb{W}$. Only in the case of equality, $\mathbb{U}$ knows that $\mathbb{S}^{ID}$ displayed in $\mathbb{W}$ maps to legitimate $PK_{\mathbb{U}}$ and $\mathcal{R}$. □

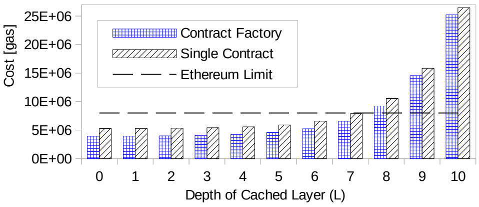


Figure 4: Deployment costs ($H = H_S$).

5.4 The Attacker Possessing the Authenticator

It is trivial to see that $\mathcal{A}$ with access to $\mathbb{A}$ is unable to initialize any operation with SmartOTPs since he does not hold $PK_{\mathbb{U}}$.

5.5 Further Properties and Implications

Requirement on Block Confirmations. Most cryptocurrencies suffer from long time to finality, potentially enabling the accidental forks, which create parallel inconsistent blockchain views. On the other hand, this issue is not present at blockchain platforms with fast finality, such as Algorand [33], HoneyBadgerBFT [52], or StrongChain [71]. In blockchains with long time to finality, overly fast confirmation of an operation may be dangerous, as, if an operation were initiated in an "incorrect" view, an attacker holding $SK_{\mathbb{U}}$ would hijack the OTP and reuse it for a malicious operation settled in the "correct" view. To prevent this threat, the recommendation is to wait for several block confirmations to ensure that an accidental fork has not happened. For example, in Ethereum, the recommended number of block confirmations to wait is 12 (i.e., ~3 minutes). Note that such waiting can be done as a background task of $\mathbb{C}$, hence $\mathbb{U}$ does not have to wait: (1) considering that $\mathcal{A}$ possesses $SK_{\mathbb{U}}$, $\mathbb{C}$ can detect such a fork during the wait and resubmit the *initOp*() transaction, (2) in the case of $\mathcal{A}$ tampering with $\mathbb{C}$, no operation can be initialized since $\mathbb{U}$ never signs $\mathcal{A}$'s transaction (due to the hardware wallet), and (3) $\mathcal{A}$ possessing $\mathbb{A}$ cannot initialize any operation as well.

Attacks with a Post Quantum Computer. Although a resilience to quantum computing (QC) is not the focus of this paper, it is of worthy to note that our scheme inherits a resilience to QC from the hash-based cryptography. The resilience of our scheme to QC is dependent on the output size of $h(.)$. A generic QC attack against $h(.)$ is Grover's algorithm [35], providing a quadratic speedup in searching for the input of the black box function. As indicated by Amy et al. [4], using this algorithm under realistic assumptions, the security of SHA-3 is reduced from 256 to 166 bits. Applying these results to OTPs with 128-bit security from examples in Section 5.1, we obtain 98-bits post-QC security. Further, when assuming the example with $\mathcal{L} = 64$ from Section 5.1 and [4], to achieve 128-bits of post-QC security, we estimate the length of OTPs to 205-bits.

6 REALIZATION IN PRACTICE

We have selected the Ethereum platform and the Solidity language for the implementation of $\mathbb{S}$, HTML/JS for DAPP of $\mathbb{C}$, Java for smartphone App of $\mathbb{A}$, and Trezor T&One [73] for $\mathbb{W}$. We selected $S = 128$ bits, which has practical advantages for an air-gapped $\mathbb{A}$, producing OTPs that are 12 mnemonic words long or a QR code v1 (with a capacity of 17B). Next, we used SHA-3 with truncated output to 128 bits as $h(.)$. We selected the size of k equal to 128 bits, fitting 12 mnemonic words ≃ 1 QR code v1.

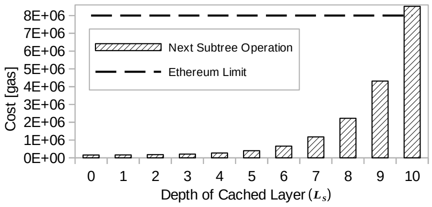


Figure 5: Cost of introducing the next subtree ($H = 20$, $H_S = 10$).

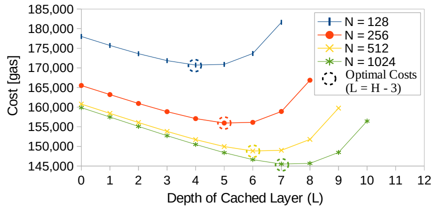


Figure 6: Average total cost per transfer ($H = H_S$).

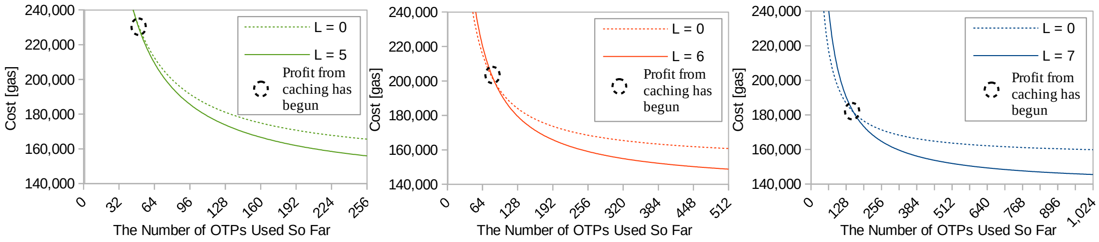
So far, we have considered only the crypto-token transfer operation. However, our proposed protocol enables us to extend the set of operations. For demonstration purposes, we extended the operation set by supporting daily limits and last resort information (see Appendix A.3). We also tested our contracts by static/dynamic analysis tools Mythril [23], Slither [70], and ContractGuard [36]; none of them detected any vulnerabilities. In addition, we made a hardware implementation of $\mathbb{A}$ using NodeMCU [58] equipped with ESP8266 (see Appendix A.6). The source code of our implementation and videos are available at https://github.com/ivan-homoliak-sutd/SmartOTPs.

6.1 Analysis of the Costs

Executing smart contracts over blockchain, i.e., performing computations and storing data, has its costs. In Ethereum Virtual Machine (EVM), these costs are expressed by the level of execution complexity of particular instructions, referred to as *gas*. One unit of gas has its market price in GWEI. In this section, we analyze the costs of our approach using the same bit-length S for $h(.)$ as well as for OTPs. S significantly influences the gas consumption for storing the cached layer on the blockchain. We remark that measured costs can also be influenced by EVM internals (e.g., 32B-long words/alignment).

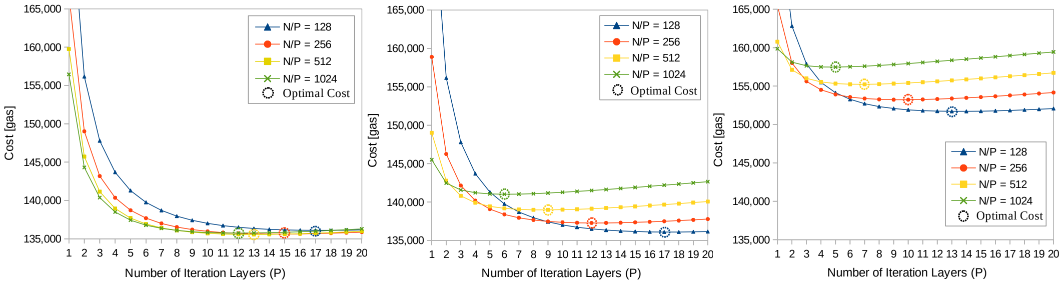
6.1.1 *Costs Related to the Merkle Tree.*

Deployment Cost. The cost of a smart contract deployment is driven mainly by the L_S (related to the first subtree) and S. A less significant factor is the consistency check of a Merkle tree, which is driven by L_S: the higher L_S is, more layers have to be reduced. Similarly, the greater $H - H_S$ is, more steps have to be done in the proof verification. On the other hand, deployment costs are independent of the length P of a hash chain; therefore, we omit the hash chain in this experiment and set $P = 1$. Further, we abstract from the concept of subtrees in order to analyze a single tree (i.e., $H = H_S$). The deployment costs of our scheme with respect to the depth L ($\equiv L_S$) of the cached layer are presented in Figure 4. The figure depicts two cases: one uses a single $\mathbb{S}$ and the second assumes a contract factory producing instances of $\mathbb{S}$. Thanks to the contract



(a) $H = 8$ **(b)** $H = 9$ **(c)** $H = 10$

Figure 7: Rolling average cost per transfer ($H = H_S$).



(a) $L = H$ **(b)** $L = 7$ **(c)** $L = 0$

Figure 8: Average total cost per transfer with regards to the length P of hash chains.

factory, we managed to save a constant amount of gas equal to $\sim 1.3M$, regardless of L_S. Since we assume $8M$ as the maximum gas limit at the Ethereum main network, we can build a caching layer with $L_S = 7$ at maximum. Later, we will see that the maximum H_S that can be used for the optimal caching layer of a subtree is $H_S = 10$, yielding 2^{10} leaves and thus $2^{10}P$ OTPs per subtree.

Cost of a Transfer. Although the cost of each operation supported by Π_O is similar, here we selected the transfer of crypto-tokens O^t, and we measured the total cost of O^t as follows:

$$\begin{aligned} O^t\_cost(L, N, P) &= \overline{cost}\Big(O^t(L, N, P)\Big) + \frac{cost\Big(O^d(L)\Big)}{N}, \\ \overline{cost}\Big(O^t(L, N, P)\Big) &= \frac{1}{N}\sum_{i=1}^{N} cost\Big(O^t_i(L, N, P)\Big), \\ cost\Big(O^t_i(L, N, P)\Big) &= cost\Big(O_i^{t.init}\Big) + cost\Big(O_i^{t.confirm}(L, N, P)\Big), \end{aligned}$$

where *cost()* measures the cost of an operation in gas units, and O_d represents the deployment operation. As the purpose of the cached layer is to reduce the number of hash computations in *confirmOp()*, the size of an optimal cached layer is subject to a trade-off between the cost of storing the cached layer on the blockchain and the savings benefit of the caching. To explore the properties of the only Merkle tree, we adjusted $H = H_S$ and $P = 1$. As each execution of O^t (i.e., O^t_i) may have a slightly different gas cost, we measured the average cost of a transaction (i.e., $\overline{cost}(O^t(L, N, P))$ for both stages of Π_O; note that the cost of $initOp() \simeq 70k$ of gas in all operations. For completeness, we present the transaction costs of all proposed operations in Appendix A.4. In Figure 6, we can see that the total average cost per transfer decreases with the increasing number of OTPs, as the deployment cost is spread across more OTPs. The optimal point depicted in the figure minimizes O^t by balancing $cost(O^d(L)))$ and $\overline{cost}(O^t(L, N, P))$. We see that $L = H - 3$ for such an optimal point. In contrast to the version without caching, this optimization has brought a cost reduction of 3.87%, 5.61%, 7.32%, and 8.92%, for 128, 256, 512, and 1,024 leaves, respectively. Next, we explored the number of transfer operations to be executed until a profit of the caching has begun (see Figure 7). We computed a rolling average cost per O^t, while distinguishing between the optimal caching layer and disabled caching – the profit from caching begins after 53, 90, and 156 transfers, respectively.

Costs with Subtrees. We measured the cost of introducing the next subtree within a parent tree depending on L_S, while we set $H = 20$ and $H_S = 10$ (see Figure 5). We found out that when subtrees (and their cached sublayers) are introduced within a dedicated operation, it is significantly cheaper compared to the introduction of a subtree during the deployment.

6.1.2 ***Costs Related to Hash Chains.*** Since each iteration layer of hash chains contributes to an average cost of $confirmOp()$ with around the same value, we measured this value on a few trees with P up to 512. Next, using this value and the deployment cost, we calculated the average total cost per transfer by adding layers of hash chains to a tree with $H = H_S$, thus increasing N by a factor of P until the minimum cost was found. As a result, the optimal caching layer shifted to the leaves of the tree (see Figure 8a), which would however, exceed the gas limit of Ethereum. To respect the gas limit, we adjusted $L = 7$, as depicted in Figure 8b. In contrast to the configurations with $L = 0$ and $P = 1$ (from Figure 6), we achieved savings of 27.80%, 19.61%, 14.95%, and 12.51% for trees with H equal to 7, 8, 9, and 10, respectively. For completeness, we calculated costs for $L = 0$ as well (see Figure 8c). Note that for $L = 0$ and $L = 7$, smaller trees are "less expensive," as they require less operations related to the proof verification in contrast to bigger trees; these operations consume substantially more gas than operations related to hash chains. Although we minimized the total cost per transfer by finding an optimal P, we highlight that increasing P contributes to the cost only minimally but on the other hand, it increases the variance of the cost. Hence, one may set this parameter even at higher values, depending on the use case.

6.1.3 ***Costs in Fiat Money.*** We assume the average exchange rate of ETH/USD equal to 211 and the "standard" gas price 5 GWEI as of May 2, 2020. For example, in the case of $N = 2^{25}$ (i.e., $H = 20$, $H_S = 10$, $P = 2^5$, $L_S = 7$), expenses per transfer operation are \$0.2, while expenses for deployment and introduction of a new subtree are \$6.90 and \$1.23, respectively.

7 RELATED WORK

In this section, we compare SmartOTPs with other hash-based approaches and other smart-contract wallets.

Hash-Based Approaches. Although Merkle signatures [51] utilize Merkle trees for aggregation of several one-time verification keys (e.g., [46]), the size of these keys and signatures is substantially larger than the size of OTPs in SmartOTPs. Even further optimization of the signature size (i.e., Winternitz OTS [28]) does not make signatures as short as in SmartOTPs. Next, we highlight that we utilize hash chains for multiplication of OTPs, which is different than their application in Winternitz OTS [28] that utilize them for the purpose of reducing the size of a single Lamport-Diffie OTS [46] by encoding multiple bits of a message digest into the number of recurrent hash computations. The next related schemes are Lamport's hash chain [47] and its modification T/Key [43] that applies the domain separation. However, since they contain only a single chain, they are not secure in the setting of the public blockchain (see Section 3.2) in contrast to SmartOTPs that never consecutively iterate OTPs within a single hash chain. Moreover, T/Key [43] is using OTPs expiring in 30s to mitigate phishing attacks, which are unrelated in our case. TESLA [63, 64] is another related scheme that utilizes a single hash-chain in a centralized setting of time-based multi-cast authentication of streamed messages.

Smart Contract Wallets. An example of the 2-of-3 multi-signature approach that only supports Trezor wallets is *TrezorMultisig2of3* [75]. A disadvantage of this solution is that $\mathbb{U}$ has to own three Trezor devices, which might be an expensive solution. The n-of-m multi-signature scheme is provided by *Gnosis Wallet* [22], which currently holds a significant amount of Ether across various smart contracts. Similar to the previous example, a disadvantage of this wallet is that $\mathbb{U}$ has to own two hardware wallets for 2FA.

The main reason why existing smart contract wallets using asymmetric cryptography are not suitable for an air-gapped authentication is due to the signature size of 64B. Hence, to input OTP, $\mathbb{U}$ has to transcribe 48 mnemonic words in the case of lacking a camera on $\mathbb{C}$, which would take ~4x longer than in the case of SmartOTPs. When $\mathbb{C}$ is equipped with a camera, $\mathbb{A}$ implemented as an embedded device might not be capable of displaying a single OTP as a small QR code since the minimal required QR code having enough data capacity is v4. Therefore, several QR codes of a lower version would be needed, which introduces additional complexity for $\mathbb{U}$.

Another drawback of asymmetric cryptography (used in these wallets) stems from its resource demands that increase the operational costs, both on $\mathbb{S}$ and $\mathbb{W}$: (1) smart contract platforms place a high execution cost for asymmetric cryptography, and (2) $\mathbb{W}$ requires more advanced MCU for cryptographic computations, while $\mathbb{A}$ from SmartOTPs requires only a secure hash function. Based on the latter, we believe that hardware realization of $\mathbb{A}$ (see Appendix A.6) in SmartOTPs is less expensive than the second hardware wallet used in multi-signature smart contracts. Moreover, we note that if SmartOTPs were to use only mnemonic words and omit QR codes, then hardware requirements of $\mathbb{A}$ (and thus the overall cost) would be even lower – mnemonic words can be displayed even on a smart-card-embedded display, such as in CoolBitX [24].

8 DISCUSSION

Vulnerability in HW Wallets. We found out that two used hardware wallets do not display all data of transactions being signed: Trezor One displays first 24B of data and Trezor T displays 35B. With regard to Ethereum transactions, this means that used wallets display only the first 8B and 19B of data representing the parameters of a contract call. Hence, $\mathcal{A}$ that tampers with $\mathbb{C}$ might purposely preserve user expected values in the displayed data while forging data that are not displayed. We reported this vulnerability to the vendor, and as a mitigation, we put the most critical parameter (i.e., address) of all concerning functions at the first displayed position.

Usability. Our approach inherits the common usability characteristics of 2FA schemes, such as an extra device to carry,[8] effort for securely storing the recovery phrase k, effort for recalling/entering passwords, and effort for a transfer of OTPs, which can be made by scanning a QR code or transcription of mnemonic words. Note that these usability implications are almost the same as in the case of existing smart contract wallets with 2FA [22, 75]. In addition to the previous, SmartOTPs requires $\mathbb{U}$ to introduce a new subtree/parent tree once in a while. Nevertheless, we envision this effort to be related only to large businesses rather than regular users; considering the example from Section 6.1.3, $\mathbb{U}$ has to introduce the next subtree after using $\sim 32K$ OTPs, while $\sim 33.5M$ OTPs are available to use before re-initialization of the parent tree. Next, we note that entering *opID* into $\mathbb{A}$ might be seen as a usability limitation, especially when N is large. However, *opID* can be reset after each iteration layer of the current subtree, thus fitting a small range (i.e., $\langle 1, \frac{N_S}{P} \rangle$).

To compare SmartOTPs with Gnosis Wallet [22], we counted the number of elementary actions (i.e., clicks, button presses, inputs of form fields, QR code scanning) required to make a transfer of funds. In the result, SmartOTPs required 33 actions while Gnosis wallet required 39 actions.

Costs. With consumption of up to $\sim 150k$ gas units per operation, our approach is comparable to equivalent 2FA solutions using smart contracts: Gnosis Wallet [22] requires $\sim 275k$ gas units[9] and TrezorMultisig2of3 [75] requires $\sim 95k$ gas units[10] per operation.

Lost Secrets. When $\mathbb{U}$ loses access to $\mathbb{A}$, he can initialize a new instance of $\mathbb{A}$ from the backup of seed k. Moreover, if $\mathbb{U}$ losses access to $\mathbb{A}$ and $\mathbb{W}$ at the same time, he can still recover the funds with the last resort functionality that we implemented (see Section 6).

[8] Assuming that the user already has a hardware wallet (the first factor).

[9] https://etherscan.io/tx/0xdb6e938... and https://etherscan.io/tx/0x328a7cc...

[10] https://etherscan.io/tx/0xfc7bbdd... (2 signatures in a single transaction).

State at the Client. The only state $\mathbb{C}$ has to store is the cache of hashes of OTPs from the first iteration layer. This might be seen as a limitation when $\mathbb{U}$ changes a client device. However, the state can be recovered anytime from the seed k or transferred by microSD card from $\mathbb{A}$ to $\mathbb{C}$ (see Section 4.2.1 and Section 4.3).

Transaction Size. Although the base version of SmartOTPs might slightly bloat the transaction due to H items of the proof, this is improved with the caching at $\mathbb{S}$, which reduces the number of items in the proof to H - L. For example, in the case of $H = 10$ and optimal caching (see Section 6.1.1) where $L = 7$, only three items of the proof are required. In this case, SmartOTPs consume 68B of transaction data (assuming 4B for operation ID), which is similar to asymmetric cryptography used in most of the blockchains.

9 CONCLUSION

In this paper, we have proposed SmartOTPs, a smart-contract wallet framework that provides a secure and usable method of managing crypto-tokens. The framework provides 2FA that is executed in two stages of interaction with the blockchain and protects against the attacker possessing a user's private key or a user's authenticator or the attacker that tampers with the client. Our framework uses OTPs constructed using a pseudo-random function, Merkle trees, and hash chains. We combine these primitives in a novel way, which enables an air-gapped setting using transcription of mnemonic words or scanning of small QR codes. Our protocol is general and can be utilized, besides the wallets, in any smart contract application for the purpose of 2FA. The provided smart contract is self-contained but its operation set can be extended by the community.

ACKNOWLEDGMENT

This work was supported by the IT4Innovations Excellence in Science project LQ1602, H2020 ECSEL project VALU3S (876852), and the internal project of Brno University of Technology (FIT-S-20-6427). This research was partially supported by ST Electronics and National Research Foundation (NRF), Prime Minister's Office Singapore, under Corporate Laboratory @ University Scheme (Programme Title: STEE Infosec – SUTD Corporate Laboratory).

REFERENCES

[1] 2015. Cryptocurrency-Stealing Malware Landscape. (2015). http://www.opensource.im/cryptocurrency/cryptocurrency-stealing-malware-landscape-dell-secureworks.php

[2] Rachel Abrams and Nathaniel Popper. 2014. Trading Site Failure Stirs Ire and Hope for Bitcoin. (2014). https://dealbook.nytimes.com/2014/02/25/trading-site-failure-stirs-ire-and-hope-for-bitcoin/

[3] Fadi Aloul, Syed Zahidi, and Wassim El-Hajj. 2009. Two factor authentication using mobile phones. In *Computer Systems and Applications, 2009. IEEE/ACS International Conference on.* IEEE, 641–644.

[4] Matthew Amy, Olivia Di Matteo, Vlad Gheorghiu, Michele Mosca, Alex Parent, and John Schanck. 2016. Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In *International Conference on Selected Areas in Cryptography.* Springer, 317–337.

[5] Myrto Arapinis, Andriana Gkaniatsou, Dimitris Karakostas, and Aggelos Kiayias. 2019. A Formal Treatment of Hardware Wallets. In *Financial Cryptography.* Springer.

[6] Armory Technologies, Inc. 2016. Bitcoin Armory. (2016). https://www.bitcoinarmory.com

[7] Daniel J Bernstein. 2009. Introduction to post-quantum cryptography. In *Post-quantum cryptography.* Springer, 1–14.

[8] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2012. High-speed high-security signatures. *Journal of Cryptographic Engineering* 2, 2 (2012), 77–89.

[9] Binance. 2019. Binance Security Breach Update. (2019). https://binance.zendesk.com/hc/en-us/articles/360028031711-Binance-Security-Breach-Update

[10] Binance.com. 2020. Binance. (2020). https://www.binance.com/

[11] BitLox. 2019. BitLox wallet. (2019). https://www.bitlox.com

[12] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short signatures from the Weil pairing. In *International Conference on the Theory and Application of Cryptology and Information Security.* Springer, 514–532.

[13] Dan Boneh, Riad S. Wahby, Sergey Gorbunov, Hoeteck Wee, and Zhenfei Zhang. 2019. RFC Internet-Draft: BLS signature. (2019). https://tools.ietf.org/html/draft-irtf-cfrg-bls-signature-00

[14] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A Kroll, and Edward W Felten. 2015. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *S&P.* IEEE, 104–121.

[15] Thanh Bui, Siddharth Prakash Rao, Markku Antikainen, Viswanathan Manihatty Bojan, and Tuomas Aura. 2018. Man-in-the-machine: exploiting ill-secured communication inside the computer. In *27th USENIX Security Symposium (USENIX Security 18).* USENIX Association, 1511–1525.

[16] Jean-Pierre Buntinx. 2016. Brain Wallets Are Not Secure and 'No One Should Use Them,' Says Study. (2016). https://news.bitcoin.com/brain-wallets-not-secure-no-one-use-says-study/

[17] CarbonWallet.com. 2019. Multi Signature Online Cryptocurrency Wallet. (2019). https://carbonwallet.com/

[18] Vincent Chia, Pieter Hartel, Qingze Hum, Sebastian Ma, Georgios Piliouras, Daniel Reijsbergen, Mark van Staalduinen, and Pawel Szalachowski. 2018. Rethinking Blockchain Security: Position Paper. In *Blockchain.*

[19] Citowise Developments. 2019. Citowise wallet. (2019). https://citowise.com/wallet

[20] coinbase. 2020. Coinbase. (2020). https://www.coinbase.com/

[21] Coinomi Ltd. 2019. Coinomi Wallet. (2019). https://coinomi.com/

[22] ConsenSys. 2019. Gnosis Wallet. (2019). https://github.com/Gnosis/MultiSigWallet

[23] ConsenSys. 2019. Mythril. (2019). https://github.com/ConsenSys/mythril

[24] CoolBitX. 2019. The CoolWallet S. (2019). https://coolwallet.io/

[25] Copay. 2019. The Secure, Shared Bitcoin Wallet. (2019). https://copay.io/

[26] Nicolas Courtois, Guangyan Song, and Ryan Castellucci. 2016. Speed optimizations in Bitcoin key recovery attacks. *Tatra Mountains Mathematical Publications* 67(1) (2016), 55–68.

[27] Vivek Dhakal, Anna Maria Feit, Per Ola Kristensson, and Antti Oulasvirta. 2018. Observations on Typing from 136 Million Keystrokes. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems.* ACM, 646–658.

[28] Chris Dods, Nigel P Smart, and Martijn Stam. 2005. Hash based digital signature schemes. In *IMA International Conference on Cryptography and Coding.* Springer, 96–115.

[29] Donjon Team. 2019. Extracting seed from Ellipal wallet. (2019). https://donjon.ledger.com/Ellipal-Security/

[30] Electrum Technologies GmbH. 2019. Electrum Bitcoin wallet. (2019). https://electrum.org/

[31] ELLIPAL. 2019. ELLIPAL Hardware Wallet 2.0. (2019). https://www.ellipal.com/

[32] Shayan Eskandari, Jeremy Clark, David Barrera, and Elizabeth Stobert. 2018. A first look at the usability of bitcoin key management. *preprint arXiv:1802.04351* (2018).

[33] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP.*

[34] Steven Goldfeder, Rosario Gennaro, Harry Kalodner, Joseph Bonneau, Joshua A Kroll, Edward W Felten, and Arvind Narayanan. 2015. Securing Bitcoin wallets via a new DSA/ECDSA threshold signature scheme. (2015).

[35] Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. In *STOC.* ACM, 212–219.

[36] GuardStrike. 2019. ContractGuard. (2019). https://contract.guardstrike.com/

[37] Johan Håstad and Mats Näslund. 2001. Practical construction and analysis of pseudo-randomness primitives. In *ASIACRYPT.* Springer, 442–459.

[38] Martin Hellman. 1980. A cryptanalytic time-memory trade-off. *IEEE transactions on Information Theory* 26, 4 (1980), 401–406.

[39] Yih-Chun Hu, Markus Jakobsson, and Adrian Perrig. 2005. Efficient constructions for one-way hash chains. In *International Conference on Applied Cryptography and Network Security.* Springer, 423–441.

[40] Infinity Blockchain Labs Europe. 2019. Infinito wallet. (2019). https://www.infinitowallet.io/

[41] Don Johnson, Alfred Menezes, and Scott Vanstone. 2001. The elliptic curve digital signature algorithm (ECDSA). *International journal of information security* 1, 1 (2001), 36–63.

[42] KeepKey. 2019. The Simple Cryptocurrency Hardware Wallet. (2019). https://www.keepkey.com/

[43] Dmitry Kogan, Nathan Manohar, and Dan Boneh. 2017. T/key: second-factor authentication from secure hash chains. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 983–999.

[44] Kraken. 2019. Inside Kraken Security Labs: Flaw Found in Keepkey Crypto Hardware Wallet. (2019). https://blog.kraken.com/post/3245/

flaw-found-in-keepkey-crypto-hardware-wallet/

[45] Kraken. 2020. Kraken Identifies Critical Flaw in Trezor Hardware Wallets. (2020). https://blog.kraken.com/post/3662/kraken-identifies-critical-flaw-in-trezor-hardware-wallets/

[46] Leslie Lamport. 1979. *Constructing digital signatures from a one-way function.* Technical Report. Technical Report CSL-98, SRI International Palo Alto.

[47] Leslie Lamport. 1981. Password authentication with insecure communication. *Commun. ACM* 24, 11 (1981), 770–772.

[48] Ledger. 2018. Ledger Nano. (2018). https://www.ledgerwallet.com/products/1-ledger-nano

[49] Frank T Leighton and Silvio Micali. 1995. Large provably fast and secure digital signature schemes based on secure hash functions. (1995). US Patent 5,432,852.

[50] Luno. 2019. Luno wallet. (2019). https://www.luno.com/wallet/

[51] Ralph C Merkle. 1989. A certified digital signature. In *Conference on the Theory and Application of Cryptology*. Springer, 218–238.

[52] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *ACM CCS*.

[53] Tyler Moore and Nicolas Christin. 2013. Beware the middleman: Empirical analysis of Bitcoin-exchange risk. In *International Conference on Financial Cryptography and Data Security*. Springer, 25–33.

[54] D M'raihi, M Bellare, F Hoornaert, D Naccache, and O Ranen. 2005. *HOTP: An HMAC-based one-time password algorithm.* Technical Report.

[55] David M'Raihi, Salah Machani, Mingliang Pei, and Johan Rydell. 2011. *Totp: Time-based one-time password algorithm.* Technical Report.

[56] Mycelium Holding LTD. 2019. Mycelium Entropy. (2019). https://mycelium.com/mycelium-entropy.html

[57] Mycelium LTD. 2019. Mycelium wallet. (2019). https://wallet.mycelium.com/

[58] NodeMcu Team. 2018. NodeMCU. (2018). https://nodemcu.readthedocs.io/en/master/

[59] Marek Palatinus, Pavol Rusnak, Aaron Voisine, and Sean Bowe. 2013. BIP-39. (2013). https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki

[60] Parity Technologies. 2017. The Multi-sig Hack: A Postmortem. (2017). https://paritytech.io/the-multi-sig-hack-a-postmortem/

[61] Paxful, Inc. 2020. Paxful. (2020). https://paxful.com/wallet

[62] Payward, Inc. 2020. Kraken. (2020). https://www.kraken.com/

[63] Adrian Perrig, Ran Canetti, J Doug Tygar, and Dawn Song. 2000. Efficient authentication and signing of multicast streams over lossy channels. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000.* IEEE, 56–73.

[64] A Perrig, D Song, R Canetti, JD Tygar, and B Briscoe. 2005. RFC4082: Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction. *Request for Comments. IETF* (2005).

[65] Antony Peyton. 2017. Cyren sounds siren over Bitcoin siphon scam. (2017). https://www.bankingtech.com/2017/01/cyren-sounds-siren-over-bitcoin-siphon-scam/

[66] Polo Digital Assets, Ltd. 2020. Poloniex. (2020). https://poloniex.com/

[67] QRStuff.com. 2011. What Size Should A Printed QR Code Be? (2011). https://blog.qrstuff.com/2011/01/18/what-size-should-a-qr-code-be

[68] Reuters. 2016. Bitcoin Worth $72M Was Stolen in Bitfinex Exchange Hack in Hong Kong. (2016). http://fortune.com/2016/08/03/bitcoin-stolen-bitfinex-hack-hong-kong/

[69] Bruce Schneier. 2005. Two-factor authentication: too little, too late. *Commun. ACM* 48, 4 (2005), 136.

[70] Slither Team. 2019. Slither. (2019). https://github.com/crytic/slither

[71] Pawel Szalachowski, Daniël Reijsbergen, Ivan Homoliak, and Siwei Sun. 2019. StrongChain: Transparent and Collaborative Proof-of-Work Consensus. In *USENIX*. 819–836.

[72] Parity Technologies. 2019. Parity Wallet. (2019). https://www.parity.io/

[73] Trezor. 2019. Trezor. (2019). https://trezor.io/

[74] TrustedCoin, LLC. 2019. TrustedCoin cosigning service. (2019). https://trustedcoin.com

[75] Unchained Capital. 2019. TrezorMultisig2of3. (2019). https://github.com/unchained-capital/ethereum-multisig

[76] Marie Vasek and Tyler Moore. 2015. There's no free lunch, even using Bitcoin: Tracking the popularity and profits of virtual currency scams. In *Financial Cryptography*. Springer, 44–61.

[77] Wolfie Zhao. 2018. Bithumb $31 Million Crypto Exchange Hack: What We Know (And Don't). (2018). https://www.coindesk.com/bithumb-exchanges-31-million-hack-know-dont-know/

A APPENDIX

A.1 Notation

We use the following notation in addition to the notation used so far: $LSB(.)$ extracts a value of the least significant bit; $a \ll b$ represents the bitwise left shift of a by b bits; a & b represents bitwise AND; $a \oplus b$ represents bitwise exclusive OR; and $h_{\mathcal{D}[a:b]}(.)$ represent $(b - a)$-times chained function $h(.)$ with embedded domain separation respecting interval $\langle a,b \rangle$, e.g., $h_{\mathcal{D}[2:3]}(.) = h(3 \parallel h(2 \parallel .))$,

Algorithm 4: Generation of OTPs of the last it. layer

```
function $generateOTPs(k, N, \eta)$
    LL_OTPs ← [];
    for $\{i \in [0, \ldots, \frac{N}{P} - 1]\}$ do
        LL_OTPs.append($F_k(\eta * \frac{N}{P} + i)$);
    return LL_OTPs;
```

Algorithm 5: A reconstruction of a node in a cached sublayer of a subtree from OTP and its proof π

```
function $deriveNodeInCache(otp, \pi, opID)$
    assert $\pi$.len = $H_S - L_S$;                              ▷ $H_S = log_2\left(\frac{N_S}{P}\right)$
    eci ← $getExpectedIdxInCache\left(opID \% \left(\frac{N_S}{P}\right)\right)$;
    assert eci = deriveIdxInCache($\pi$);
    $a \leftarrow \lfloor (opID \% N_S) * P/N_S \rfloor$;
    res ← $h^a_{\mathcal{D}[a:P]}(otp)$;                          ▷ Resolve hash chain
                                            ▷ Then resolve $\pi$
    for $\{i \leftarrow 0;\ i < \pi.len;\ i{+}{+}\}$ do
        if $1 = LSB(\pi[i])$ then
            res ← h(res ∥ $\pi$[i]);                  ▷ A node of $\pi[i]$ is on the right
        else
            res ← h($\pi$[i] ∥ res);                  ▷ A node of $\pi[i]$ is on the left
    return res;
function $deriveIdxInCache(\pi)$
    idx ← 0;
    for $\{i \leftarrow 0;\ i < H_S - L_S;\ i{+}{+}\}$ do
        if $1 = LSB(\pi[i])$ then
            idx ← idx | (1 ≪ i);
    return idx;
function $getExpectedIdxInCache(childLeafID)$
    mask ← 0xFFFFFFFF ≡ $2^{32} - 1$;                ▷ Assuming max. $H_S = 32$
    retID ← $childLeafID$;
    for $\{i \leftarrow H_S - L_S;\ i < H_S;\ i{+}{+}\}$ do
        bitToClear ← 0x01 ≪ i;
        retID ← retID & (mask ⊕ bitToClear);
    return retID;
```

Algorithm 6: A reconstruction of $\mathcal{R}$ from OTP and π

```
function $deriveRootHash(otp, \pi, opID)$
    assert $\pi$.len = H;                                    ▷ $H = log_2\left(\frac{N}{P}\right)$
    assert $opID \% \left(\frac{N}{P}\right)$ = deriveIdx($\pi$);
    $a \leftarrow \lfloor (opID \% N_S) * P/N_S \rfloor$;
    res ← $h^a_{\mathcal{D}[a:P]}(otp)$;                          ▷ Resolve hash chain
                                            ▷ Then resolve $\pi$
    for $\{i \leftarrow 0;\ i < \pi.len;\ i{+}{+}\}$ do
        if $1 = LSB(\pi[i])$ then
            res ← h(res ∥ $\pi$[i]);                  ▷ A node of $\pi[i]$ is on the right
        else
            res ← h($\pi$[i] ∥ res);                  ▷ A node of $\pi[i]$ is on the left
    return res;
function $deriveIdx(\pi)$
    idx ← 0;
    for $\{i \leftarrow 0;\ i < \pi.len;\ i{+}{+}\}$ do
        if $1 = LSB(\pi[i])$ then
            idx ← idx | (1 ≪ i);
    return idx;
```

Algorithm 7: Aggregation of OTPs

```
function aggregateOTPs(OTPs)
    hOTPs ← [];
    for {$i \in [0, \dots, OTPs.len - 1]$} do
        hOTPs[i] ← $h^P_{\mathcal{D}}(i + 1 \parallel OTPs[i])$;        ▷ Leaves of par. tree
    return reduceMT(hOTPs, hOTPs.len);

function reduceMT(hashes, length)
    if 1 = length then
        return hashes[0];
    for {$i \leftarrow 0; i \leq length/2; i$++} do
        hashes[i] ← h(hashes[2i] || hashes[2i + 1]);
    return reduceMT(hashes, length / 2);
```

A.2 Details of Algorithms and Implementation

When bootstrapping $\mathbb{C}$, OTPs of the last iteration layer are generated by Algorithm 4. Generated OTPs are then processed by hash chains, obtaining the first iteration layer of OTPs; this layer is further aggregated into $\mathcal{R}$ by Algorithm 7, which contains recursive in-situ implementation. When the OTP_{opID} is used for the authentication of the operation O_{opID}, $\mathcal{R}$ is reconstructed from the OTP and its proof π_{opID}; first, by resolving hash chains and then π_{opID} (see Algorithm 6).

A.3 Functionality Extension of the Wallet

Daily Limit. Adjusting a daily limit is a functionality that contributes primarily to $\mathbb{U}$'s self-monitoring of expenses but at the same time it avoids typos in transfers that exceeds a daily limit. This operation has the only argument representing an amount that can be spent in a single calendar day. Security implications for this operation are the same as in the case of the transfer crypto-tokens operation (see Section 5).

Last Resort Address and Timeout. As users may lose all secrets, leading to an unrecoverable state, we propose an extension that deals with such a situation based on the last resort address and timeout options. This sort of a functionality needs two dedicated operations of Π_O: one for the adjustment of the last resort address (enforced to be different than the address of $\mathbb{U}$) and another one for the adjustment of the timeout. If the timeout has elapsed, then anyone may call a dedicated function that transfers all the funds to the last resort address and destroys the contract. Note that the last resort address is enforced to be different than the address of the owner of the smart contract in order to avoid transferring all funds of the wallet to the owner's address (i.e., that might be under control of the $\mathcal{A}$) when $\mathbb{U}$ loses all secrets. Note that update of the activity is made only in the second stage of Π_O, requiring an OTP.

Operation	Stage	Mean $[gas]$	Standard Deviation $[gas]$	Sum $[gas]$
Transfer	Init. Confirm.	70,558 68,540	0 129	139,098
Set Daily Limit	Init. Confirm.	69,342 64,596	0 129	133,938
Set Last Resort Timeout	Init. Confirm.	69,342 64,982	0 474	134,324
Set Last Resort Address	Init. Confirm.	70,366 65,238	0 129	135,604
Introduction of the Next Parent Tree	Stage 1 Stage 2 Stage 3	34,223 49,459 1,082,009	- - -	1,165,691
Introduction of the Next Subtree	Depends mainly on L_S (see Figure 5)			
Send Crypto-Tokens to the Last Resort Address	-	13,887	-	13,887

Table 1: Costs of all operations ($H = 10,\ L_S = 7,\ P = 1$).

A.4 Cost of All Operations

Operational costs of all implemented operations are shown in Table 1. In the table, we do not account for deployment costs, hence we measure only instant gas consumption of the function calls. The cost measurements were obtained using configuration with the optimal cost (i.e., $L_S = H_S - 3$), $H = H_S$ and $P = 1$, which are independent of H.

A.5 Detailed Description of Protocols

Bootstrapping – protocol $\Pi^{\mathcal{S}}_B$
(for a secure environment)

- **Authenticator $\mathbb{A}$:** Generate $k \leftarrow random()$ and display k to $\mathbb{U}$.
- **Client $\mathbb{C}$:** Upon k, N, N_S, and P are entered by $\mathbb{U}$ into $\mathbb{C}$, compute $OTPs_{LL} \leftarrow F_k(\eta * \frac{N}{P} + i),\ \ i \in \{0, \dots, \frac{N}{P} - 1\},\ \ \eta \in \{0, 1, \dots\}$. Then compute and store $hOTPs \leftarrow h^P_{\mathcal{D}}(OTPs_{LL}[i]),\ \ i \in \{0, \dots, \frac{N}{P} - 1\}$ (leaves of the parent tree). Then delete $OTPs_{LL}$ and k. Then compute $\mathcal{R} \leftarrow reduceMT(hOTPs)$ by Algorithm 7, the cached sublayer *cache* of the first subtree and the proof π_{sr} of that subtree's root hash $\mathcal{R}^s$ against $\mathcal{R}$. Then create $tx_{constructor}(\mathcal{R}, cache, \pi_{sr})$ and send it to $\mathbb{W}$. Upon receiving $\{tx_{constructor}(\mathcal{R}, cache, \pi_{sr}, PK_{\mathbb{U}})\}$ from $\mathbb{W}$, forward it to $\mathbb{S}$. Upon receiving the event *ContrDeployed*($\mathbb{S}^{ID}$) from $\mathbb{S}$, update UI and inform $\mathbb{U}$ about the deployment and display $\mathbb{S}^{ID}$.
- **User $\mathbb{U}$:** Once k is generated by $\mathbb{A}$, transfer k from $\mathbb{A}$ to $\mathbb{C}$ in an air-gapped manner. Once $\mathbb{C}$ displays $\mathbb{S}^{ID}$, record $\mathbb{S}^{ID}$ as a public reference to $\mathbb{S}$.
- **Private Key Wallet $\mathbb{W}$:** Generate private/public key-pair $SK_{\mathbb{U}}$, $PK_{\mathbb{U}} \leftarrow \Sigma.KeyGen()$. Upon receiving $tx_{constructor}(\mathcal{R}, cache)$ from $\mathbb{C}$, add $PK_{\mathbb{U}}$ to this transaction and send it to $\mathbb{C}$.
- **Smart Contract $\mathbb{S}$:** Upon receiving $\{tx_{constructor}(\mathcal{R}, cache, \pi_{sr}, PK_{\mathbb{U}})\}$ from $\mathbb{C}$, deploy the code of $\mathbb{S}$ (i.e., Algorithm 1 enriched by storing of *cache*) on the blockchain, assigning $\mathbb{S}^{ID}$ to $\mathbb{S}$. During the deployment, store $\mathcal{R}$, $PK_{\mathbb{U}}$, *cache*, and adjust $nextOpID \leftarrow 0$. Next, compute root hash from *currentSubLayer* and verify its consistency against $\mathcal{R}$ using π_{sr}. Finally, send event *ContrDeployed*($\mathbb{S}^{ID}$) to $\mathbb{C}$.

Bootstrapping – protocol $\Pi^{\mathcal{I}}_B$
(for an insecure environment)

- **Authenticator** $\mathbb{A}$**:** Generate $k \leftarrow random()$. Once $\mathbb{U}$ enters N, N_S, and P to $\mathbb{A}$, compute $OTPs_{LL} \leftarrow F_k(\eta * \frac{N}{P} + i)$, $i \in \{0, \ldots, \frac{N}{P} - 1\}$. Then compute $hOTPs \leftarrow h_{\mathcal{D}}^{P}(OTPs_{LL}[i])$, $i \in \{0, \ldots, \frac{N}{P} - 1\}$ and export them to microSD card. Then compute $\mathcal{R} \leftarrow reduceMT(hOTPs)$ by Algorithm 7 and display it to $\mathbb{U}$.
- **Client** $\mathbb{C}$**:** Upon delivering $hOTPs$ by $\mathbb{U}$ to $\mathbb{C}$, store them in the local storage. Then compute $root \leftarrow reduceMT(hOTPs)$ by Algorithm 7, the cached sublayer $cache$ of the first subtree and the proof π_{sr} of that subtree's root hash $\mathcal{R}^s$. Then create $tx_{constructor}(\mathcal{R}, cache, \pi_{sr})$ and send it to $\mathbb{W}$. Upon receiving $\{tx_{constructor}(\mathcal{R}, cache, \pi_{sr}, PK_{\mathbb{U}})\}$ from $\mathbb{W}$, forward it to $\mathbb{S}$. Upon receiving event $ContrDeployed(\mathbb{S}^{ID})$ from $\mathbb{S}$, inform $\mathbb{U}$ in UI.
- **User** $\mathbb{U}$**:** Enter N, N_S, and P to $\mathbb{A}$ and $\mathbb{C}$. Upon $hOTPs$ are exported by $\mathbb{A}$ to microSD card, transfer them to $\mathbb{C}$. Upon $\mathcal{R}'$ of $\{tx_{constructor}(\mathcal{R}', cache, \pi_{sr})\}$ is displayed at $\mathbb{W}$, verify whether $\mathcal{R} = \mathcal{R}'$ by reading displays of $\mathbb{W}$ and $\mathbb{A}$. In the positive case, proceed with the deployment by pressing a hardware button of $\mathbb{W}$. Once $\mathbb{W}$ displays $\mathbb{S}^{ID}$, record it as a public reference.
- **Private Key Wallet** $\mathbb{W}$**:** Generate private/public key-pair $SK_{\mathbb{U}}$, $PK_{\mathbb{U}} \leftarrow \Sigma.KeyGen()$. Upon receiving $tx_{constructor}(\mathcal{R}', cache, \pi_{sr})$ from $\mathbb{C}$, display $\mathcal{R}'$ and $\mathbb{S}^{ID} \leftarrow h(PK_{\mathbb{U}} \parallel \mathcal{R}')$ to $\mathbb{U}$. Upon confirmation by $\mathbb{U}$, add $PK_{\mathbb{U}}$ to this transaction and send it to $\mathbb{C}$.
- **Smart Contract** $\mathbb{S}$**:** The same as in Π_B^S. The only difference in contrast to Π_B^S is the requirement of a deterministic computation of $\mathbb{S}^{ID}$ by a blockchain platform using both $PK_{\mathbb{U}}$ and $\mathcal{R}$. Hence $\mathbb{S}^{ID}$ can be computed by $\mathbb{W}$ and $\mathbb{S}$ independently.

Operation execution – protocol Π_O

- **Authenticator** $\mathbb{A}$**:** Upon receiving $opID$ from $\mathbb{U}$, compute $OTP_{opID} \leftarrow h_{\mathcal{D}}^{\alpha(opID)}(F_k(\beta(opID)))$, where $\alpha(opID)$ and $\beta(opID)$ are computed by Equation 4. Then display OTP_{opID} to $\mathbb{U}$.
- **Client** $\mathbb{C}$**:** Once $args$ are entered by $\mathbb{U}$ into $\mathbb{C}$, construct $tx_{initOp}(args)$ and send it to $\mathbb{W}$. Upon receiving $\{tx_{initOp}(args)\}_{\mathbb{U}}$ from $\mathbb{W}$, forward it to $\mathbb{S}$. Upon receiving event $InitOpEvent(opID)$ from $\mathbb{S}$, update UI and inform $\mathbb{U}$ about initialization of O_{opID}. Upon entering OTP_{opID} by $\mathbb{U}$, create proof π_{opID} from the local storage. Then create $tx_{confirmOp}(OTP_{opID}, \pi_{opID}, opID)$ and send it to $\mathbb{S}$. Upon receiving event $ConfirmOpEvent(opID)$ from $\mathbb{S}$, update UI and inform $\mathbb{U}$.
- **User** $\mathbb{U}$**:** Enter $args$ of an operation into $\mathbb{C}$. Upon $args'$ of $tx_{initOp}(args')$ are displayed at $\mathbb{W}$, verify whether $args = args'$ by reading display of $\mathbb{W}$ and UI of $\mathbb{C}$. In the positive case, confirm signing of transaction by a hardware button of $\mathbb{W}$. Once $\mathbb{C}$ informs about initialized O_{opID}, enter $opID$ into $\mathbb{A}$. Once $\mathbb{A}$ displays OTP_{opID}, transfer OTP_{opID} to $\mathbb{C}$ in an air-gapped manner.
- **Private Key Wallet** $\mathbb{W}$**:** Upon receiving $tx_{initOp}(args')$ from $\mathbb{C}$, display $args'$ to $\mathbb{U}$. Upon confirmation of $args'$ by $\mathbb{U}$, sign $tx_{initOp}(args')$ by $\Sigma.Sign(tx, SK_{\mathbb{U}})$ and send it to $\mathbb{C}$.
- **Smart Contract** $\mathbb{S}$**:** Upon receiving $\{tx_{initOp}(args)\}_{\mathbb{U}}$ from $\mathbb{C}$, verify signature $tx.\sigma$ by $\Sigma.Verify(tx.\sigma, PK_{\mathbb{U}})$. Then create a new operation O_{opID} with $opID \leftarrow nextOpID$ using $args$ and increment $nextOpID$. Then send $InitOpEvent(opID)$ to $\mathbb{C}$. Upon receiving $tx_{confirmOp}(OTP_{opID}, \pi_{opID}, opID)$ from $\mathbb{C}$, verify $O_{opID}.pending = true$. Then verify correctness of OTP_{opID} by checking $currentSubLayer[(opID \ \% \ (N_S \ / \ P)) \ / \ 2^{H_S - L_S}] = deriveNodeInCache(OTP_{opID}, \pi_{opID}, opID)$ from Algorithm 5 (or alternatively $\mathcal{R} = deriveRootHash(OTP_{opID}, \pi_{opID}, opID)$ from Algorithm 6 for the version without subtrees). Then execute O_{opID} and set $O_{opID}.pending \leftarrow false$. Finally, send $ConfirmOpEvent(opID)$ to $\mathbb{C}$.

Introduction of a new parent tree – protocol Π_{NR}^S
(for a secure environment)

- **Authenticator** $\mathbb{A}$**:** Once $\mathbb{U}$ enters $opID$ into $\mathbb{A}$, check whether $opID \ \% \ N = N - 1$, and if so, notify $\mathbb{U}$ that a new parent tree is being introduced and display k to $\mathbb{U}$. Then compute $OTP_{opID} \leftarrow h^{\alpha(opID)}(F_k(\beta(opID)))$. Next, compute $OTPs_{LL} \leftarrow F_k(\eta \frac{N}{P} + i)$, $i \in \{0, \ldots, \frac{N}{P} - 1\}$, where $\eta \leftarrow \eta + 1$. Then compute $\mathcal{R}^{new} \leftarrow aggregateOTPs(OTPs_{LL})$ by Algorithm 7 and $hRootAndOTP \leftarrow h(\mathcal{R}^{new} \parallel OTP_{opID})$. Finally, show $\mathcal{R}^{new}$ and $hRootAndOTP$ to $\mathbb{U}$.
- **Client** $\mathbb{C}$**:** •[*Stage I*] $\mathbb{C}$ notifies $\mathbb{U}$ that a new parent tree needs to be introduced and displays $opID = N - 1 + \eta N$, $\eta \in \{0, 1, \ldots\}$. Once $\mathbb{U}$ enters k into $\mathbb{C}$, compute $OTP_{N-1} \leftarrow h^{\alpha(opID)}(F_k(\beta(opID)))$, where $\alpha(opID)$ and $\beta(opID)$ are computed by Equation 4. Then create proof π_{opID} from the local storage. Then compute $OTPs_{LL} \leftarrow F_k(\eta \frac{N}{P} + i)$, $i \in \{0, \ldots, \frac{N}{P} - 1\}$, where $\eta \leftarrow \eta + 1$. Then compute and store $hOTPs \leftarrow h_{\mathcal{D}}^{P}(OTPs_{LL}[i])$, $i \in \{0, \ldots, \frac{N}{P} - 1\}$. Then delete $OTPs_{LL}$ and k. Then compute $\mathcal{R}^{new} \leftarrow reduceMT(hOTPs)$ by Algorithm 7. Then compute $hRootAndOTP \leftarrow h(\mathcal{R}^{new} \parallel OTP_{opID})$, construct $tx_{1\_newRootHash}(hRootAndOTP)$, and send it to $\mathbb{W}$. Upon receiving $\{tx_{1\_newRootHash}(hRootAndOTP)\}_{\mathbb{U}}$ from $\mathbb{W}$, forward it to $\mathbb{S}$. •[*Stage II*] Upon $newRootHash1(hRootAndOTP)$ event is received from $\mathbb{S}$, construct $tx_{2\_newRootHash}(\mathcal{R}^{new})$ and send it to $\mathbb{W}$. Once $\{tx_{2\_newRootHash}(\mathcal{R}^{new})\}_{\mathbb{U}}$ is received from $\mathbb{W}$, forward it to $\mathbb{S}$. •[*Stage III*] Upon receiving the event $newRootHash2(\mathcal{R}^{new})$ from $\mathbb{S}$, compute the cached sublayer cs of the first subtree in the new parent tree and the proof π_{sr} of the subtree's root hash $\mathcal{R}^s$. Then construct $tx_{3\_newRootHash}(OTP_{opID}, \pi_{opID}, cs, \pi_{sr})$ and send it to $\mathbb{S}$. Upon receiving event $newRootHash3(OTP_{opID})$ from $\mathbb{S}$, update UI and inform $\mathbb{U}$.
- **User** $\mathbb{U}$**:** Once $\mathbb{C}$ displays $opID = N - 1 + \eta N$, $\eta \in \{0, 1, \ldots\}$ and informs $\mathbb{U}$ about the necessity of introducing a new parent tree, enter $opID$ into $\mathbb{A}$. Once

$\mathbb{A}$ displays k and shows a message that a new parent tree is being introduced, transfer k from $\mathbb{A}$ to $\mathbb{C}$ in an air-gapped manner. Once $hRootAndOTP'$ of $\{tx_{1\_newRootHash}(hRootAndOTP')\}$ is displayed at $\mathbb{W}$, verify $hRootAndOTP' = hRootAndOTP$ by reading displays of $\mathbb{W}$ and $\mathbb{A}$. Once $\mathcal{R}^{new'}$ of $\{tx_{2\_newRootHash}(\mathcal{R}^{new'})\}$ is displayed at $\mathbb{W}$, verify $\mathcal{R}^{new'} = \mathcal{R}^{new}$ by reading displays of $\mathbb{W}$ and $\mathbb{A}$. If so, confirm signing by a hardware button of $\mathbb{W}$.

- **Private Key Wallet** $\mathbb{W}$**:** The same as in Π_O.
- **Smart Contract** $\mathbb{S}$**:** •[*Stage I*] Upon receiving $\{tx_{1\_newRootHash}(hRootAndOTP)\}_{\mathbb{U}}$ from $\mathbb{C}$, verify signature $tx.\sigma$ by $\Sigma.Verify(tx.\sigma,\ PK_{\mathbb{U}})$. Then verify $nextOpID\ \%\ N = N-1$; if so, append $hRootAndOTP$ into L_1. Then send event $newRootHash1(hRootAndOTP)$ to $\mathbb{C}$. •[*Stage II*] Upon receiving $\{tx_{2\_newRootHash}(\mathcal{R}^{new})\}_{\mathbb{U}}$ from $\mathbb{C}$, verify signature $tx.\sigma$ by $\Sigma.Verify(tx.\sigma,\ PK_{\mathbb{U}})$. Then verify $nextOpID\ \%\ N = N-1$; if so, append $\mathcal{R}^{new}$ into L_2. Then send event $newRootHash2(\mathcal{R}^{new})$ to $\mathbb{C}$. •[*Stage III*] Once $\{tx_{3\_newRootHash}(OTP_{opID},\ \pi_{opID},\ cs, \pi_{sr})\}$ is received from $\mathbb{C}$, verify $nextOpID\ \%\ N = N-1$. Then verify correctness of OTP_{opID} by checking $currentSubLayer[(opID\ \%\ (N_S\ /\ P))\ /\ 2^{H_S-L_S}] = deriveNodeInCache(OTP_{opID},\ \pi_{opID},\ opID)$ from Algorithm 5 (or $\mathcal{R} = deriveRootHash(OTP_{opID},\ \pi_{opID},\ opID)$ from Algorithm 6 in the version without subtrees). Then locate the first entries of L_1 and L_2 that match the condition $h(L_2[i]\ \|\ OTP_{opID}) = L_1[j]$. If matching entries are found, then set $\mathcal{R} \leftarrow L_2[i]$, increment $nextOpID$, adjust $currentSubLayer \leftarrow cs$ and verify its consistency by $subtreeConsistency(\mathcal{R}^s,\ \pi_{sr},\ \mathcal{R})$, where $\mathcal{R}^s \leftarrow reduceMT(currentSubLayer, currentSubLayer.len)$. Finally, clear the lists $L_1, L_2 \leftarrow [], []$.

Introduction of a new parent tree – protocol $\Pi^{\mathcal{I}}_{NR}$
(for an insecure environment)

- **Authenticator** $\mathbb{A}$**:** Once $\mathbb{U}$ enters $opID$ into $\mathbb{A}$, check whether $opID\ \%\ N = N-1$, and if so display $OTP_{opID} \leftarrow h^{\alpha(opID)}(F_k(\beta(opID)))$ and notify $\mathbb{U}$ that a new parent tree is being introduced. Then, compute $OTPs_{LL} \leftarrow F_k(\eta\frac{N}{P} + i),\ i \in \{0, \ldots, \frac{N}{P} - 1\}$, where $\eta \leftarrow \eta + 1$. Then compute $hOTPs \leftarrow h^P(OTPs_{LL}[i]),\ i \in \{0, \ldots, \frac{N}{P} - 1\}$ and export it to a microSD card. Finally, compute $\mathcal{R}^{new} \leftarrow reduceMT(hOTPs)$ by Algorithm 7 and $hRootAndOTP \leftarrow h(\mathcal{R}^{new}\ \|\ OTP_{opID})$, and display both to $\mathbb{U}$.
- **Client** $\mathbb{C}$**:** [*Stage I*] $\mathbb{C}$ notifies $\mathbb{U}$ that a new parent tree needs to be introduced and displays $opID = N - 1 + \eta N,\ \eta \in \{0, 1, \ldots\}$. Upon entering OTP_{opID} by $\mathbb{U}$, create proof π_{opID} from the local storage. Once leaves of the tree $hOTPs$ are delivered by $\mathbb{U}$ into $\mathbb{C}$, store $hOTPs$ in the local storage. Then compute $\mathcal{R}^{new} \leftarrow reduceMT(hOTPs)$ by Algorithm 7. Then compute $hRootAndOTP \leftarrow h(\mathcal{R}^{new}\ \|\ OTP_{opID})$, construct $tx_{1\_newRootHash}(hRootAndOTP)$, and send it to $\mathbb{W}$. Upon receiving $\{tx_{1\_newRootHash}(hRootAndOTP)\}_{\mathbb{U}}$ from $\mathbb{W}$, forward it to $\mathbb{S}$. [*Stages II*] and [*Stage III*] are the same as in $\Pi^{\mathcal{S}}_{NR}$
- **User** $\mathbb{U}$**:** Once $\mathbb{C}$ displays $opID = N - 1 + \eta N,\ \eta \in \{0, 1, \ldots\}$ and informs $\mathbb{U}$ about necessity of introducing a new parent tree, enter $opID$ into $\mathbb{A}$. Once $\mathbb{A}$ displays OTP_{opID} and notify $\mathbb{U}$ that the new parent tree is being introduced, transfer OTP_{opID} to $\mathbb{C}$ in an air-gapped manner. Once $hOTPs$ are exported by $\mathbb{A}$ to microSD card, transfer them to $\mathbb{C}$. Once $hRootAndOTP'$ of $\{tx_{1\_newRootHash}(hRootAndOTP')\}$ is displayed at $\mathbb{W}$, verify $hRootAndOTP' = hRootAndOTP$ by reading displays of $\mathbb{W}$ and $\mathbb{A}$; in the positive case, confirm signing of transaction within $\mathbb{W}$ by a hardware button. Once $\mathcal{R}^{new'}$ of $\{tx_{2\_newRootHash}(\mathcal{R}^{new'})\}$ is displayed at $\mathbb{W}$, verify $\mathcal{R}^{new'} = \mathcal{R}^{new}$ by reading displays of $\mathbb{W}$ and $\mathbb{A}$; in the positive case, confirm signing of transaction within $\mathbb{W}$ by a hardware button.
- **Private Key Wallet** $\mathbb{W}$**:** The same as in Π_O.
- **Smart Contract** $\mathbb{S}$**:** The same as in Π_{NR_S}.

Introduction of the next subtree – protocol Π_{ST}

- **Authenticator** $\mathbb{A}$**:** The same as in Π_O, while in addition, $\mathbb{A}$ displays a message that the next tree is being introduced.
- **Client** $\mathbb{C}$**:** $\mathbb{C}$ notifies $\mathbb{U}$ that a new subtree needs to be introduced and displays $opID = (N_S - 1) + \delta N_S,\ \delta \in \{0,\ \ldots,\ \frac{N}{N_S} - 2\}$. Upon entering OTP_{opID} by $\mathbb{U}$, create the proof π_{opID} from the local storage. Then compute $nextSubLayer$ (i.e., the cached sublayer of the next subtree) and π_{sr} (i.e., the proof of the next subtree's root) from $\mathbb{C}$'s storage. Next construct $tx_{nextSubtree}(nextSubLayer,\ OTP_{opID},\ \pi_{opID},\ \pi_{sr})$ and send it to $\mathbb{S}$. Upon receiving event $newSubtree(opID)$ from $\mathbb{S}$, update UI and inform $\mathbb{U}$.
- **User** $\mathbb{U}$**:** Once $\mathbb{C}$ displays $opID = (N_S - 1) + \delta N_S,\ \delta \in \{0,\ \ldots,\ \frac{N}{N_S} - 2\}$ and informs $\mathbb{U}$ about necessity of introducing the next subtree, enter $opID$ into $\mathbb{A}$. Once $\mathbb{A}$ displays OTP_{opID} and confirming that the next tree is being introduced, transfer it to $\mathbb{C}$ in an air-gapped manner.
- **Private Key Wallet** $\mathbb{W}$**:** No interaction required.
- **Smart Contract** $\mathbb{S}$**:** Upon receiving $tx_{nextSubtree}(nextSubLayer,\ OTP_{opID},\ \pi_{opID},\ \pi_{sr})$ from $\mathbb{C}$, verify $nextOpID\ \%\ N \neq N - 1 \wedge nextOpID\ \%\ N_S = N_S - 1 \wedge currentSubLayer.len = nextSubLayer.len$.

Then verify correctness of OTP_{opID} by checking $cache[(opID\ \%\ (N_S\ /\ P))\ /\ 2^{H_S-L_S}] = deriveNodeInCache(OTP_{opID},\ \pi_{opID},\ opID)$ from Algorithm 5. Next, update the current cached sublayer $currentSubLayer \leftarrow nextSubLayer$ and check its consistency against $\mathcal{R}$ by a function $subtreeConsistency(\mathcal{R}^s,\ \pi_{sr},\ \mathcal{R})$. Note that this function requires already computed $\mathcal{R}$ of the next subtree using Algorithm 7: $\mathcal{R}^s \leftarrow reduceMT(currentSubLayer,\ \ currentSubLayer.len)$ and its proof π_{sr}. Finally, increment $nextOpID$ and send event $newSubtree(opID)$ to $\mathbb{C}$.

A.6 Hardware Implementation of $\mathbb{A}$

For demonstration purposes, we constructed a proof-of-concept hardware implementation of the authenticator of SmartOTPs using cheap hardware and C language. In detail, we selected NodeMCU with ESP8266 MCU that costs around $2. Next, we selected a 0.96" OLED display with resolution of 128x64 that was connected to MCU by 4-wire SPI interface (the cost of such a display falls below $2). To control the display, we used Adafruit_SSD1306 a Adafruit_GFX libraries. Finally, we used a simple 4x4 keyboard with 3+4 wires addressing a combination of 3 columns and 4 rows (the cost of the keyboard is around $0.5). To interact with the keyboard, we utilized Keypad library of Adruino that is built for matrix style keyboards. Further, we used software version of hash function, available from https://github.com/ethereum/ethash. The scheme of our hardware implementation is depicted in Figure 9 and the source with a demonstration video is provided at https://github.com/ivan-homoliak-sutd/SmartOTPs.

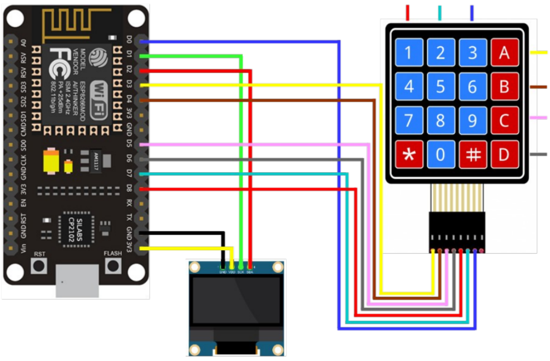


Figure 9: The scheme of a proof-of-concept hardware implementation of the authenticator.

Aquareum: A Centralized Ledger Enhanced with Blockchain and Trusted Computing

Ivan Homoliak,[†‡] Pawel Szalachowski[†]
[†]Singapore University of Technology and Design
[‡]Brno University of Technology

ABSTRACT

Distributed ledger systems (i.e., blockchains) have received a lot of attention recently. They promise to enable mutually untrusted participants to execute transactions, while providing the immutability of the transaction history and censorship resistance. Although decentralized ledgers may become a disruptive innovation, as of today, they suffer from scalability, privacy, or governance issues. Therefore, they are inapplicable for many important use cases, where interestingly, centralized ledger systems quietly gain adoption and find new use cases. Unfortunately, centralized ledgers have also several drawbacks, like a lack of efficient verifiability or a higher risk of censorship and equivocation.

In this paper, we present Aquareum, a novel framework for centralized ledgers removing their main limitations. By combining a trusted execution environment with a public blockchain platform, Aquareum provides publicly verifiable, non-equivocating, censorship-evident, private, and high-performance ledgers. Aquareum ledgers are integrated with a Turing-complete virtual machine, allowing arbitrary transaction processing logics, including tokens or client-specified smart contracts. Aquareum is fully implemented and deployment-ready, even with currently existing technologies.

1 INTRODUCTION

Ledger systems are append-only databases providing immutability (i.e., tamper resistance) as a core property. To facilitate their append-only feature, cryptographic constructions, such as hash chains or hash trees, are usually deployed. Traditionally, public ledger systems are centralized, controlled by a single entity that acts as a trusted party. In such a setting, ledgers are being deployed in various applications, including payments, logging, timestamping services, repositories, or public logs of various artifacts (e.g., keys [6, 33], certificates issued by authorities [29], and binaries [17]).

Although being successfully deployed and envisioned for multiple novel use cases, centralized ledgers have some fundamental limitations due to their centralization. Firstly, they lack efficient verifiability, which would ensure their clients that the ledger is indeed append-only and internally consistent (i.e., does not contain conflicting transactions). A naive solution is just to publish the ledger or share it with parties interested in auditing it, which, however, may be inefficient or stand against the ledger operator's deployment models (e.g., the privacy of the clients conducting financial transactions can be violated). Second, it is challenging to provide non-equivocation to centralized systems [31]. In simple yet devastating fork attacks, a ledger operator creates two conflicting copies of the ledger and presents it to different clients. Although the forked ledgers are internally consistent, the "global" view of the database is equivocated, thus completely undermining the security of the entire system. Finally, centralized systems are inherently prone to censorship. A ledger operator can refuse any request or a transaction at her will without leaving any evidence of censoring. This may be risky especially when a censored client may suffer from some consequences (e.g., fines when being unable to settle a transaction on time) or in the case when the operator wishes to hide some ledger content (e.g., data proving her misbehavior). On the other hand, recently emerged public distributed ledgers combine an append-only cryptographic data structure with a consensus algorithm, spreading trust across all participating consensus nodes. These systems are by design publicly verifiable, non-equivocating, and censorship resistant. However, they offer a low throughput, they are expensive in deployment, they do not inherently provide privacy, and their public nature makes their governance difficult and unacceptable for many applications.

Proposed Approach. In this paper, we propose Aquareum, a framework for centralized ledgers mitigating their main limitations. Aquareum employs trusted execution environment (TEE) and a public smart contract platform (i.e., built on a blockchain) to provide verifiability, non-equivocation, and to mitigate censorship. In Aquareum, a ledger operator deploys a pre-defined TEE enclave code, which verifies the consistency and correctness of the ledger for every ledger update. Then, a proof produced by the enclave is published utilizing an existing public smart contract platform, guaranteeing that the given snapshot of the ledger is verified and no alternative snapshot of this ledger exists. Furthermore, whenever a client suspects that her query (or transaction) is censored, she can (confidentially) request a resolution of the query via the smart contract platform. The ledger operator noticing the query is obligated to handle it. She passes the query to the enclave that creates a public proof of query resolution and publishes it using the smart contract platform. With such a censorship-evident design, an operator is publicly visible when misbehaving, thus the clients can take appropriate actions (e.g., suing the operator) or encode some automated service-level agreements into their smart contracts.

Aquareum can be adjusted to different ledgers and use cases, but we implemented and deployed it with minimized Ethereum Virtual Machine (EVM) since EVM provides a Turing-complete execution environment and it is widely adopted in the community of decentralized applications. Aquareum enables hosting and execution of arbitrary ledger applications, such as key:value databases, tokens, or client-defined smart contracts, while preserving the same enclave code for verification. Aquareum is fully implemented and we show that it is practical and efficient, even when built using the current technologies and tools.

2 BACKGROUND

2.1 Blockchain and Smart Contracts

A blockchain (a.k.a., a distributed ledger) is an append-only data structure that is resistant by design against modifications combined with a consensus protocol. In a blockchain, blocks containing data records are linked using a cryptographic hash function, and each new block has to be agreed upon by participants running a consensus protocol (i.e., *consensus nodes*). Each block may contain data records representing orders that transfer crypto-tokens, application codes written in a platform-supported language, and the execution orders of such application codes. These application codes are referred to as *smart contracts*, and they encode arbitrary processing logic (e.g., agreements) written in a supported language of a smart contract platform. Interactions between clients and the smart contract platform are based on messages called *transactions*, which can contain either orders transferring crypto-tokens or calls of smart contract functions. All transactions sent to a blockchain are validated by consensus nodes who maintain a replicated state of the blockchain. To incentivize consensus nodes, blockchain platforms introduce reward and fee schemes.

2.2 Trusted Execution Environment

Trusted Execution Environment (TEE) is a hardware-based component that can securely execute arbitrary code in an isolated environment. TEE uses cryptography primitives and hardware-embedded secrets that protect data confidentiality and the integrity of computations. In particular, the adversary model of TEE usually includes privileged applications and an operating system, which may compromise unprivileged user-space applications. There are several practical instances of TEE, such as Intel Software Guard Extensions (SGX) [1, 23, 32] available at Intel's CPUs or based on RISC-V architecture such as Keystone-enclave [15] and Sanctum [11]. In the context of this work, we built on top of Intel SGX, therefore we adopt the terminology introduced by it.

Intel SGX is a set of instructions that ensures hardware-level isolation of protected user-space codes called *enclaves*. An enclave process cannot execute system calls but can read and write memory outside the enclave. Thus isolated execution in SGX may be viewed as an ideal model in which a process is guaranteed to be executed correctly with ideal confidentiality, while it might run on a potentially malicious operating system.

Intel SGX allows a local process or a remote system to securely communicate with the enclave as well as execute verification of the integrity of the enclave's code. When an enclave is created, the CPU outputs a report of its initial state, also referred to as a *measurement*, which is signed by the private key of TEE and encrypted by a public key of Intel Attestation Service (IAS). The hardware-protected signature serves as the proof that the measured code is running in an SGX-protected enclave, while the encryption by IAS public key ensures that the SGX-equipped CPU is genuine and was manufactured by Intel. This proof is also known as a *quote* or *attestation*, and it can be verified by a local process or by a remote system. The enclave process-provided public key can be used by a verifier to establish a secure channel with the enclave or to verify the signature during the attestation. We assume that a trustworthy measurement of the enclave's code is available for any client that wishes to verify an attestation.

2.3 Merkle Tree

A Merkle tree [34] is a data structure based on the binary tree in which each leaf node contains a hash of a single data block, while each non-leaf node contains a hash of its concatenated children. At the top of a Merkle tree is the root hash, which provides a tamper-evident summary of the contents. A Merkle tree enables efficient verification as to whether some data are associated with a leaf node by comparing the expected root hash of a tree with the one computed from a hash of the data in the query and the remaining nodes required to reconstruct the root hash (i.e., *proof* or *authentication path*). The reconstruction of the root hash has the logarithmic time and space complexity, which makes the Merkle tree an efficient scheme for membership verification. To provide a membership verification of element x_i in the list of elements $X = \{x_i\}, i \geq 1$, the Merkle tree supports the following operations:

$\mathbf{MkRoot(X)} \rightarrow \mathbf{Root}$: an aggregation of all elements of the list X by a Merkle tree, providing a single value *Root*.

$\mathbf{MkProof(x_i, X)} \rightarrow \pi^{\mathbf{mk}}$: a Merkle proof generation for the ith element x_i present in the list of all elements X.

$\pi^{\mathbf{mk}}.\mathbf{Verify(x_i, Root)} \rightarrow \{\mathbf{True}, \mathbf{False}\}$: verification of the Merkle proof π^{mk}, witnessing that x_i is included in the list X that is aggregated by the Merkle tree with the root hash *Root*.

2.4 History Tree

A Merkle tree has been primarily used for proving membership. However, Crosby and Wallach [12] extended its application for an append-only tamper-evident log, denoted as a *history tree*. A history tree is the Merkle tree, in which leaf nodes are added in an append-only fashion, and which allows to produce logarithmic proofs witnessing that arbitrary two versions of the tree are consistent (i.e., one version of the tree is an extension of another). Therefore, once added, a leaf node cannot be modified or removed.

A history tree brings a versioned computation of hashes over the Merkle tree, enabling to prove that different versions (i.e., commitments) of a log, with distinct root hashes, make consistent claims about the past. To provide a tamper-evident history system [12], the log represented by the history tree L supports the following operations:

$\mathbf{L.add(x)} \rightarrow \mathbf{C_j}$**:** appending of the record x to L, returning a new commitment C_j that represents the most recent value of the root hash of the history tree.

$\mathbf{L.IncProof(C_i, C_j)} \rightarrow \pi^{\mathbf{inc}}$**:** an incremental proof generation between two commitments C_i and C_j, where $i \leq j$.

$\mathbf{L.MemProof(i, C_j)} \rightarrow \pi^{\mathbf{mem}}$**:** a membership proof generation for the record x_i from the commitment C_j, where $i \leq j$.

$\pi^{\mathbf{inc}}.\mathbf{Verify(C_i, C_j)} \rightarrow \{\mathbf{True}, \mathbf{False}\}$**:** verification of the incremental proof π^{inc}, witnessing that the commitment C_j contains the same history of records $x_k, k \in \{0, \ldots, i\}$ as the commitment C_i, where $i \leq j$.

$\pi^{\mathbf{mem}}.\mathbf{Verify(i, x_i, C_j)} \rightarrow \{\mathbf{True}, \mathbf{False}\}$**:** verification of the membership proof π^{mem}, witnessing that x_i is the ith record in the jth version of L, fixed by the commitment C_j, $i \leq j$.

$\pi^{\mathbf{inc}}$**.DeriveNewRoot() → $\mathbf{C_j}$:** a reconstruction of the commitment C_j from the incremental proof π^{inc} that was generated by $L.IncProof(C_i, C_j)$.

$\pi^{\mathbf{inc}}$**.DeriveOldRoot() → $\mathbf{C_i}$:** a reconstruction of the commitment C_i from the incremental proof π^{inc} that was generated by $L.IncProof(C_i, C_j)$.

2.5 Radix and Merkle-Patricia Tries

Radix trie serves as a key-value storage. In the Radix trie, every node at the l-th layer of the trie has the form of $\langle(p_0, p_1, \ldots, p_n), v\rangle$, where v is a stored value and all $p_i,\ i \in \{0, 1, \ldots, n\}$ represent the pointers on the nodes in the next (lower) layer $l + 1$ of the trie, which is selected by following the $(l + 1)$-th item of the key. Note that key consists of an arbitrary number of items that belong to an alphabet with n symbols (e.g., hex symbols). Hence, each node of the Radix trie has n children and to access a leaf node (i.e., data v), one must descend the trie starting from the root node while following the items of the key one-by-one. Note that Radix trie requires underlying database of key-value storage that maps pointers to nodes. However, Radix trie does not contain integrity protection, and when its key is too long (e.g., hash value), the Radix trie will be sparse, thus imposing a high overhead for storage of all the nodes on the path from the root to values.

Merkle Patricia Trie (MPT) [40, 49] is a combination of the Merkle tree (see Section 2.3) and Radix trie data structures, and similar the Radix Trie, it serves as a key-value data storage. However, in contrast to Radix trie, the pointers are replaced by a cryptographically secure hash of the data in nodes, providing integrity protection. In detail, MPT guarantees integrity by using a cryptographically secure hash of the value for the MPT key as well as for the realization of keys in the underlying database that maps the hashes of nodes to their content; therefore, the hash of the root node of the MPT represents an integrity snapshot of the whole MPT trie. Next, Merkle-Patricia trie introduces the *extension nodes*, due to which, there is no need to keep a dedicated node for each item of the path in the key. The MPT trie T supports the following operations:

T.root → Root: accessing the hash of the root node of MPT, which is stored as a key in the underlying database.

T.add(k, x) → Root: adding the value x with the key k to T while obtaining the new hash value of the root node.

T.get(k) → {x, ⊥}: fetching a value x that corresponds to key k; return ⊥ if no such value exists.

T.delete(k) → {True, False}: deleting the entry with key equal to k, returning $True$ upon success, $False$ otherwise.

T.MptProof(k) → $\{\pi^{\mathbf{mpt}}, \pi^{\overline{\mathbf{mpt}}}\}$**:** a MPT (inclusion / exclusion) proof generation for the entry with key k.

$\pi^{\mathbf{mpt}}$**.Verify(k, Root) → {True, False}:** verification of the MPT proof π^{mpt}, witnessing that entry with the key k is in the MPT whose hash of the root node is equal to $Root$.

$\pi^{\overline{\mathbf{mpt}}}$**.VerifyNeg(k, Root) → {True, False}:** verification of the negative MPT proof, witnessing that entry with the key k is not in the MPT with the root hash equal to $Root$.

2.6 Notation

The notation used throughout the paper is presented in the following. By $\{msg\}_{\mathbb{U}}$, we denote the message msg digitally signed by $\mathbb{U}$, and by $msg.\sigma$ we refer to a signature; $h(.)$ stands for a cryptographic hash function; $\|$ is the string concatenation; % represents modulo operation over integers; $\Sigma_p.\{KeyGen, Verify, Sign\}$ represents a signature (and encryption) scheme of the platform p, where $p \in \{pb, tee\}$ (i.e., public blockchain platform and trusted execution environment platform); and $SK^p_{\mathbb{U}}, PK^p_{\mathbb{U}}$ is the private/public key-pair of $\mathbb{U}$, under Σ_p. Then, we use π^s for denoting proofs of various data structures $s \in \{mk, mem, inc\}$: π^{mk} denotes the inclusion proof in the Merkle tree, π^{mem} and π^{inc} denote the membership proof and the incremental proof in the history tree, respectively.

3 SYSTEM MODEL AND OVERVIEW

3.1 System Model

In Aquareum, an *operator* is an entity that maintains and manages a ledger containing chronologically sorted transactions. *Clients* interact with the ledger by sending requests, such as queries and transactions to be handled. We assume that all involved parties can interact with a blockchain platform supporting smart contracts (e.g., Ethereum). Next, we assume that the operator has access to a TEE platform (e.g., Intel SGX). Finally, we assume that the operator can be malicious and her goals are as follows:

Violation of the ledger's integrity by creating its internal inconsistent state – e.g., via inserting two conflicting transactions or by removing/modifying existing transactions.

Equivocation of the ledger by presenting at least two inconsistent views of the ledger to (at least) two distinct clients who would accept such views as valid.

Censorship of client queries without leaving any audit trails evincing the censorship occurrence.

Next, we assume that the adversary cannot undermine the cryptographic primitives used, the underlying blockchain platform, and the TEE platform deployed.

3.2 Desired Properties

We target the following security properties for Aquareum ledgers:

Verifiability: clients should be able to obtain easily verifiable evidence that the ledger they interact with is internally *correct* and *consistent*. In particular, it means that none of the previously inserted transaction was neither modified nor deleted, and there are no conflicting transactions. Traditionally, the verifiability is achieved by replicating the ledger (like in blockchains) or by trusted auditors who download the full copy of the ledger and sequentially validate it. However, this property should be provided even if the operator does not wish to share the full database with third parties. Besides, the system should be *self-auditable*, such that any client can easily verify (and prove to others) that some transaction is included in the ledger, and she can prove the state of the ledger at the given point in time.

Non-Equivocation: the system should protect from forking attacks and thus guarantee that no concurrent (equivocating) versions of the ledger exist at any point in time. The consequence of this property is that whenever a client interacts with the ledger or relies on the ledger's logged artifacts, the client is ensured that other clients have ledger views consistent with her view.

Censorship Evidence: preventing censorship in a centralized system is particularly challenging, as its operator can simply pretend unavailability in order to censor undesired queries or transactions. However, this property requires that whenever the operator censors client's requests, the client can do a resolution of an arbitrary (i.e., censored) request publicly. We emphasize that proving censorship is a non-trivial task since it is difficult to distinguish "pretended" unavailability from "genuine" one. Genuine censorship evidence enables clients to enforce potential service-level agreements with the operator, either by a legal dispute or by automated rules encoded in smart contracts.

Besides those properties, we intend the system to provide *privacy* (keeping the clients' communication confidential), *efficiency* and *high performance*, not introducing any significant overhead, *deployability* with today's technologies and infrastructures, as well as *flexibility* enabling various applications and scenarios.

3.3 High-Level Overview

Aquareum ledger is initialized by an operator ($\mathbb{O}$) who creates an internal ledger (L) that will store all transactions processed and the state that they render. Initially, L contains an empty transaction set and a null state. During the initialization, $\mathbb{O}$ creates a TEE enclave ($\mathbb{E}$) whose role is to execute updates of L and verify consistency of L before each update. Initialization of $\mathbb{E}$ involves the generation of two public private key pairs – one for the signature scheme of TEE (i.e., $PK_{\mathbb{E}}^{tee}, SK_{\mathbb{E}}^{tee}$) and one for the signature scheme of the public blockchain (i.e., $PK_{\mathbb{E}}^{pb}, SK_{\mathbb{E}}^{pb}$).[1] The code of $\mathbb{E}$ is publicly-known (see Algorithm 1 and Algorithm 6), and it can be remotely attested with the TEE infrastructure by any client.

Next, $\mathbb{O}$ generates her public-private key pair (i.e., $PK_{\mathbb{O}}, SK_{\mathbb{O}}$) and deploys a special smart contract ($\mathbb{S}$) initialized with the empty L represented by its hash *LHash*, the operator's public key $PK_{\mathbb{O}}$, and both enclave public keys $PK_{\mathbb{E}}^{tee}$ and $PK_{\mathbb{E}}^{pb}$. After the deployment of $\mathbb{S}$, an instance of L is uniquely identified by the address of $\mathbb{S}$. A client ($\mathbb{C}$) wishing to interact with L obtains the address of $\mathbb{S}$ and performs the remote attestation of $\mathbb{E}$ using the $PK_{\mathbb{E}}^{tee}$.

Whenever $\mathbb{C}$ sends a transaction to $\mathbb{O}$ (see Figure 1), $\mathbb{E}$ validates whether it is authentic and non-conflicting; and if so, $\mathbb{E}$ updates L with the transaction, yielding the new version of L. The $\mathbb{C}$ is responded with a *receipt* and *"a version transition of L"*, both signed by $\mathbb{E}$, which prove that the transaction was processed successfully and is included in the new version of L. For efficiency reasons, transactions are processed in batches that are referred to as *blocks*. In detail, $\mathbb{O}$ starts the update procedure of L (see Figure 1) as follows:

a) $\mathbb{O}$ sends all received transactions since the previous update to $\mathbb{E}$, together with the current partial state of L and a small subset of L's data ∂L_i, such that $h(\partial L_i) = h(L_i)$, which is required to validate L's consistency and perform its incremental extension.
b) $\mathbb{E}$ validates and executes the transactions in its virtual machine, updates the current partial state and partial data of L, and finally creates a blockchain transaction[2] $\{h(\partial L_i), h(\partial L_{i+1})\}_{\mathbb{E}}$ signed by $SK_{\mathbb{E}}^{pb}$, which represents a version transition of the ledger from version i to its new version $i+1$, also referred to as the **version transition pair**.
c) The blockchain transaction with version transition pair is returned to $\mathbb{O}$, who sends this transaction to $\mathbb{S}$.
d) $\mathbb{S}$ accepts the second item of the version transition pair as the current hash of L iff it is signed by $SK_{\mathbb{E}}^{pb}$ and the current hash of L stored by $\mathbb{S}$ (i.e., *LHash*) is equal to the first item of the pair.

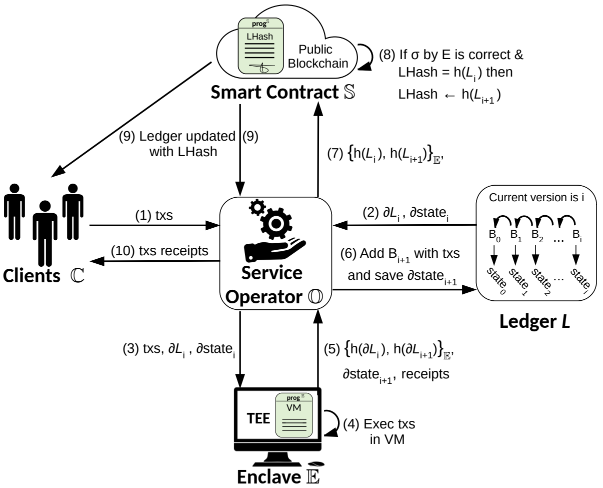

prog$^{\mathbb{S}}$
LHash
Public Blockchain
Smart Contract $\mathbb{S}$
(8) If σ by E is correct & LHash = h(L_i) then LHash ← h(L_{i+1})
(9) Ledger updated with LHash
(9)
(7) $\{h(L_i), h(L_{i+1})\}_{\mathbb{E}}$,
Clients $\mathbb{C}$
(1) txs
(10) txs receipts
Service Operator $\mathbb{O}$
(2) ∂L_i, ∂state$_i$
(6) Add B$_{i+1}$ with txs and save ∂state$_{i+1}$
Current version is i
B$_0$ B$_1$ B$_2$ ... B$_i$
state$_0$ state$_1$ state$_2$ ... state$_i$
Ledger L
(3) txs, ∂L_i, ∂state$_i$
(5) $\{h(\partial L_i), h(\partial L_{i+1})\}_{\mathbb{E}}$, ∂state$_{i+1}$, receipts
TEE
prog$^{\mathbb{E}}$
VM
(4) Exec txs in VM
Enclave $\mathbb{E}$

Figure 1: Operation procedure of Aquareum ledger.

After the update of L is finished, clients with receipts obtained can verify that their transactions were processed by $\mathbb{E}$ (see details in Section 4.3). The update procedure ensures that the new version of L is: (1) **internally correct** since it was executed by trusted code of $\mathbb{E}$, (2) a **consistent** extension of the previous version – relying on trusted code of $\mathbb{E}$ and a witnessed version transition by $\mathbb{S}$, and (3) **non-equivocating** since $\mathbb{S}$ stores only hash of a single version of L (i.e., *LHash*) at any point in time.

Whenever $\mathbb{C}$ suspects that her transactions or read queries are censored, $\mathbb{C}$ might send such requests via $\mathbb{S}$ (see details in Section 4.4 and Section 4.5). To do so, $\mathbb{C}$ encrypts her request with $PK_{\mathbb{E}}^{pb}$ and publishes it on the blockchain via $\mathbb{S}$. $\mathbb{O}$ noticing a new request is obligated to pass the request to $\mathbb{E}$, which will process the request and reply with an encrypted response (by $PK_{\mathbb{C}}^{pb}$) that is processed by $\mathbb{S}$. If a pending request at $\mathbb{S}$ is not handled by $\mathbb{O}$, it is public evidence that $\mathbb{O}$ censors the request. We do not specify how can $\mathbb{C}$ utilize such a proof, but it could be shown in a potential legal dispute or $\mathbb{S}$ itself could have an automated deposit-based punishments rules.

3.4 Design Consideration

We might design L as an append-only chain (as in blockchains), but such a design would bring a high overhead on clients who want to verify that a particular block is a part of L. During the verification, clients would have to download the headers of all blocks between the head of L and the block in the query, resulting into linear space & time complexity. In contrast, when a history tree (see Section 2.4) is utilized for integrity preservation of L, the presence of any block in L can be verified with logarithmic space and time complexity.

[1] Note that neither of the private keys ever leaves $\mathbb{E}$.

[2] Note that $\{h(\partial L_i), h(\partial L_{i+1})\}_{\mathbb{E}} = \{h(L_i), h(L_{i+1})\}_{\mathbb{E}}$

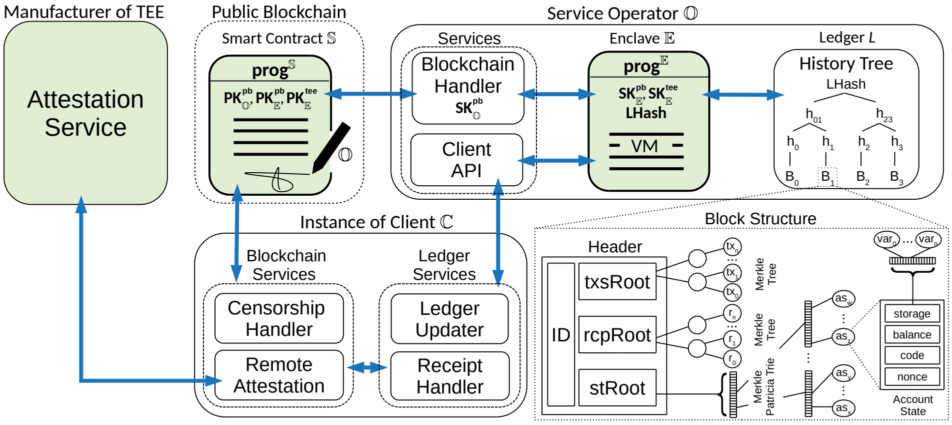


Figure 2: Aquareum components. Trusted components are depicted in green.

4 DETAILS

The schematic overview of Aquareum is depicted in Figure 2, where trusted components are depicted in green. The right part of the figure describes data aggregation of L. We utilize a history tree [12] for tamper-evident logging of data blocks due to its efficient membership and incremental proofs (see Section 3.4). The aggregation of blocks within a history tree is represented by root hash $LRoot$, which instantiates ledger hash $LHash$ from Section 3.3. In Aquareum, each data block consists of a header, a list of transactions, and a list of execution receipts from VM that is running within $\mathbb{E}$. A header contains the following fields:

- **ID**: this field is assigned for each newly created block as a counter of all blocks. ID of each block represents the IDth version of the history tree of L, which contains blocks $B_0, \ldots, B_{ID-1}$ and is characterized by the root hash $r \leftarrow MkRoot(\{H_0, \ldots, H_{ID-1}\})$, where H_i stands for a header of a block B_i. Note that the IDth version of L with the root hash r can also be expressed by the notation $\#(r)$.
- **txsRoot**, **rcpRoot**: two root hash values that aggregate set of transactions and the set of their corresponding execution receipts (containing execution logs) by Merkle trees [34],
- **stRoot**: the root hash that aggregates the current global state of the virtual machine by Merkle-Patricia trie [40, 49]. In detail, MPT aggregates all account states into a global state, where keys of MPT represent IDs of client accounts (i.e., $h(PK^{pb}_{\mathbb{C}})$) and values represent an account state data structures, which (similar to [49]) contains: (1) *balance* of a native token (if any), (2) *code* that is executed when an account receives a transaction; accounts with no code represent simple accounts and accounts with a code field represent smart contract accounts, (3) *nonce* represents the number of transactions sent from the simple account or the number of contracts created by the smart contract account, (4) *storage* represents encoded variables of a smart contract, which can be realized by Merkle-Patricia trie [40, 49] or other integrity-preserving mapping structures.

Although $\mathbb{O}$ persists the full content of L (and maintains its full state in the memory), she is unable to directly modify L while remaining undetected since all modifications of L must be done through $\mathbb{E}$. In detail, upon receiving enough transactions from clients, $\mathbb{E}$ executes received transactions by its virtual machine (VM) and updates L accordingly. While updating L, $\mathbb{E}$ leverages the incremental proofs of the history tree to ensure integrity and consistency with the past versions of L.

The enclave $\mathbb{E}$ in our approach stores the last produced header (hdr_{last}) and the current root hash of the history tree of L (i.e., $LRoot$), which enables $\mathbb{E}$ to make extensions of L that are consistent with L's history and at the same time avoiding dishonest $\mathbb{O}$ to tamper with L. Although state-fullness of $\mathbb{E}$ might be seen as a limitation in the case of a failed enclave, we show how to deal with this situation and provide a procedure that publicly replaces a failed enclave using $\mathbb{S}$ (see Section 4.6).

4.1 Setup

The setup of Aquareum is presented in Figure 3.[3] First, $\mathbb{O}$ initializes an empty L, a root hash $LRoot_{cur}$ for the most recent local version of L, the root hash $LRoot_{pb}$ for the version synchronized with PB, the empty global state of L, and the empty list of reported censored requests. Then, $\mathbb{O}$ initializes $\mathbb{E}$ with code $prog^{\mathbb{E}}$ (see Algorithm 1). In this initialization, $\mathbb{E}$ generates two key-pairs, $SK^{pb}_{\mathbb{E}}, PK^{pb}_{\mathbb{E}}$ and $SK^{tee}_{\mathbb{E}}, PK^{tee}_{\mathbb{E}}$, respectively; the first key-pair is intended for interaction with the blockchain platform and the second one is intended for the remote attestation with TEE infrastructure. Next, $\mathbb{E}$ initializes L and two root hashes in the same vein as $\mathbb{O}$ did. In addition, $\mathbb{E}$ stores the header hdr_{cur} of the last block created and signed by $\mathbb{E}$ and its ID. Then, $\mathbb{E}$ sends its public keys $PK^{pb}_{\mathbb{E}}$ and $PK^{tee}_{\mathbb{E}}$ to $\mathbb{O}$. Next, $\mathbb{O}$ creates a deployment transaction of $\mathbb{S}$'s code $prog^{\mathbb{S}}$ (see Algorithm 2) with public keys $PK^{pb}_{\mathbb{E}}, PK^{tee}_{\mathbb{E}}, PK_{\mathbb{O}}$ as the arguments (see Algorithm 4 in Appendix for pseudo-code of $\mathbb{O}$). Then, $\mathbb{O}$ sends the deployment transaction to the blockchain. In the constructor of $\mathbb{S}$, all public keys are stored, and the root hash of L with the list of censored requests are initialized. Finally, $\mathbb{S}$ publishes its identifier $\mathbb{S}^{ID}$, which serves as a public reference to $\mathbb{S}$.

When the infrastructure of Aquareum is initialized, $\mathbb{C}$s register at $\mathbb{O}$. For simplicity, we omit details of the registration and access control, and we let this up to the discretion of $\mathbb{O}$.

[3] We assume that $\mathbb{O}$ has already generated her public/private key-pair $PK^{pb}_{\mathbb{O}}$, $SK^{pb}_{\mathbb{O}}$.

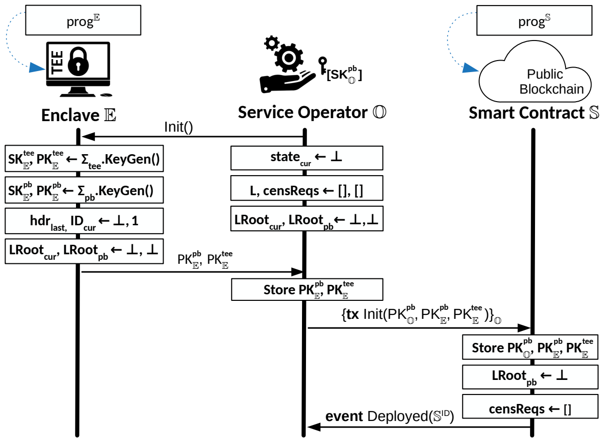

$prog^{\mathbb{E}}$
TEE
Enclave $\mathbb{E}$
$[SK^{pb}_{\mathbb{O}}]$
Service Operator $\mathbb{O}$
$prog^{\mathbb{S}}$
Public Blockchain
Smart Contract $\mathbb{S}$
Init()
$SK^{tee}_{\mathbb{E}}, PK^{tee}_{\mathbb{E}} \leftarrow \Sigma_{tee}.KeyGen()$
$SK^{pb}_{\mathbb{E}}, PK^{pb}_{\mathbb{E}} \leftarrow \Sigma_{pb}.KeyGen()$
$hdr_{last,} ID_{cur} \leftarrow \perp, 1$
$LRoot_{cur}, LRoot_{pb} \leftarrow \perp, \perp$
$state_{cur} \leftarrow \perp$
L, censReqs ← [], []
$LRoot_{cur}, LRoot_{pb} \leftarrow \perp,\perp$
$PK^{pb}_{\mathbb{E}}, PK^{tee}_{\mathbb{E}}$
Store $PK^{pb}_{\mathbb{E}}, PK^{tee}_{\mathbb{E}}$
{tx Init($PK^{pb}_{\mathbb{O}}, PK^{pb}_{\mathbb{E}}, PK^{tee}_{\mathbb{E}}$)}$_{\mathbb{O}}$
Store $PK^{pb}_{\mathbb{O}}, PK^{pb}_{\mathbb{E}}, PK^{tee}_{\mathbb{E}}$
$LRoot_{pb} \leftarrow \perp$
censReqs ← []
event Deployed($\mathbb{S}^{ID}$)

Figure 3: Protocol for setup (Π_S).

4.2 Normal Operation

In the case of normal operation (see Figure 4), $\mathbb{O}$ is not censoring any transactions produced by instances of $\mathbb{C}$, hence all transactions are correctly executed within $\mathbb{E}$ and appended to L, while $\mathbb{S}$ publicly witnesses the correct execution of transactions and the consistency of the new version of L with its history. In detail, when $\mathbb{O}$ receives a transaction from $\mathbb{C}_x$, it performs access control of $\mathbb{C}_x$,[4] and upon the success, $\mathbb{O}$ adds the transaction in its cache of unprocessed transactions (see Algorithm 4 in Appendix). When $\mathbb{O}$ accumulates enough transactions from clients, it passes these transactions to $\mathbb{E}$, together with the current partial state $\partial state_{cur}$ of the VM.

4.2.1 *VM Execution with Partial State.*

The current partial state of VM represents only data related to all account states that the execution of transactions is about to modify or create. The motivation for such an approach is the limited memory size of $\mathbb{E}$ (e.g., in the case of SGX it is only ~100MB), which does not allow to internally store the full global state of L (neither L itself). The partial state does not contain only the account states of concerning transactions, but it also contains intermediary nodes of MPT (i.e., extension and branch nodes) that are on the path from the root node of MPT to leaf nodes of concerning account states. Using passed partial state, $\mathbb{E}$ verifies its integrity, obtains the state root of MPT and compares it with the last known state root (i.e., $hdr_{last}.stRoot$) produced by $\mathbb{E}$. If the roots match, $\mathbb{E}$ executes transactions using the passed partial state, obtains the new partial state of VM and execution receipts with additional information about the execution of particular transactions (i.e., return codes and logs). Note that $\mathbb{E}$ obtains the new partial state by consecutively updating the current partial state with each transaction executed.

Next, $\mathbb{E}$ creates the header of the new block (i.e., hdr_{cur}) from aggregated transactions, receipts, and new partial state. Using created header, $\mathbb{E}$ extends the previous version of the history tree of L, while obtaining the new root hash $LRoot_{cur}$ of L (see Section 4.2.2). Then, $\mathbb{E}$ signs a version transition pair $\langle LRoot_{pb}, LRoot_{cur}\rangle$ of the history tree by $SK^{pb}_{\mathbb{E}}$ and sends it to $\mathbb{O}$, together with the new header, the new partial state, and execution receipts. Moreover, $\mathbb{E}$ stores the last produced header (i.e., hdr_{last}) and the root hash $LRoot_{cur}$ of L associated with the last version of the history tree.

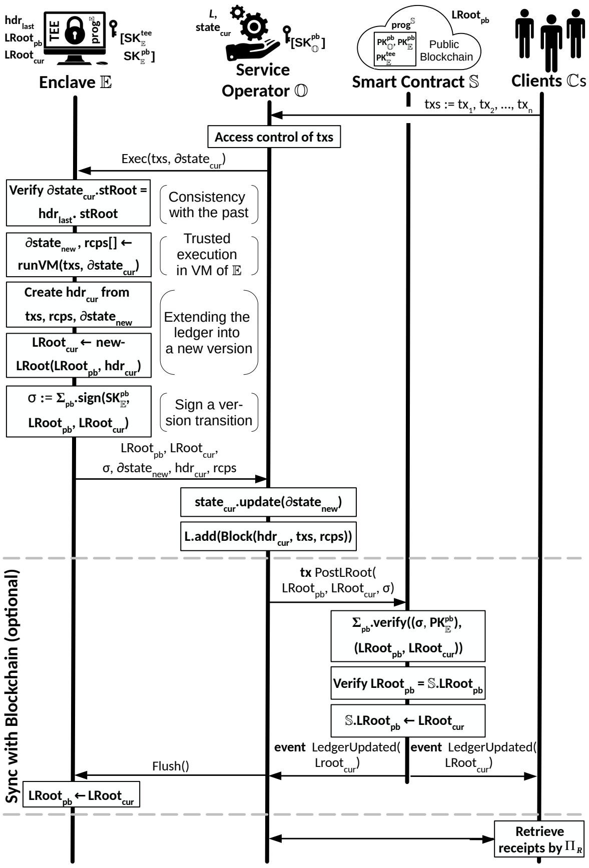

hdr_{last}
$LRoot_{pb}$
$LRoot_{cur}$
TEE
$prog^{\mathbb{E}}$
$[SK^{tee}_{\mathbb{E}}$
$SK^{pb}_{\mathbb{E}}]$
Enclave $\mathbb{E}$
L,
$state_{cur}$
$[SK^{pb}_{\mathbb{O}}]$
Service Operator $\mathbb{O}$
$prog^{\mathbb{S}}$
$LRoot_{pb}$
$PK^{pb}_{\mathbb{O}}, PK^{pb}_{\mathbb{E}}$
$PK^{tee}_{\mathbb{E}}$
Public Blockchain
Smart Contract $\mathbb{S}$
Clients $\mathbb{C}$s
$txs := tx_1, tx_2, ..., tx_n$
Access control of txs
Exec(txs, $\partial state_{cur}$)
Verify $\partial state_{cur}.stRoot = hdr_{last}. stRoot$
Consistency with the past
$\partial state_{new}$, rcps[] ← runVM(txs, $\partial state_{cur}$)
Trusted execution in VM of $\mathbb{E}$
Create hdr_{cur} from txs, rcps, $\partial state_{new}$
$LRoot_{cur}$ ← new-LRoot($LRoot_{pb}, hdr_{cur}$)
Extending the ledger into a new version
$\sigma := \Sigma_{pb}.sign(SK^{pb}_{\mathbb{E}}, LRoot_{pb}, LRoot_{cur})$
Sign a version transition
$LRoot_{pb}, LRoot_{cur}, \sigma, \partial state_{new}, hdr_{cur}$, rcps
$state_{cur}.update(\partial state_{new})$
L.add(Block(hdr_{cur}, txs, rcps))
Sync with Blockchain (optional)
tx PostLRoot($LRoot_{pb}, LRoot_{cur}, \sigma$)
$\Sigma_{pb}.verify((\sigma, PK^{pb}_{\mathbb{E}}), (LRoot_{pb}, LRoot_{cur}))$
Verify $LRoot_{pb} = \mathbb{S}.LRoot_{pb}$
$\mathbb{S}.LRoot_{pb} \leftarrow LRoot_{cur}$
event LedgerUpdated($Lroot_{cur}$)
event LedgerUpdated($LRoot_{cur}$)
Flush()
$LRoot_{pb} \leftarrow LRoot_{cur}$
Retrieve receipts by Π_R

Figure 4: Protocol for normal operation (Π_N).

4.2.2 *Incremental Update of the Ledger.*

We omitted the details about the consistent update of L within $\mathbb{E}$ in the above text and Figure 4. In general, an incremental update of a history tree assumes trusted full access to its data. However, $\mathbb{E}$ does not store full data of L (only the last header created), and thus cannot directly make a consistent update of L. Therefore, we design a simple procedure that enables $\mathbb{E}$ to extend L without storing it internally.

In the procedure, $\mathbb{O}$ creates a proof template π^{inc}_{next} for the next incremental proof of the history tree, which extends L exactly by one empty block (see function $nextIncProof()$ in Algorithm 4) while obtaining a new version of L with the (temporary) root hash $LRoot_{tmp}$. Note that this template represents ∂L from Section 3.3, and it enables $\mathbb{E}$ to make an integrity verification and consistent extension of L without storing it. In detail, $\mathbb{O}$ sends π^{inc}_{next} and $LRoot_{tmp}$ to $\mathbb{E}$, together with transactions that are about to be processed by function $Exec()$. $\mathbb{E}$ verifies π^{inc}_{next} with respect to its last known version of L (i.e., $\#(LRoot_{cur})$), replaces the header hash of the empty block in the proof template by the hash of newly created header in $\mathbb{E}$, and

[4] For simplicity, we omit access control at $\mathbb{O}$.

Algorithm 1: The program $prog^{\mathbb{E}}$ of enclave $\mathbb{E}$

```
▷ Declaration of types and functions:
  Header { $ID$, $txsRoot$, $rcpRoot$, $stRoot$};
  $\#(r) \rightarrow v$: denotes the version $v$ of $L$ having $LRoot = r$,
▷ Variables of TEE:
  $SK_{\mathbb{E}}^{tee}$, $PK_{\mathbb{E}}^{tee}$: keypair of $\mathbb{E}$ under $\Sigma_{tee}$,
  $SK_{\mathbb{E}}^{pb}$, $PK_{\mathbb{E}}^{pb}$: keypair of $\mathbb{E}$ under $\Sigma_{pb}$,
  $hdr_{last} \leftarrow \perp$: the last header created by $\mathbb{E}$,
  $LRoot_{pb} \leftarrow \perp$: the last root of $L$ flushed to PB,
  $LRoot_{cur} \leftarrow \perp$: the root of $L \cup blks_p$ (not flushed to PB),
  $ID_{cur} \leftarrow 1$: the current version of $L$ (not flushed to PB),
▷ Declaration of functions:
function $Init()$ public
    $(SK_{\mathbb{E}}^{pb}, PK_{\mathbb{E}}^{pb}) \leftarrow \Sigma_{pb}.Keygen()$;
    $(SK_{\mathbb{E}}^{tee}, PK_{\mathbb{E}}^{tee}) \leftarrow \Sigma_{tee}.Keygen()$;
    Output($PK_{\mathbb{E}}^{tee}$, $PK_{\mathbb{E}}^{pb}$);

function $Exec(txs[], \partial st^{old}, \pi_{next}^{inc}, LRoot_{tmp})$ public
    assert $\partial st^{old}.root = hdr_{last}.stRoot$;
    $\partial st^{new}, rcps, txs_{er} \leftarrow processTxs(txs, \partial st^{old}, \pi_{next}^{inc}, LRoot_{tmp})$;
    $\sigma \leftarrow \Sigma_{pb}.sign(SK_{\mathbb{E}}^{pb}, (LRoot_{pb}, LRoot_{cur}))$;
    Output($LRoot_{pb}, LRoot_{cur}, \partial st^{new}, hdr_{last}, rcps, txs_{er}, \sigma$);

function $Flush()$ public
    $LRoot_{pb} \leftarrow LRoot_{cur}$;        ▷ Shift the version of L synchronized with PB.

function $processTxs(txs[], \partial st^{old}, \pi_{next}^{inc}, LRoot_{tmp})$ private
    $\partial st^{new}, rcps[], txs_{er} \leftarrow$ runVM($txs, \partial st^{old}$);        ▷ Run $txs$ in VM.
    $txs \leftarrow txs \setminus txs_{er}$;        ▷ Filter out parsing errors/wrong signatures.
    $hdr \leftarrow$ Header($ID_{cur}, MkRoot(txs), MkRoot(rcps), \partial st^{new}.root$));
    $hdr_{last} \leftarrow hdr$;
    $ID_{cur} \leftarrow ID_{cur} + 1$;
    $LRoot_{cur} \leftarrow newLRoot(hdr, \pi_{next}^{inc}, LRoot_{tmp})$;
    return $\partial st^{new}, rcps, txs_{er}$;

function $newLRoot(hdr, \pi_{next}^{inc}, LRoot_{tmp})$ private
            ▷ A modification of the incr. proof. template to contain $hdr$
    assert $\#(LRoot_{cur}) + 1 = \#(LRoot_{tmp})$;        ▷ 1 block $\Delta$.
    assert $\pi_{next}^{inc}.Verify(LRoot_{cur}, LRoot_{tmp})$;
    $\pi_{next}^{inc}[-1] \leftarrow h(hdr)$;
    return $deriveNewRoot(\pi_{next}^{inc})$;
```

Algorithm 2: The program $prog^{\mathbb{S}}$ of the smart contract $\mathbb{S}$

```
▷ Declaration of types and constants:
  CensInfo { $etx$, $equery$, $status$, $edata$ },
  $msg$: a current transaction that called $\mathbb{S}$,
▷ Declaration of functions:
function $Init(PK_{\mathbb{E}}^{pb}, PK_{\mathbb{E}}^{tee}, PK_{\mathbb{O}})$ public
    $PK_{\mathbb{E}}^{tee}[].add(PK_{\mathbb{E}}^{tee})$;        ▷ PK of enclave $\mathbb{E}$ under $\Sigma_{tee}$.
    $PK_{\mathbb{E}}^{pb}[].add(PK_{\mathbb{E}}^{pb})$;        ▷ PK of enclave $\mathbb{E}$ under $\Sigma_{pb}$.
    $PK_{\mathbb{O}}^{pb} \leftarrow PK_{\mathbb{O}}$;        ▷ PK of operator $\mathbb{O}$ under $\Sigma_{pb}$.
    $LRoot_{pb} \leftarrow \perp$;        ▷ The most recent root hash of L synchronized with $\mathbb{S}$.
    $censReqs \leftarrow []$;        ▷ Request that $\mathbb{C}$s wants to resolve publicly.
function $PostLRoot(root_A, root_B, \sigma)$ public
            ▷ Verify whether a state transition was made within $\mathbb{E}$.
    assert $\Sigma_{pb}.verify((\sigma, PK_{\mathbb{E}}^{pb}[-1]), (root_A, root_B))$;
            ▷ Verify whether a version transition extends the last one.
    if $LRoot_{pb} = root_A$ then
        $LRoot_{pb} \leftarrow root_B$;        ▷ Do a version transition of L.

function $ReplaceEnc(PKN_{\mathbb{E}}^{pb}, PKN_{\mathbb{E}}^{tee}, r_A, r_B, \sigma, \sigma_{msg})$ public
            ▷ Called by $\mathbb{O}$ in the case of enclave failure.
    assert $\Sigma_{pb}.verify((\sigma_{msg}, PK_{\mathbb{O}}^{pb}), msg)$;        ▷ Avoiding MiTM attack.
    $PostLRoot(r_A, r_B, \sigma)$ ;        ▷ Do a version transition.
    $PK_{\mathbb{E}}^{tee}.add(PKN_{\mathbb{E}}^{tee})$;        ▷ Upon change, $\mathbb{C}$s make remote attestation.
    $PK_{\mathbb{E}}^{pb}.add(PKN_{\mathbb{E}}^{pb})$;
function $SubmitCensTx(etx, \sigma_{msg})$ public
            ▷ Called by $\mathbb{C}$ in the case her TX is censored.
    accessControl($\sigma_{msg}, msg.PK_{\mathbb{C}}^{pb}$);
    $censReqs$.add(CensInfo($etx, \perp, \perp, \perp$));
function $ResolveCensTx(idx_{req}, status, \sigma)$ public
            ▷ Called by $\mathbb{O}$ to prove that $\mathbb{C}$'s TX was processed.
    assert $idx_{req} < |censReqs|$;
    $r \leftarrow censReqs[idx_{req}]$;
    assert $\Sigma_{pb}.verify((\sigma, PK_{\mathbb{E}}^{pb}[-1]), (h(r.etx), status))$;
    $r.status \leftarrow status$;
function $SubmitCensQry(equery, \sigma_{msg})$ public
            ▷ Called by $\mathbb{C}$ in the case its read query is censored.
    accessControl($\sigma_{msg}, msg.PK_{\mathbb{C}}^{pb}$);
    $censReqs$.add(CensInfo($\perp, equery, \perp, \perp$));
function $ResolveCensQry(idx_{req}, status, edata, \sigma)$ public
            ▷ Called by $\mathbb{O}$ as a response to the $\mathbb{C}$'s censored read query.
    assert $idx_{req} < |censReqs|$;
    $r \leftarrow censReqs[idx_{req}]$;
    assert
      $\Sigma_{pb}.verify((\sigma, PK_{\mathbb{E}}^{pb}[-1]), (h(r.equery), status, h(edata)))$;
    $r.\{edata \leftarrow edata, status \leftarrow status\}$;
```

then uses such modified proof to compute the new root hash of L, which is then stored as $LRoot_{cur}$ by $\mathbb{E}$.

When $\mathbb{O}$ receives the output of $\mathbb{E}$, it updates the full state of L and creates the new block using client transactions, the received receipts, and the header of the new block. Then, $\mathbb{O}$ appends the new block to L and responds to client requests for receipts of their transactions (see Section 4.3), which serve as *promises* confirming the execution of transactions. These promises became irreversible when O syncs L with $\mathbb{S}$ that runs on the blockchain platform.

4.2.3 ***Syncing the Ledger with the Blockchain.*** $\mathbb{O}$ periodically syncs L with $\mathbb{S}$ to provide non-equivocation of L. However, $\mathbb{O}$ is able to sync only such a version of L that was signed within $\mathbb{E}$ and is newer than the last known version by $\mathbb{S}$, which provides consistency and non-equivocation of L. During the sync of L, $\mathbb{O}$ creates a special blockchain transaction containing the version transition pair $\langle LRoot_{pb}, LRoot_{cur}\rangle$ signed within $\mathbb{E}$ and sends it to $\mathbb{S}$ (i.e., calling the function *PostLRoot*()). $\mathbb{S}$ verifies whether the version transition pair was signed within $\mathbb{E}$ by checking the signature with $PK_{\mathbb{E}}^{pb}$. Then, $\mathbb{S}$ verifies whether the last published version of L (corresponding to $LRoot_{pb}$ at $\mathbb{S}$) is equal to the first entry in the version transition pair. In the positive case, $\mathbb{S}$ publicly performs the version transition of L by updating its $LRoot_{pb}$ to the second item of the version transition pair. From that moment, the Aquareum transactions processed until the current version of L cannot be tampered with – providing a non-equivocation of L. Finally, $\mathbb{O}$ notifies $\mathbb{E}$ about successful sync by calling function *Flush*() (see Algorithm 1), where $\mathbb{E}$ "shifts" $LRoot_{pb}$ to $LRoot_{cur}$.

Note that if $\mathbb{O}$ were to sync L with $\mathbb{S}$ upon every new block created, it might be too expensive. On the other hand, if $\mathbb{O}$ were to sync L to $\mathbb{S}$ with long delays, "a level" of non-equivocation would be decreased, which in turn would extend the time to finality. Hence, the sync interval must be viewed as a trade-off between costs and a level of non-equivocation (see examples in Section 5.2). The frequency of syncs might be defined in SLA with clients and violation might be penalized by $\mathbb{S}$.

4.3 Retrieval and Verification of Receipts

Receipt retrieval and verification serves as a lightweight audit procedure in which $\mathbb{C}$ verifies inclusion and execution of the transaction

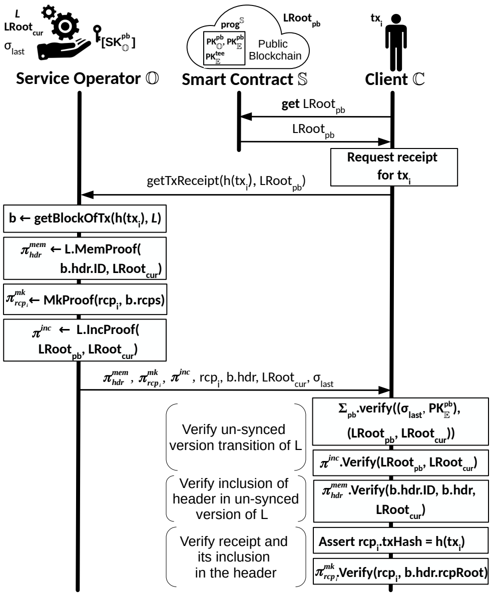

L
$LRoot_{cur}$
σ_{last}
$[SK_{\mathbb{O}}^{pb}]$
Service Operator $\mathbb{O}$
$prog^{\mathbb{S}}$
$PK_{\mathbb{O}}^{pb}, PK_{\mathbb{E}}^{pb}$
$PK_{\mathbb{E}}^{tee}$
Public Blockchain
$LRoot_{pb}$
Smart Contract $\mathbb{S}$
tx_i
Client $\mathbb{C}$
get $LRoot_{pb}$
$LRoot_{pb}$
Request receipt for tx_i
getTxReceipt($h(tx_i)$, $LRoot_{pb}$)
b ← getBlockOfTx($h(tx_i)$, L)
π_{hdr}^{mem} ← L.MemProof(b.hdr.ID, $LRoot_{cur}$)
$\pi_{rcp_i}^{mk}$ ← MkProof(rcp_i, b.rcps)
π^{inc} ← L.IncProof($LRoot_{pb}$, $LRoot_{cur}$)
π_{hdr}^{mem}, $\pi_{rcp_i}^{mk}$, π^{inc}, rcp_i, b.hdr, $LRoot_{cur}$, σ_{last}
Verify un-synced version transition of L
Σ_{pb}.verify((σ_{last}, $PK_{\mathbb{E}}^{pb}$), ($LRoot_{pb}$, $LRoot_{cur}$))
π^{inc}.Verify($LRoot_{pb}$, $LRoot_{cur}$)
Verify inclusion of header in un-synced version of L
π_{hdr}^{mem}.Verify(b.hdr.ID, b.hdr, $LRoot_{cur}$)
Verify receipt and its inclusion in the header
Assert rcp_i.txHash = $h(tx_i)$
$\pi_{rcp_i}^{mk}$.Verify(rcp_i, b.hdr.rcpRoot)

Figure 5: Protocol for receipt retrieval (Π_R).

tx_i by obtaining its receipt. An execution receipt contains three fields: (1) hash of tx_i, (2) return code of VM, and (3) log of events emitted by EVM.

To obtain an execution receipt of tx_i, $\mathbb{C}$ first retrieves the last root hash of L (i.e., $LRoot_{pb}$) from $\mathbb{S}$. Then, $\mathbb{C}$ requests $\mathbb{O}$ for an inclusion proof of her transaction tx_i in the most recent version of L that extends the version $\#(LRoot_{pb})$. Upon request, $\mathbb{O}$ finds a block b that contains tx_i and computes a membership proof π_{hdr}^{mem} of b's header in the most recent version $\#(LRoot_{cur})$ of L. The second proof that $\mathbb{O}$ computes is the Merkle proof $\pi_{rcp_i}^{mk}$, which witnesses that receipt rcp_i of transaction tx_i is included in the block b. Then, $\mathbb{O}$ computes the incremental proof π^{inc} of the most recent version transition $\langle LRoot_{pb}, LRoot_{cur} \rangle$ that was executed within $\mathbb{E}$. In response, $\mathbb{O}$ sends the following data to $\mathbb{C}$:

- the receipt rcp_i with its proof $\pi_{rcp_i}^{mk}$,
- the header of b with its proof π_{hdr}^{mem},
- the most recent version $LRoot_{cur}$ of L with its proof π^{inc},
- the signature σ_{last} of the most recent version transition $\langle LRoot_{pb}, LRoot_{cur} \rangle$ made by $\mathbb{E}$.

$\mathbb{C}$ verifies the signature and the provided proofs against $LRoot_{pb}$, and it also checks whether the retrieved receipt corresponds to tx_i. In the positive case, $\mathbb{C}$ has a guarantee that the transaction tx_i was included in L and its execution in VM exited with a particular status, represented by a return code in the receipt.

We highlight that the previous receipt retrieval protocol assumes that tx_i is "very recent," and is included only in the version of L that was not synchronized with $\mathbb{S}$ yet. When tx_i is already included in the synchronized version of L, we can put $LRoot_{cur} = LRoot_{pb}$ in the protocol, and thus omit computation of π^{inc} and its verification. We also note that the receipt retrieval protocol can be integrated with the transaction submission in Π_N by following it.

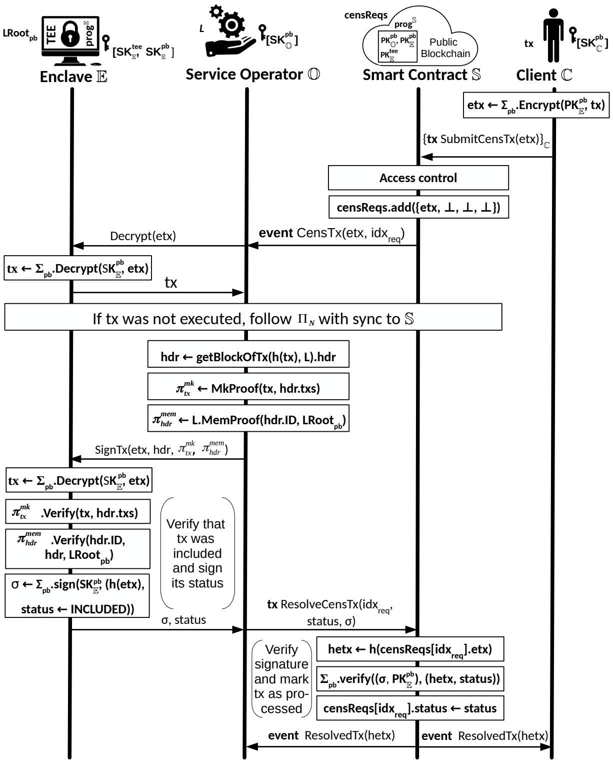

$LRoot_{pb}$
TEE
$prog^{\mathbb{E}}$
$[SK_{\mathbb{E}}^{tee}, SK_{\mathbb{E}}^{pb}]$
Enclave $\mathbb{E}$
L
$[SK_{\mathbb{O}}^{pb}]$
Service Operator $\mathbb{O}$
censReqs
$prog^{\mathbb{S}}$
$PK_{\mathbb{O}}^{pb}, PK_{\mathbb{E}}^{pb}$
$PK_{\mathbb{E}}^{tee}$
Public Blockchain
Smart Contract $\mathbb{S}$
tx
$[SK_{\mathbb{C}}^{pb}]$
Client $\mathbb{C}$
etx ← Σ_{pb}.Encrypt($PK_{\mathbb{E}}^{pb}$, tx)
$\{$**tx** SubmitCensTx(etx)$\}_{\mathbb{C}}$
Access control
censReqs.add({etx, ⊥, ⊥, ⊥})
Decrypt(etx)
event CensTx(etx, idx_{req})
tx ← Σ_{pb}.Decrypt($SK_{\mathbb{E}}^{pb}$, etx)
tx
If tx was not executed, follow Π_N with sync to $\mathbb{S}$
hdr ← getBlockOfTx(h(tx), L).hdr
π_{tx}^{mk} ← MkProof(tx, hdr.txs)
π_{hdr}^{mem} ← L.MemProof(hdr.ID, $LRoot_{pb}$)
SignTx(etx, hdr, π_{tx}^{mk}, π_{hdr}^{mem})
tx ← Σ_{pb}.Decrypt($SK_{\mathbb{E}}^{pb}$, etx)
π_{tx}^{mk} .Verify(tx, hdr.txs)
π_{hdr}^{mem} .Verify(hdr.ID, hdr, $LRoot_{pb}$)
σ ← Σ_{pb}.sign($SK_{\mathbb{E}}^{pb}$, (h(etx), status ← INCLUDED))
Verify that tx was included and sign its status
σ, status
tx ResolveCensTx(idx_{req}, status, σ)
Verify signature and mark tx as pro-cessed
hetx ← h(censReqs[idx_{req}].etx)
Σ_{pb}.verify((σ, $PK_{\mathbb{E}}^{pb}$), (hetx, status))
censReqs[idx_{req}].status ← status
event ResolvedTx(hetx)
event ResolvedTx(hetx)

Figure 6: Protocol for resolution of a censored transaction (Π_{CT}).

4.4 Resolution of Censored Transactions

In the case when $\mathbb{C}$ suspects $\mathbb{O}$ of censoring a transaction tx (see Figure 6), $\mathbb{C}$ initiates a request for an inclusion proof of tx through $\mathbb{S}$. In detail, $\mathbb{C}$ creates a transaction of the public blockchain platform, which calls the function $SubmitCensTx()$ with tx encrypted by $PK_{\mathbb{E}}^{pb}$ (i.e., etx) as an argument and sends it to $\mathbb{S}$; hence, preserving confidentially for public. $\mathbb{S}$ does the access control (see Section 6.1.2), appends etx to the list of censored requests,[5] and generates asynchronous event informing $\mathbb{O}$ about new unresolved transaction. When $\mathbb{O}$ receives the event, first she decrypts tx through $\mathbb{E}$ and then executes tx in $\mathbb{E}$ if it has not been executed before. If a fresh execution of tx occurred, $\mathbb{O}$ syncs the most recent version of L with $\mathbb{S}$. Then, $\mathbb{O}$ sends the encrypted tx to $\mathbb{E}$ (i.e., function $SignTx()$) together with the header and the proofs that bind tx to L, i.e., to its version $\#(LRoot_{pb})$. In the function $SignTx()$ (see Algorithm 1), $\mathbb{E}$ decrypts tx and checks whether it is correctly parsed and whether its signature is correct. If these checks are not successful, $\mathbb{E}$ includes this information into a status of the response and signs it. If the checks are successful, $\mathbb{E}$ proceeds to the verification of provided proofs with regard to the version $\#(LRoot_{pb})$ of L synchronized to $\mathbb{S}$. Upon successful verification, $\mathbb{E}$ signs both the transaction's status and the hash of encrypted tx, and then returns them to $\mathbb{O}$, who publishes the signature and the status through $\mathbb{S}$ (i.e., the function $ResolveCensTx()$). When $\mathbb{S}$ receives the message with the status of tx signed by $\mathbb{E}$, it computes the hash of etx and uses it in the

[5] Note that to save operational costs for allocating storage of the the smart contract platform, $\mathbb{S}$ can store only the hash of etx instead (see Section 5.2.1).

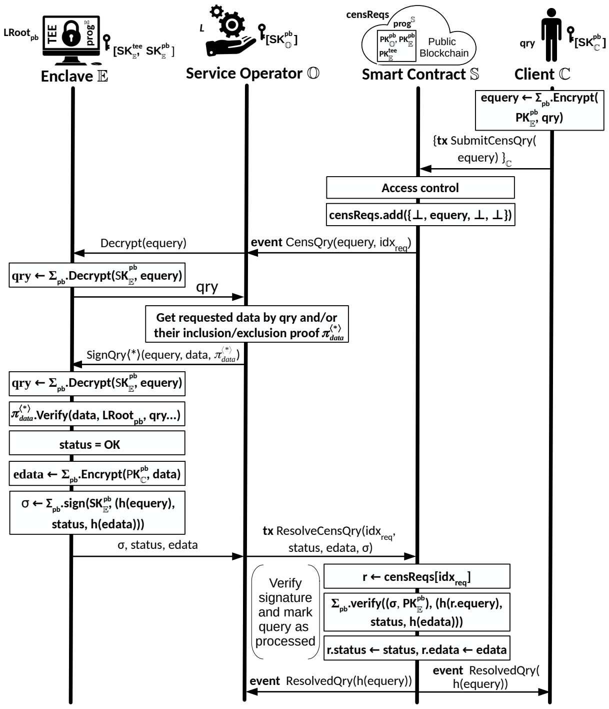

LRoot_pb
TEE
prog^𝔼
[SK_𝔼^tee, SK_𝔼^pb]
Enclave 𝔼
L
[SK_𝕆^pb]
Service Operator 𝕆
censReqs
prog^𝕊
PK_𝕆^pb, PK_𝔼^pb
PK_𝔼^tee
Public Blockchain
Smart Contract 𝕊
qry
[SK_ℂ^pb]
Client ℂ
equery ← Σ_pb.Encrypt(PK_𝔼^pb, qry)
{tx SubmitCensQry(equery) }_ℂ
Access control
censReqs.add({⊥, equery, ⊥, ⊥})
Decrypt(equery)
event CensQry(equery, idx_req)
qry ← Σ_pb.Decrypt(SK_𝔼^pb, equery)
qry
Get requested data by qry and/or their inclusion/exclusion proof π_data^⟨*⟩
SignQry⟨*⟩(equery, data, π_data^⟨*⟩)
qry ← Σ_pb.Decrypt(SK_𝔼^pb, equery)
π_data^⟨*⟩.Verify(data, LRoot_pb, qry...)
status = OK
edata ← Σ_pb.Encrypt(PK_ℂ^pb, data)
σ ← Σ_pb.sign(SK_𝔼^pb, (h(equery), status, h(edata)))
σ, status, edata
tx ResolveCensQry(idx_req, status, edata, σ)
Verify signature and mark query as processed
r ← censReqs[idx_req]
Σ_pb.verify((σ, PK_𝔼^pb), (h(r.equery), status, h(edata)))
r.status ← status, r.edata ← edata
event ResolvedQry(h(equery))
event ResolvedQry(h(equery))

Figure 7: Protocol for resolution of a censored query (Π_{CQ}).

signature verification. In the successful case, $\mathbb{S}$ updates the status of the suspected censored transaction with the result from $\mathbb{E}$. Finally, $\mathbb{S}$ notifies $\mathbb{C}$ and $\mathbb{O}$ about the resolution of *tx*.

4.5 Resolution of Censored Queries

Besides censoring transactions, $\mathbb{O}$ might also censor read queries of $\mathbb{C}$s. When $\mathbb{C}$ suspects $\mathbb{O}$ of censoring a read query *qry* (see Figure 7), $\mathbb{C}$ initiates inclusion of *qry* through $\mathbb{S}$. In detail, $\mathbb{C}$ creates a transaction of the public blockchain platform, which calls the function *SubmitCensQry*() (see Algorithm 2) with *qry* encrypted by $PK_{\mathbb{E}}^{pb}$ (i.e., *equery*) as an argument and sends it to $\mathbb{S}$. $\mathbb{S}$ does the access control (see Section 6.1.2), appends *equery* to the list of censored requests, and generates asynchronous event informing $\mathbb{O}$ about new unresolved query. When $\mathbb{O}$ receives the event, first she decrypts *qry* through $\mathbb{E}$, then fetches the data requested by *qry* and computes their inclusion proof(s) $\pi_{data}^{<*>}$ in the version of L synchronized to $\mathbb{S}$.[6] Then, $\mathbb{O}$ sends the encrypted *qry* to $\mathbb{E}$ (i.e., function *SignQry*⟨∗⟩()) together with the fetched data and their inclusion proof $\pi_{data}^{<*>}$. In the function *SignQry*⟨∗⟩(), $\mathbb{E}$ decrypts *qry* and checks whether it is correctly parsed. Upon the success, $\mathbb{E}$ proceeds to the verification of $\pi_{data}^{<*>}$ with regards to the passed data and the version #($LRoot_{pb}$) of L synchronized with $\mathbb{S}$. Upon successful verification, $\mathbb{E}$ encrypts data by $PK_{\mathbb{C}}^{pb}$ (i.e., *edata*) and signs the triplet consisting of the transaction's status, the hash of *eqry*, and the hash of *edata*; which are returned to $\mathbb{O}$. Then, $\mathbb{O}$ calls the function *ResolveCensQry*() of $\mathbb{S}$ (see Algorithm 2) with signature, status, and encrypted data contained in the arguments. When $\mathbb{S}$ receives the message with the status of *qry* signed by $\mathbb{E}$, it computes the hashes of *equery* and *edata*, and it uses them in the signature verification. Upon successful verification, $\mathbb{S}$ updates the status and *edata* of the suspected censored query with the data from $\mathbb{E}$. Finally, $\mathbb{S}$ notifies $\mathbb{C}$ and $\mathbb{O}$ about the resolution of *qry*. We provide code of $\mathbb{E}$ specific to censorship resolution and examples of handling different queries in Appendix A.1.

4.6 Terminated and Failed Enclave

During the execution of $prog^{\mathbb{E}}$, $\mathbb{E}$ stores its secrets and state objects in a sealed file, which is updated and stored on the hard drive of $\mathbb{O}$ with each new block created. Hence, if $\mathbb{E}$ terminates due a temporary reason, such as a power outage or intentional command by $\mathbb{O}$, it can be initialized again by $\mathbb{O}$ who provides $\mathbb{E}$ with the sealed file; this file is used to recover its protected state objects.

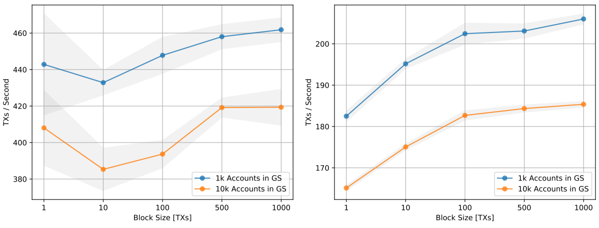
However, if $\mathbb{E}$ experiences a permanent hardware failure of TEE, the sealed file cannot be decrypted on other TEE platforms. Therefore, we propose a simple mechanism that deals with this situation under the assumption that $\mathbb{O}$ is the only allowed entity that can replace the platform of $\mathbb{E}$. In detail, $\mathbb{O}$ first snapshots the header hdr_{sync} of the last block that was synchronized with $\mathbb{S}$ as well as all blocks $blks_{unsync}$ of L that were not synchronized with $\mathbb{S}$. Then, $\mathbb{O}$ restores L and her internal state objects into the version #($LRoot_{pb}$). After the restoration of L, $\mathbb{O}$ calls the function *ReInit*() of $\mathbb{E}$ (see Algorithm 3) with hdr_{sync}, $blks_{unsync}$, and $LRoot_{pb}$ as the arguments. In this function, $\mathbb{E}$ first generates its public/private key-pair $SK_{\mathbb{E}}^{pb}$, $PK_{\mathbb{E}}^{pb}$, and then stores the passed header as hdr_{last} and copies the passed root hash into $LRoot_{cur}$ and $LRoot_{pb}$. Then, $\mathbb{E}$ iterates over all passed unprocessed blocks and their transactions *txs*, which are executed within VM of $\mathbb{E}$. Before the processing of *txs* of each passed block, $\mathbb{E}$ calls the unprotected code of $\mathbb{O}$ to obtain the current partial state ∂st^{old} of L and incremental proof template (see Section 4.2.2) that serves for extending L within $\mathbb{E}$. However, these unprotected calls are always verified within $\mathbb{E}$ and malicious $\mathbb{O}$ cannot misuse them. In detail, $\mathbb{E}$ verifies ∂st^{old} obtained from $\mathbb{O}$ against the root hash of the state stored in the last header hdr_{last} of $\mathbb{E}$, while the incremental proof template is also verified against $LRoot_{cur}$ in the function *newLRoot*() of $\mathbb{E}$.

Next, $\mathbb{E}$ processes *txs* of a block, extends L, and then it calls the unprotected code of $\mathbb{O}$ again, but this time to process *txs* of

Algorithm 3: Reinitialization of a failed $\mathbb{E}$ (part of $prog^{\mathbb{E}}$).

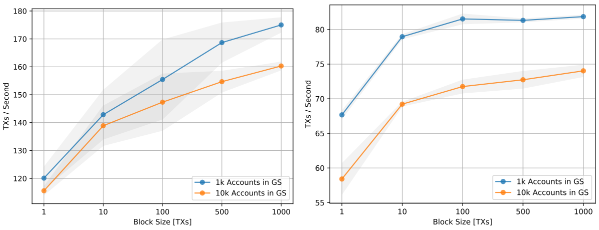
```
function $ReInit(LRoot_{old}, prevBlks[], hdr_{last})$ public
    $(SK_{\mathbb{E}}^{pb}, PK_{\mathbb{E}}^{pb}) \leftarrow \Sigma_{pb}.Keygen()$;
    $hdr_{last} \leftarrow hdr_{last}$;
    $LRoot_{cur} \leftarrow LRoot_{old}$, $LRoot_{pb} \leftarrow LRoot_{old}$;
    for $\{b : prevBlks\}$ do
        $\pi_{next}^{inc}$, $LRoot_{tmp} \leftarrow prog^{\mathbb{O}}.nextIncProof()$;
        $\partial st^{old} \leftarrow prog^{\mathbb{O}}.getPartialState(b.txs)$;
        assert $\partial st^{old}.root = hdr_{last}.stRoot$;
        $\ldots \leftarrow processTxs(b.txs, \partial st^{old}, \pi_{next}^{inc}, LRoot_{tmp})$;
        $LRoot_{ret} \leftarrow prog^{\mathbb{O}}.runVM(b.txs)$;          ▷ Run VM at $\mathbb{O}$.
        assert $LRoot_{cur} = LRoot_{ret}$;          ▷ $\mathbb{E}$ and $\mathbb{O}$ are at the same point.
    $\sigma \leftarrow \Sigma_{pb}.sign(SK_{\mathbb{E}}^{pb}, (LRoot_{pb}, LRoot_{cur}))$;
    Output($LRoot_{pb}$, $LRoot_{cur}$, $\sigma$, $PK_{pb}^{\mathbb{E}}$, $PK_{tee}^{\mathbb{E}}$);
```

[6] Note that if the query requests non-existing data, $\mathbb{O}$ creates an exclusion proof instead.



(a) Turbo Boost enabled **(b) Turbo Boost disabled**

Figure 8: Performance of Aquareum for native payments.



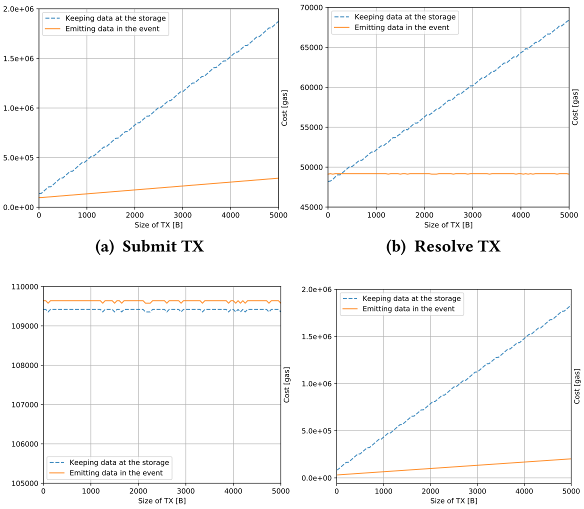
(a) Turbo Boost enabled **(b) Turbo Boost disabled**

Figure 9: Performance of Aquareum for ERC20 smart contract calls.

the current block by $\mathbb{O}$, and thus getting the same version and state of L in both $\mathbb{E}$ and $\mathbb{O}$. Note that any adversarial effect of this unprotected call is eliminated by the checks made after the former two unprotected calls. When all passed blocks are processed, $\mathbb{E}$ signs the version transition pair $\langle LRoot_{pb}, LRoot_{cur}\rangle$ and returns it to $\mathbb{O}$, together with the new public keys of $\mathbb{E}$. $\mathbb{O}$ creates a blockchain transaction that calls the function $ReplaceEnc()$ of $\mathbb{S}$ with data from $\mathbb{E}$ passed in the arguments. In $ReplaceEnc()$, $\mathbb{S}$ first verifies whether the signature of the transaction was made by $\mathbb{O}$ to avoid MiTM attacks on this functionality. Then, $\mathbb{S}$ calls its function $PostLRoot()$ with the signed version transition pair in the arguments. Upon the success, the current root hash of L is updated and $\mathbb{S}$ replaces the stored $\mathbb{E}$'s PKs by PKs passed in parameters. Finally, $\mathbb{E}$ informs $\mathbb{C}s$ by an event containing new PKs of $\mathbb{E}$, and $\mathbb{C}$s perform the remote attestation of $prog^{\mathbb{E}}$ using the new key $PK^{\mathbb{E}}_{tee}$ and the attestation service. We refer the reader to Algorithm 4 in Appendix for the relevant pseudo-code of $\mathbb{O}$.

5 IMPLEMENTATION

We have made a proof-of-concept implementation of Aquareum, where we utilized Intel SGX and C++ for instantiation of $\mathbb{E}$, while $\mathbb{S}$ was built on top of Ethereum and Solidity. Although Aquareum can be integrated with various VMs running within $\mathbb{E}$, we selected EVM since it provides a Turing-complete execution environment and it is widely adopted in the community of decentralized applications. In detail, we utilized OpenEnclave SDK [38] and a minimalistic EVM, called eEVM [35]. However, eEVM is designed with the standard C++ map for storing the full state of L, which lacks efficient integrity-oriented operations. Moreover, eEVM assumes the unlimited size of $\mathbb{E}$ for storing the full state, while the size of $\mathbb{E}$ in SGX is constrained to ~100 MB. This might work with enabled swapping but the performance of $\mathbb{E}$ would be significantly deteriorated with a large full state. Due to these limitations, we replaced eEVM's full state handling by Merkle-Patricia Trie from Aleth [16], which we customized to support operations with the partial state. $\mathbb{O}$ and $\mathbb{C}$ were also implemented in C++.



(a) Submit TX **(b) Resolve TX**

(c) Submit Query (Get TX) **(d) Resolve Query (Get TX)**

Figure 10: Costs for resolution of censored transactions and queries.

Our implementation enables the creation and interaction of simple accounts as well as the deployment and execution of smart contracts written in Solidity. We verified the code of $\mathbb{S}$ by static/dynamic analysis tools Mythril [10], Slither [44], and ContractGuard [20]; none of them detected any vulnerabilities. The source code of our implementation will be made available upon publication of our work.

5.1 Performance Evaluation

All our experiments were performed on commodity laptop with Intel i7-10510U CPU supporting SGX v1, and they were aimed at reproducing realistic conditions – i.e., they included all operations and verifications described in Section 4, such as verification of recoverable ECDSA signatures, aggregation of transactions by Merkle tree, integrity verification of partial state, etc. We evaluated the performance of Aquareum in terms of transaction throughput per second, where we distinguished transactions with native payments (see Figure 8) and transactions with ERC20 smart contract calls (see Figure 9). All measurements were repeated 100 times, and we depict the mean and standard deviation in the graphs.

5.1.1 ***A Size of the Full State.*** The performance of Aquareum is dependent on a size of data that is copied from $\mathbb{O}$ to $\mathbb{E}$ upon call of $Exec()$. The most significant portion of the copied data is a partial state, which depends on the height of the MPT storing the full state. Therefore, we repeated our measurements with two different full states, one containing $1k$ accounts and another one containing $10k$ accounts. In the case of native payments, the full state with 10k accounts caused a decrease of throughput by 7.8%-12.1% (with enabled TB) in contrast to the full state with 1k accounts. In the case of smart contract calls, the performance deterioration was in the range 2.8%-8.4% (with enabled TB).

5.1.2 ***Block Size & Turbo Boost***. In each experiment, we varied the block size in terms of the number of transactions aggregated in the block. Initially, we performed measurements with enabled Turbo Boost (see Figure 8a and Figure 9a), where we witnessed a high throughput and its high variability. For smart contract calls (see Figure 9a), the throughput increased with the size of the block modified from 1 to 1000 by 45.7% and 38.7% for a full state with 1k and 10k accounts, respectively. However, in the case of native payments the improvement was only 4.3% and 2.8%, while the throughput was not increased monotonically with the block size.

Therefore, we experimentally disabled Turbo Boost (see Figure 8b) and observed the monotonic increase of throughput with increased block size, where the improvement achieved was 11.41% and 12.26% for a full state with 1k and 10k accounts, respectively. For completeness, we also disabled Trubo Boost in the case of smart contract calls (see Figure 9b), where the performance improvement was 20.9% and 26.7% for both full states under consideration.

5.2 Analysis of Costs

Besides the operational cost resulting from running the centralized infrastructure, Aquareum imposes costs for interaction with the public blockchain with $\mathbb{S}$ deployed. The deployment cost of $\mathbb{S}$ is $1.51M$ of gas and the cost of most frequent operation – syncing L with $\mathbb{S}$ (i.e., $PostLRoot()$) – is $33k$ of gas, which is only 33% higher than the cost of a standard Ethereum transaction.[7] For example, if L is synced with $\mathbb{S}$ every 5 minutes, $\mathbb{O}$'s monthly expenses for this operation would be $285M$ of gas, while in the case of syncing every minute, monthly expenses would be $1,425M$ of gas.[8]

5.2.1 ***Censorship Resolution***. Our mechanism for censorship resolution imposes costs on $\mathbb{C}$s submitting requests as well as for $\mathbb{O}$ resolving these requests. The cost of submitting a censored request is mainly dependent on the size of the request/response and whether $\mathbb{S}$ keeps data of a request/response in the storage (i.e., an expensive option) or whether it just emits an asynchronous event with the data (i.e., a cheap option). We measured the costs of both options and the results are depicted in Figure 10. Nevertheless, for practical usage, only the option with event emitting is feasible (see solid lines in Figure 10).

Figure 10a and Figure 10b depict the resolution of a censored transaction, which is more expensive for $\mathbb{C}$ than for $\mathbb{O}$, who resolves each censored transaction with constant cost 49k of gas (see Figure 10b). On the other hand, the resolution of censored queries is more expensive for $\mathbb{O}$ since she has to deliver a response with data to $\mathbb{S}$ (see Figure 10d), while $\mathbb{C}$ submits only a short query, e.g., get a transaction (see Figure 10c).

6 SECURITY ANALYSIS AND DISCUSSION

In this section, we demonstrate resilience of Aquareum against adversarial actions that the malicious operator $\mathcal{A}$ can perform to violate the desired properties (see Section 3.2).

Theorem 6.1. *(Correctness) $\mathcal{A}$ is unable to modify the full state of L in a way that does not respect the semantics of VM deployed in $\mathbb{E}$.*

Justification. The update of the L's state is performed exclusively in $\mathbb{E}$. Since $\mathbb{E}$ contains trusted code that is publicly known and remotely attested by $\mathbb{C}$s, $\mathcal{A}$ cannot tamper with this code. □

Theorem 6.2. *(Consistency) $\mathcal{A}$ is unable to extend L while modifying the past records of L.*

Justification. All extensions of L are performed within trusted code of $\mathbb{E}$, while utilizing the history tree [12] as a tamper evident data structure, which enables us to make only such incremental extensions of L that are consistent with L's past. □

Theorem 6.3. *(Verifiability) $\mathcal{A}$ is unable to unnoticeably modify or delete a transaction tx that was previously inserted to L using Π_N, if sync with $\mathbb{S}$ was executed anytime afterward.*

Justification. Since tx was correctly executed (Theorem 6.1) as a part of the block b_i in a trusted code of $\mathbb{E}$, $\mathbb{E}$ produced a signed version transition pair $\{h(L_{i-1}), h(L_i)\}_{\mathbb{E}}$ of L from the version $i-1$ to the new version i that corresponds to L with b_i included. $\mathcal{A}$ could either sync L with $\mathbb{S}$ immediately after b_i was appended or she could do it n versions later. In the first case, $\mathcal{A}$ published $\{h(L_{i-1}), h(L_i)\}_{\mathbb{E}}$ to $\mathbb{S}$, which updated its current version of L to i by storing $h(L_i)$ into $LRoot_{pb}$. In the second case, n blocks were appended to L, obtaining its $(i+n)$th version. $\mathbb{E}$ executed all transactions from versions $(i+1), \ldots, (i+n)$ of L, while preserving correctness (Theorem 6.1) and consistency (Theorem 6.2). Then $\mathbb{E}$ generated a version transition pair $\{h(L_{i-1}), h(L_{i+n})\}_{\mathbb{E}}$ and $\mathcal{A}$ posted it to $\mathbb{S}$, where the current version of L was updated to $i+n$ by storing $h(L_{i+n})$ into $LRoot_{pb}$. When any $\mathbb{C}$ requests tx and its proofs from $\mathcal{A}$ with regard to publicly visible $LRoot_{pb}$, she might obtain a modified tx' with a valid membership proof $\pi^{mem}_{hdr_i}$ of the block b_i but an invalid Merkle proof $\pi^{mk}_{tx'}$, which cannot be forged. □

In the case of tx deletion, $\mathcal{A}$ provides $\mathbb{C}$ with the tampered full block b'_i (maliciously excluding tx) whose membership proof $\pi^{mem}_{hdr'_i}$ is invalid – it cannot be forged. □

Theorem 6.4. *(Non-Equivocation) Assuming L synced with $\mathbb{S}$: $\mathcal{A}$ is unable to provide two distinct $\mathbb{C}$s with two distinct valid views on L.*

Justification. Since L is regularly synced with publicly visible $\mathbb{S}$, and $\mathbb{S}$ stores only a single current version of L (i.e., $LRoot_{pb}$), all $\mathbb{C}s$ share the same view on L. □

Theorem 6.5. *(Censorship Evidence) $\mathcal{A}$ is unable to censor any request (transaction or query) from $\mathbb{C}$ while staying unnoticeable.*

Justification. If $\mathbb{C}$'s request is censored, $\mathbb{C}$ asks for a resolution of the request through public $\mathbb{S}$. $\mathcal{A}$ observing the request might either ignore it and leave the proof of censoring at $\mathbb{S}$ or she might submit the request to $\mathbb{E}$ and obtain an enclave signed proof witnessing that a request was processed – this proof is submitted to $\mathbb{S}$, whereby publicly resolving the request. □

6.1 Other Properties and Implications

6.1.1 ***Privacy VS Performance***. Aquareum provides privacy of data submitted to $\mathbb{S}$ during the censorship resolution since the requests and responses are encrypted. However, Aquareum does not provide privacy against $\mathbb{O}$ who has the read access to L. Although

[7] This cost is low since we leverage the native signature scheme of the blockchain Σ_{pb}.

[8] Representing \$305 and \$1525 as of May 2020, assuming standard gas price of 5 GWEI.

Aquareum could be designed with the support of full privacy, a disadvantage of such an approach would be the performance drop caused by the decryption of requested data from L upon every $\mathbb{C}$'s read query, requiring a call of $\mathbb{E}$. In contrast, with partial-privacy, $\mathbb{O}$ is able to respond queries of $\mathbb{C}$s without touching $\mathbb{E}$.

6.1.2 *Access Control at* $\mathbb{S}$.

$\mathbb{C}$s interact with $\mathbb{S}$ only through functions for submission of censored requests. Nevertheless, access to these functions must be regulated through an access control mechanism in order to avoid exhaustion (i.e., DoS) of this functionality by external entities. This can be performed with a simple access control requiring $\mathbb{C}$s to provide access tickets when calling the functions of $\mathbb{S}$. An access ticket could be provisioned by $\mathbb{C}$ upon registration at $\mathbb{O}$, and it could contain $PK_{\mathbb{C}}^{pb}$ with a time expiration of the subscription, signed by $\mathbb{E}$. Whenever $\mathbb{C}$ initiates a censored request, verification of an access ticket would be made by $\mathbb{S}$, due to which DoS of this functionality would not be possible.

6.1.3 *Security of TEE*.

Aquareum assumes that its TEE platform is secure. However, recent research showed that this might not be the case in practical implementations of TEE, such as SGX that was vulnerable to memory corruption attacks [4] as well as side channel attacks [5, 48]. A number of software-based defense and mitigation techniques have been proposed [5, 7, 19, 42, 43] and some vulnerabilities were patched by Intel at the hardware level [25]. Nevertheless, we note that Aquareum is TEE-agnostic thus can be integrated with other TEEs such as ARM TrustZone or RISC-V architectures (using Keystone-enclave [15] or Sanctum [11]).

6.1.4 *Time to Finality*.

Many blockchain platforms suffer from accidental forks, which temporarily create parallel inconsistent blockchain views. To mitigate this phenomenon, it is recommended to wait a certain number of block confirmations after a given block is created, considering it irreversible. This waiting time (a.k.a., time to finality) influences the non-equivocation property of Aquareum, and Aquareum inherits it from the underlying blockchain platform. Most blockchains have a long time to finality, e.g., ~3mins in Bitcoin [37], ~3mins in Ethereum [49], ~2mins in Cardano [26]. However, some blockchains have a short time to finality, e.g., HoneyBadgerBFT [36], Algorand [18], and StrongChain [45]. The selection of the underlying blockchain platform (or the protocol of the consensus layer [24]) is dependent on the requirements of the particular use case that Aquareum is applied for.

7 RELATED WORK

Due to their importance and potential applications, centralized ledgers, under different names (like logs, notaries, timestamp services, etc.), were extensively investigated in the literature.

Append-Only Designs. The first line of research is around authenticated append-only data structures. Haber and Stornetta [22] proposed a hash chain associated with transactions, proving their order. Subsequently, their work was improved [3] by aggregating transactions in a Merkle tree, allowing more efficient proofs and updates. However, these constructions still require $O(n)$ messages to prove that one version of the ledger is an extension of another. Crosby and Wallach [12] introduced append-only logs with $O(\log n)$-long incremental and membership proofs. Certificate Transparency (CT) [29] deploys this data structure to create a public append-only log of digital certificates supporting efficient membership and extension proofs, but with inefficient exclusion proofs. The idea of CT's publicly verifiable logs was then extended to other applications, like revocation transparency [28], binary transparency [17], or key transparency [33]. The CT's base construction was further improved by systems combining an append-only Merkle tree with an ordered Merkle tree [27, 41] aiming to implement a variant of an authenticated append-only dictionary. Besides making all certificates visible and append-only, these constructions use a constructed key:value mapping to prove e.g., that a certificate is revoked, or that a given domain has a certain list of certificates. These systems provide more powerful properties than CT, but unfortunately, they have inefficient $O(n)$ proofs in verifying both properties of their logs at the same time (i.e., append-only ledger with the correct key:value mapping). A construction of append-only dictionaries with succinct proofs was recently proposed [46]. Despite achieving the desired properties, this construction relies on stronger cryptographic assumptions. Moreover, the scheme has efficiency bottlenecks as proving time grows with the data and as of today, it is impractical even for low transaction throughputs. The system also requires a trusted setup which may be unacceptable for many applications (like public ledgers). Finally, schemes of this class are designed for use cases specific to key:value databases, unable to handle smart contracts as of today.

Non-Equivocation Designs. Although the above systems try to minimize trust in the operator of a ledger and aim at public verifiability by deploying cryptographic constructions, they require an out of band mechanism to provide non-equivocation. One family of solutions detecting equivocations are gossip protocols, where users exchange their ledger views in order to find any inconsistencies [9, 13]. A disadvantage of these solutions is that they are primarily detective, unable to effectively prevent equivocation attacks. Moreover, these solutions are usually underspecified, and we are not aware of any system of this class deployed for this use case.

Another approach for providing the non-equivocation of a ledger was proposed by introducing multiple auditing nodes [2, 27] running a consensus protocol. Mitigations of this class, like the one proposed in Aquareum, include systems built on top of a blockchain platform (providing non-equivocation by design). An advantage of those solutions is that they are as strong as the underlying blockchain platform and with some latency (i.e., minutes) can prevent operator equivocations. Catena [47] proposes a system where a centralized log proves its non-equivocation by posting a sequence of integrity preserving transactions in the Bitcoin blockchain for its updates. However, it requires clients to obtain all Catena transactions and their number is linear with the number of log updates. PDFS [21] reduces this overhead (to constant) by a smart contract that validates consistency with the past by incremental update of the ledger using the history tree data structure (similarly, as in Aquareum); however, it does not guarantee the correct execution.

Decentralized Designs with TEE. Several systems combine TEE with blockchains, mostly with the intention to improve the lacking properties of blockchains like confidentiality or throughput bottlenecks. The most related work includes Teechain [30], a system where Bitcoin transactions can be executed off-chain in TEE enclaves. By relying on properties provided by TEE, the scheme can support secure, efficient, and scalable Bitcoin transfers. Another system is Ekiden [8], which offloads smart contract execution to dedicated TEE-supported parties. These parties can execute smart contract transactions efficiently and privately and since they are agnostic to the blockchain consensus protocol the transaction throughput can be scaled horizontally. A similar approach is taken by Das et al. [14] who propose FastKitten. In contrast to the previous work, the authors focus on backward compatibility, choosing Bitcoin as the blockchain platform and enhancing it with Turing-complete smart contracts (Bitcoin natively supports only simple smart contracts). In FastKitten, smart contracts are executed off-chain within TEE of the operator. The focus of FastKitten is the execution of multi-round smart contracts within the set of parties who interact with the operator. FastKitten supports native coins of the underlying blockchain due to SPV verification of coin locking transactions embedded into TEE. Custos [39] focuses on a detection of tampering with system logs and it utilizes TEE for the logger and decentralized auditors. However, auditors must regularly perform audit challenges to detect tampering, which is expensive and time consuming. In contrast, Aquareum provides instant efficient proofs of data genuineness or tampering upon request of the data.

8 CONCLUSION

In this paper, we proposed Aquareum, a framework for centralized ledgers, which provides verifiability, non-equivocation, and censorship evidence. To achieve these properties, we leveraged a combination of TEE and public blockchain with support for Turing-complete smart contracts. We showed that Aquareum is deployable with the current tools and is able to process over 450 transactions per second on a commodity PC, while accounting for the overhead of all verifications and updates.

REFERENCES

[1] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. 2013. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, Vol. 13. ACM New York, NY, USA.

[2] David Basin, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. 2014. ARPKI: Attack resilient public-key infrastructure. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 382–393.

[3] Dave Bayer, Stuart Haber, and W Scott Stornetta. 1993. Improving the efficiency and reliability of digital time-stamping. In *Sequences Ii*. Springer, 329–334.

[4] Andrea Biondo, Mauro Conti, Lucas Davi, Tommaso Frassetto, and Ahmad-Reza Sadeghi. 2018. The Guard's Dilemma: Efficient Code-Reuse Attacks Against Intel {SGX}. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 1213–1227.

[5] Ferdinand Brasser, Srdjan Capkun, Alexandra Dmitrienko, Tommaso Frassetto, Kari Kostiainen, Urs Müller, and Ahmad-Reza Sadeghi. 2017. DR. SGX: hardening SGX enclaves against cache attacks with data location randomization. *arXiv preprint arXiv:1709.09917* (2017).

[6] Melissa Chase, Apoorvaa Deshpande, Esha Ghosh, and Harjasleen Malvai. 2019. SEEMless: Secure End-to-End Encrypted Messaging with less Trust. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1639–1656.

[7] Sanchuan Chen, Xiaokuan Zhang, Michael K Reiter, and Yinqian Zhang. 2017. Detecting privileged side-channel attacks in shielded execution with Déjá Vu. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. 7–18.

[8] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. 2018. Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contract Execution. *arXiv preprint arXiv:1804.05141* (2018).

[9] Laurent Chuat, Pawel Szalachowski, Adrian Perrig, Ben Laurie, and Eran Messeri. 2015. Efficient gossip protocols for verifying the consistency of certificate logs. In *2015 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 415–423.

[10] ConsenSys. 2019. Mythril. (2019). https://github.com/ConsenSys/mythril

[11] Victor Costan, Ilia Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal hardware extensions for strong software isolation. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 857–874.

[12] Scott A Crosby and Dan S Wallach. 2009. Efficient Data Structures For Tamper-Evident Logging.. In *USENIX Security Symposium*. 317–334.

[13] Rasmus Dahlberg, Tobias Pulls, Jonathan Vestin, Toke Høiland-Jørgensen, and Andreas Kassler. 2018. Aggregationbased gossip for certificate transparency. *CoRR abs/1806.08817, August* (2018).

[14] Poulami Das, Lisa Eckey, Tommaso Frassetto, David Gens, Kristina Hostáková, Patrick Jauernig, Sebastian Faust, and Ahmad-Reza Sadeghi. 2019. FastKitten: practical smart contracts on bitcoin. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 801–818.

[15] Keystone Enclave. 2019. Keystone: An Open Framework for Architecting Trusted Execution Environments. https://keystone-enclave.github.io/. (2019).

[16] Ethereum. 2019. Aleth — Ethereum C++ client, tools and libraries. (2019). https://github.com/ethereum/aleth

[17] Sascha Fahl, Sergej Dechand, Henning Perl, Felix Fischer, Jaromir Smrcek, and Matthew Smith. 2014. Hey, nsa: Stay away from my market! future proofing app markets against powerful attackers. In *proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. 1143–1155.

[18] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP*.

[19] Daniel Gruss, Julian Lettner, Felix Schuster, Olya Ohrimenko, Istvan Haller, and Manuel Costa. 2017. Strong and efficient cache side-channel protection using hardware transactional memory. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 217–233.

[20] GuardStrike. 2019. ContractGuard. (2019). https://contract.guardstrike.com/

[21] Juan Guarnizo and Pawel Szalachowski. 2019. PDFS: practical data feed service for smart contracts. In *European Symposium on Research in Computer Security*. Springer, 767–789.

[22] Stuart Haber and W Scott Stornetta. 1990. How to time-stamp a digital document. In *Conference on the Theory and Application of Cryptography*. Springer, 437–455.

[23] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. 2013. Using innovative instructions to create trustworthy software solutions. *HASP@ ISCA* 11 (2013).

[24] Ivan Homoliak, Sarad Venugopalan, Qingze Hum, Daniel Reijsbergen, Richard Schumi, and Pawel Szalachowski. 2019. The Security Reference Architecture for Blockchains: Towards a Standardized Model for Studying Vulnerabilities, Threats, and Defenses. *arXiv preprint arXiv:1910.09775* (2019).

[25] Intel. 2018. Resources and Response to Side Channel L1 Terminal Fault. (2018). https://www.intel.com/content/www/us/en/architecture-and-technology/l1tf.html

[26] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*. Springer, 357–388.

[27] Tiffany Hyun-Jin Kim, Lin-Shung Huang, Adrian Perrig, Collin Jackson, and Virgil Gligor. 2013. Accountable key infrastructure (AKI) a proposal for a public-key validation infrastructure. In *Proceedings of the 22nd international conference on World Wide Web*. 679–690.

[28] Ben Laurie and Emilia Kasper. 2012. Revocation transparency. *Google Research, September* (2012), 33.

[29] Ben Laurie, Adam Langley, and Emilia Kasper. 2013. *RFC 6962 – Certificate Transparency*. RFC 6962. RFC Editor. https://www.rfc-editor.org/info/rfc6962

[30] Joshua Lind, Ittay Eyal, Florian Kelbert, Oded Naor, Peter Pietzuch, and Emin Gün Sirer. 2017. Teechain: Scalable blockchain payments using trusted execution environments. *arXiv preprint arXiv:1707.05454* (2017).

[31] David Mazieres and Dennis Shasha. 2002. Building secure file systems out of Byzantine storage. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*. 108–117.

[32] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. 2013. Innovative instructions and software model for isolated execution. *Hasp@isca* 10, 1 (2013).

[33] Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. 2015. {CONIKS}: Bringing Key Transparency to End Users. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*. 383–398.

[34] Ralph C Merkle. 1989. A certified digital signature. In *Conference on the Theory and Application of Cryptology*. Springer, 218–238.

[35] Microsoft. 2020. Enclave EVM. (2020). https://github.com/microsoft/eEVM

[36] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *ACM CCS*.

[37] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[38] Open Enclave SDK Contributors. 2020. Open Enclave SDK. (2020). https://openenclave.io/sdk/

[39] Riccardo Paccagnella, Pubali Datta, Wajih Ul Hassan, Adam Bates, Christopher W Fletcher, Andrew Miller, and Dave Tian. 2020. Custos: Practical tamper-evident auditing of operating systems using trusted execution. In *Proc. of the Symposium on Network and Distributed System Security (NDSS).*

[40] James Ray. 2019. Modified Merkle Patricia Trie Specification. https://github.com/ethereum/wiki/wiki/Patricia-Tree. (2019).

[41] Mark Dermot Ryan. 2014. Enhanced Certificate Transparency and End-to-End Encrypted Mail.. In *NDSS*. 1–14.

[42] Jaebaek Seo, Byoungyoung Lee, Seong Min Kim, Ming-Wei Shih, Insik Shin, Dongsu Han, and Taesoo Kim. 2017. SGX-Shield: Enabling Address Space Layout Randomization for SGX Programs.. In *NDSS*.

[43] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. 2017. T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs.. In *NDSS*.

[44] Slither Team. 2019. Slither. (2019). https://github.com/crytic/slither

[45] Pawel Szalachowski, Daniël Reijsbergen, Ivan Homoliak, and Siwei Sun. 2019. StrongChain: Transparent and Collaborative Proof-of-Work Consensus. In *USENIX*. 819–836.

[46] Alin Tomescu, Vivek Bhupatiraju, Dimitrios Papadopoulos, Charalampos Papamanthou, Nikos Triandopoulos, and Srinivas Devadas. 2019. Transparency logs via append-only authenticated dictionaries. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1299–1316.

[47] Alin Tomescu and Srinivas Devadas. 2017. Catena: Efficient non-equivocation via bitcoin. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 393–409.

[48] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 991–1008.

[49] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151 (2014), 1–32.

A APPENDIX

A.1 Examples of Censored Queries

While in Section 4.5 and Figure 7 we omit the details about the data that a query might fetch, here we describe two examples.

Get Transaction. In the first example, a query fetches the transaction tx identified by id_{tx} that is part of the block identified by id_{blk}.[9] Upon notification from $\mathbb{S}$ about unresolved request, $\mathbb{O}$ fetches the full block with ID equal to id_{blk},[10] computes its membership proof π_{hdr}^{mem} in the version $\#(LRoot_{pb})$ of L, and calls the function *SignQryTx*() of $\mathbb{E}$ (see Algorithm 6) with these data in the arguments. $\mathbb{E}$ verifies π_{hdr}^{mem} and search for tx with id_{tx} in the passed block. If tx is found, $\mathbb{E}$ signs encrypted tx and the positive status of the query. On the other hand, if tx is not found in the block, $\mathbb{E}$ signs the negative query status and empty data. The signature, the status, and encrypted tx are passed to $\mathbb{S}$, where the censorship of query is finished.

Get Account State. In the second example, a query fetches an account state *as* identified by id_{as} from the most recent version $\#(LRoot_{cur})$ of L. When $\mathbb{O}$ is notified by $\mathbb{S}$ about an unresolved request, $\mathbb{O}$ retrieves *as* from MPT trie storing the full global state of L, computes its MPT proof π_{as}^{mpt},[11] and calls the function *SignQryAS*() of $\mathbb{E}$ (see Algorithm 6) with these data in the arguments. $\mathbb{E}$ verifies π_{as}^{mpt} with regards to $\#(LRoot_{cur})$, and if it is a positive MPT proof, $\mathbb{E}$ signs the encrypted *as* and a positive status of the query. In contrary, if π_{as}^{mpt} is a negative MPT proof, $\mathbb{E}$ signs the negative query status and the empty data. The signature, status, and encrypted *as* are passed to $\mathbb{S}$, where the censorship of the query is completed.

Algorithm 4: The program $prog^{\mathbb{O}}$ of service operator $\mathbb{O}$

```
▷ Variables and functions of $\mathbb{O}$:
    $PK_{\mathbb{E}}^{tee}, PK_{\mathbb{E}}^{pb}$: public keys of enclave $\mathbb{E}$ (under $\Sigma_{tee}$ & $\Sigma_{pb}$),
    $PK_{\mathbb{O}}, SK_{\mathbb{O}}$: keypair of operator $\mathbb{O}$ (under $\Sigma_{pb}$),
    $prog^{\mathbb{E}}, prog^{\mathbb{S}}$: program of enclave/smart contract,
    $txs_u$: cache of unprocessed TXs,
    $blks_p^{\#}$: counter of processed blocks, not synced with PB yet,
    $\tau_{vm}, \tau_{pb}$: time of the last flush to enclave/PB,
    $state_{cur} \leftarrow \perp$: current full global state of VM,
    $censTxs \leftarrow []$: cache of posted censored TXs to $\mathbb{S}$,
    $L \leftarrow []$: data of $L$ (not synced with PB),
    $LRoot_{pb}$: the last root of $L$ flushed to PB,
    $LRoot_{cur}$: the current root of $L$ (not flushed to PB),
    $\sigma_{last}$: a signature of the last version transition pair signed by $\mathbb{E}$,
▷ Declaration of types and constants:
    Block {$hdr, txs, rcps$};
    $FL_{vm}^{\#}, FL_{pb}^{\#}$: # of txs/blocks for flushing to enclave/PB,
    $FL_{vm}^{\tau}, FL_{pb}^{\tau}$: timeout for flushing to enclave/PB,
▷ Declaration of functions:
function $Init()$
    $PK_{\mathbb{E}}^{tee}, PK_{\mathbb{E}}^{pb} \leftarrow prog^{\mathbb{E}}.Init()$;
    $prog^{\mathbb{S}}.Init(PK_{\mathbb{E}}^{pb}, PK_{\mathbb{E}}^{tee}, PK_{\mathbb{O}})$;

function $UponRecvTx(tx)$
    assert accessControl($tx$);
    $txs_u$.add($tx$);
    if $|txs_u| = FL_{vm}^{\#} \vee now() - FL_{vm}^{\tau} \geq \tau_{vm}$ then
        $\pi_{next}^{inc}, LRoot_{tmp} \leftarrow nextIncProof()$;
        $LRoot_{pb}, LRoot_{cur}, \partial st^{new}, hdr, rcps, txs_{er}, \sigma_{last} \leftarrow$
          $prog^{\mathbb{E}}.Exec(txs_u, \partial state^{cur}, \pi_{next}^{inc}, LRoot_{tmp})$;
        $state_{cur}.update(\partial st_{new})$;
        $L$.add(Block($hdr, txs_u \setminus txs_{er}, rcps$));
        $txs_u \leftarrow []$; $blks_p^{\#} \leftarrow blks_p^{\#} + 1$ ;
                    ▷ Sync with $\mathbb{S}$ on public blockchain
        if $blks_p^{\#} = FL_{pb}^{\#} \vee now() - FL_{pb}^{\tau} \geq \tau_{pb}$ then
            $prog^{\mathbb{S}}.PostLRoot(LRoot_{pb}, LRoot_{cur}, \sigma_{last})$;
            $prog^{\mathbb{E}}.Flush()$;
            $resolveCensTxs()$;
            $blks_p^{\#} \leftarrow 0$ ;

function $nextIncProof()$
    $LRoot_{tmp} \leftarrow L.add($Block$(\perp, [], []))$;
    $\pi_{next}^{inc} \leftarrow L.IncProof(LRoot_{cur}, LRoot_{tmp})$ ;
    $L.deleteLastBlock()$;
    return $\pi_{next}^{inc}, LRoot_{tmp}$;        ▷ It serves as an incr. proof template for $\mathbb{E}$.
function $RestoreFailedEnc()$
    $hdr_{sync} \leftarrow L[\#(LRoot_{pb}) - 1].hdr$;
    $blks_{unsync} \leftarrow L[\#(LRoot_{pb}) : -1]$ ;
    $L.restore(\#(LRoot_{pb}))$;        ▷ Restore all data to the target version.
    $LRoot_A, LRoot_B, \sigma, PK_{pb}^{\mathbb{E}}, PK_{tee}^{\mathbb{E}} \leftarrow$
      $prog^{\mathbb{E}}.ReInit(LRoot_{pb}, blks_{unsync}, hdr_{sync})$;
    assert $LRoot_{cur} = LRoot_A$;        ▷ $\mathbb{E}$ and $\mathbb{O}$ run VM into the same point.
    $prog^{\mathbb{S}}.ReplaceEnc(PK_{pb}^{\mathbb{E}}, PK_{tee}^{\mathbb{E}}, LRoot_A, LRoot_B, \sigma)$;
```

[9] To verify whether the block with id_{blk} exists, we check $id_{blk} \leq \#(LRoot_{pb})$.

[10] Note that a full block is required to pass into $\mathbb{E}$ since Merkle tree (aggregating transactions) does not support exclusion proofs, and thus all transactions of the block need to be compared.

[11] If as is not found, π_{as}^{mpt} serves as a negative proof of as.

Algorithm 5: Censorship resolution in $\mathbb{O}$ (part of $prog^{\mathbb{O}}$)

```
function $UponPostedCensTX(etx, idx_{req})$
    $tx \leftarrow prog^{\mathbb{E}}.Decrypt(etx)$;
    $censTxs.add(\{tx, etx, idx_{req}\})$;
    $UponRecvTx(tx)$;    ▹ Delay response until the current block is finished.
function $UponPostedCensQry(equery, idx_{req})$
    $qry \leftarrow parse(prog^{\mathbb{E}}.Decrypt(equery))$;
    if $READ\_TX = qry.type$ then
        $blk \leftarrow getBlockById(qry.id_{blk})$;
        $\pi_{hdr}^{mem} \leftarrow L.MemProof(blk.hdr.ID, LRoot_{pb})$;
        $\sigma, status, edata \leftarrow prog^{\mathbb{E}}.SignQryTx(equery, blk, \pi_{hdr}^{mem})$;
    else if $READ\_AS = qry.type$ then
        $as \leftarrow state_{cur}.get(qry.id_{as})$;
        $\pi_{as}^{mpt} \leftarrow L.MptProof(qry.id_{as})$;    ▹ Inclusion/exclusion proof.
        $\sigma, status, edata \leftarrow prog^{\mathbb{E}}.SignQryAS(equery, as, \pi_{as}^{mpt})$;
    $prog^{\mathbb{S}}.ResolveCensQry(idx_{req}, status, edata, \sigma)$;
function $resolveCensTxs()$
    for $\{ct : censTxs\}$ do
        $blk \leftarrow getBlockOfTx(h(ct.tx), L)$;
        $\pi_{hdr}^{mem} \leftarrow L.MemProof(blk.hdr.ID, LRoot_{pb})$;
        $\pi_{tx}^{mk} \leftarrow MkProof(ct.tx, blk.txs)$;
        $\sigma, status \leftarrow prog^{\mathbb{E}}.SignTx(ct.etx, \pi_{tx}^{mk}, blk.hdr, \pi_{hdr}^{mem})$;
        $prog^{\mathbb{S}}.ResolveCensTx(ct.idx_{req}, status, \sigma)$;
    $censTxs \leftarrow []$;
```

Algorithm 6: Censorship resolution in $\mathbb{E}$ (part of $prog^{\mathbb{E}}$).

```
function $Decrypt(edata)$ public
    $data \leftarrow \Sigma_{pb}.Decrypt(SK_{\mathbb{E}}^{pb}, edata)$;
    Output($data$);
function $SignTx(etx, \pi_{tx}^{mk}, hdr, \pi_{hdr}^{mem})$ public
            ▹ Resolution of a censored write tx.
    $tx \leftarrow \Sigma_{pb}.Decrypt(SK_{\mathbb{E}}^{pb}, etx)$;
    if $ERROR = parse(tx)$ then
        $status$ = PARSING_ERROR;
    else if $ERROR = \Sigma_{pb}.Verify((tx.\sigma, tx.PK_{\mathbb{C}}^{pb}), tx)$ then
        $status$ = SIGNATURE_ERROR;
    else
                ▹ Verify proofs binding TX to header and header to L.
        assert $\pi_{tx}^{mk}$.Verify($tx, hdr.txsRoot$);
        assert $\pi_{hdr}^{mem}$.Verify($hdr.ID, hdr, LRoot_{pb}$);
        $status \leftarrow$ INCLUDED;
                    ▹ TX was processed, so $\mathbb{E}$ can issue a proof.
    $\sigma \leftarrow \Sigma_{pb}.sign(SK_{\mathbb{E}}^{pb}, (h(etx), status))$;
    Output($\sigma, status$);
function $SignQryTx(equery, blk, \pi_{hdr}^{mem})$ public
            ▹ Resolution of a censored read tx query.
    $\ldots, id_{tx}, id_{blk}, PK_{\mathbb{C}}^{pb} \leftarrow parse(Decrypt(equery))$;
    if $id_{blk} > \#(LRoot_{pb})$ then
        $status \leftarrow$ BLK_NOT_FOUND, $edata \leftarrow \perp$;
    else
        assert $\pi_{hdr}^{mem}$.Verify($blk.hdr.ID, blk.hdr, LRoot_{pb}$);
        assert VerifyBlock(blk);    ▹ Full check of block consistency.
        $tx \leftarrow$ findTx($id_{tx}, blk.txs$);
        if $\perp = tx$ then
            $status \leftarrow$ TX_NOT_FOUND, $edata \leftarrow \perp$;
        else
            $status \leftarrow$ OK, $edata \leftarrow \Sigma_{pb}.Encrypt(PK_{\mathbb{C}}^{pb}, tx)$;
    $\sigma \leftarrow \Sigma_{pb}.sign(SK_{\mathbb{E}}^{pb}, (h(equery), status, edata))$;
    Output($\sigma, status, edata$);
function $SignQryAS(equery, as, \pi_{as}^{mpt})$ public
            ▹ Resolution of a censored read account state query.
    $\ldots, id_{as}, PK_{\mathbb{C}}^{pb} \leftarrow parse(Decrypt(equery))$;
    if $\perp = as$ then
        assert $\pi_{as}^{mpt}.VerifyNeg(id_{as}, LRoot_{cur})$;
        $status \leftarrow$ NOT_FOUND, $edata \leftarrow \perp$;
    else
        assert $\pi_{as}^{mpt}.Verify(id_{as}, LRoot_{cur})$;
        $status \leftarrow$ OK, $edata \leftarrow \Sigma_{pb}.Encrypt(PK_{\mathbb{C}}^{pb}, as)$;
    $\sigma \leftarrow \Sigma_{pb}.sign(SK_{\mathbb{E}}^{pb}, (h(equery), status, h(edata)))$;
    Output($\sigma, status, edata$);
```

CBDC-AquaSphere: Interoperable Central Bank Digital Currency Built on Trusted Computing and Blockchain

Ivan Homoliak,[†] Martin Perešíni,[†] Patrik Holop,[†] Jakub Handzuš,[†] Fran Casino[*]

[†]Brno University of Technology, Faculty of Information Technology, Czech Republic
[*]Universitat Rovira i Virgili, Tarragona, Spain

ABSTRACT

The adoption of decentralized, tamper-proof ledger systems is paving the way for new applications and opportunities in different contexts. While most research aims to improve their scalability, privacy, and governance issues, interoperability has received less attention. Executing transactions across various blockchains is notably instrumental in unlocking the potential of novel applications, particularly in the financial sector, where their potential would otherwise be significantly diminished. Therefore, interoperable ledgers are crucial to ensure the expansion and further adoption of such a technology in various contexts.

In this paper, we present a protocol that uses a combination of trusted execution environment (TEE) and blockchains to enable interoperability over independent semi-centralized CBDC ledgers, guaranteeing the atomicity of inter-bank transfers. Our interoperability protocol uses a custom adaptation of atomic swap protocol and is executed by any pair of CBDC instances to realize a one-way transfer. It ensures features such as atomicity, verifiability, correctness, censorship resistance, and privacy while offering high scalability in terms of the number of CBDC instances. Our approach enables to possible deployment scenarios that can be combined: (1) CBDC instances represent central banks of multiple countries, and (2) CBDC instances represent the set of retail banks and a paramount central bank of a single country. We provide a detailed description of our protocol as well as an extensive analysis of its benefits, features, and security.

In this WIP paper, we made a proof-of-concept implementation and made a partial evaluation, while the more extensive evaluation will be made in our future work.

KEYWORDS

Blockchain, Interoperability, Central Bank Digital Currency (CBDC), Trusted Execution Environment, Cross-chain Protocol, Privacy, Censorship.

1 INTRODUCTION

Blockchain technology is becoming the backbone of a myriad of applications since it provides features such as decentralization, immutability, availability, and transparency. More recently, along with the increasing adoption and maturity of such a technology [1], central banks all over the world are accelerating the process of Central Bank Digital Currency (CBDC) development [71]. CBDC has received increasing attention in the past few years. More than 85% of central banks are actively researching the potential for CBDCs, and according to BIS survey [11] conducted in 2021 central banks covering 20% of the world's population are likely to launch retail CBDCs before 2025. Some of the reasons behind this new paradigm are the digitization of the economy, the level of development of the financial sector, and a strong decline in the use of cash [57].

Despite the generalized will to improve the worldwide financial system by utilizing blockchain technology [27] in centralized environments, there is still a road ahead for the realization of fast secure blockchain payment systems. Nevertheless, some features are essential to enable financial solutions to reach an operational level, making interoperability a crucial requirement in this context. Note that a few cross-chain solutions and protocols [7, 43, 71] that leverage the necessary level of interoperability for execution of inter-bank financial transactions have been proposed in the literature. In this regard, technologies such as Trusted Execution Environments (TEE) in a potential combination with blockchains can efficiently enforce the required security and privacy levels of centralized environments of banks, and thus provide a high level of trustworthiness for the end users.

Motivation. CBDC legislation and adoption goes in hand with privacy and security concerns. The centralized nature of banks implies that transactions are recorded in private ledgers managed by banks, contrary to the very nature of public decentralized cryptocurrencies. While this may prevent some potential malicious scenarios, users are forced into trusting a single authority and its corresponding regulations. Aiming at increasing decentralization and trust, several authors have proposed the use of TEE to leverage verifiable protocols enabling interoperability of multiple centralized isolated environments [41, 69].

While achieving blockchain interoperability is challenging regardless of its flavor (i.e., between centralized, decentralized or hybrid blockchain structures) additional features such as scalability, confidentiality, and censorship resistance are necessary to guarantee for practical scenarios. Digital Euro Association released on October 2022 the CBDC manifesto [26], in which they highlight important features of CBDC, such as strong value proposition for the end users, the highest degree of privacy, and interoperability. Our work is inline with this manifesto and adheres to its features while it provides even additional features that are interesting for the users and the whole ecosystem assuming that the CBDC-equipped bank might potentially be an untrusted entity.

Contributions. In this paper, we present a practical blockchain interoperability protocol that integrates such features. On top of the above mentioned features, to the best of our knowledge, our work represents the first TEE-based interoperable CBDC approach that provides the proof-of-censorship. Our main contributions are summarized as follows:

(1) We specify requirements for an instance of CBDC that is controlled by a single bank[1] and forms an isolated environment. These requirements include high processing performance, transparent token issuance, correctness of intra-bank transfers, immutability of historical data, non-equivocation, privacy, and the indisputable proofs of censorship.
(2) We investigate state-of-the-art approaches applicable for a CBDC instance assuming our requirements and identify the most convenient one, Aquareum [33], that we further base on. The Aquareum-based CBDC ledger ensures immutability, non-equivocation, privacy, and the indisputable proofs of transaction censorship by utilizing a permissionless blockchain (e.g., Ethereum). Next, by using TEE (e.g., Intel SGX), it ensures the correctness of any transaction execution.
(3) Our main contribution resides in a design and implementation of a protocol resolving interoperability over multiple instances of semi-centralized CBDC, which guarantees the atomicity of inter-bank transfers.
(4) We provide a security analysis, to prove the the properties of our approach.

Organization. The remainder of the article is organized as follows. In Section 2, we provide a background on blockchain, atomic swap, and trusted computing. We define the problem in Section 3, where we describe the attacker model and required features of a single CBDC instance as well as the environment of multiple interoperable CBDC instances. Section 4 provides a description of the proposed interoperability protocol and its deployment scenarios. Next, the implementation details of the designed protocol and its partial evaluation are described in Section 5. We make a security analysis of our approach in Section 6. Section 7 reviews the state of the art of CBDC approaches and TEE-based blockchain solutions. Section 8 discusses the benefits and limitations of our approach. We conclude the paper in Section 9 with some final remarks.

2 BACKGROUND

This section provides the reader with the essential context needed to understand the topics that will be discussed in this article.

2.1 Blockchain

Blockchain is a tamper-resistant data structure, in which data records (i.e., *blocks*) are linked using a cryptographic hash function, and each new block has to be agreed upon by participants (a.k.a., *miners*) running a consensus protocol (i.e., *consensus nodes*). Each block may contain data records representing orders that transfer tokens, application codes written in a platform-supported language, and the execution orders of such application codes. These application codes are referred to as *smart contracts*, and they encode arbitrary processing logic written in a supported language of a smart contract platform. Interactions between clients and the smart contract platform are based on messages called *transactions*.

2.2 Trusted Execution Environment

Trusted Execution Environment (TEE) is a hardware-based component that enables secure (remote) execution [65] of a pre-defined code (i.e., enclave) in an isolated environment. TEE uses cryptographic primitives and hardware-embedded secrets that protect data confidentiality and the integrity of computations. In particular, the adversary model of TEE involves the operating system (OS) that may compromise user-space applications but not TEE-protected applications. An enclave process cannot execute system calls but can read and write memory outside the enclave. Thus isolated execution in TEE may be viewed as an ideal model in which a process is guaranteed to be executed correctly with ideal confidentiality, while it might run on a potentially malicious OS.

Intel SGX. While there exist multiple instances of TEE, in the context of this work we will focus on Intel SGX (Software Guard Extensions) [3, 32, 47]. Intel SGX allows a local process or a remote system to securely communicate with the enclave as well as execute verification of the integrity of the enclave's code. When an enclave is created, the CPU outputs a report of its initial state, also referred to as a *measurement*, which is signed by the private key of TEE and encrypted by a public key of Intel Attestation Service (IAS). The hardware-protected signature serves as the proof that the measured code is running in an SGX-protected enclave, while the encryption by IAS public key ensures that the SGX-equipped CPU is genuine and was manufactured by Intel. This proof is also known as a *quote* or *attestation*, and it can be verified by a local process or by a remote system. The enclave-provided public key can be used by a verifier to establish a secure remote channel with the enclave or to verify the signature during the attestation.

2.3 CBDC

CBDC is often defined as a digital liability backed and issued by a central bank that is widely available to the general public. CBDC encompasses many potential benefits such as efficiency and resiliency, flexible monetary policies, and enables enhanced control of tax evasion and money laundering [38]. However, regulations, privacy and identity management issues, as well as design vulnerabilities are potential risks that are shared with cryptocurrencies. Many blockchain-based CBDC projects rely on using some sort of stable coins adapting permissioned blockchains due to their scalability and the capability to establish specific privacy policies, as compared to public blockchains [61, 71]. Therefore, the level of decentralization and coin volatility are two main differences between blockchain-based CBDCs and common cryptocurrencies. These CBDCs are often based on permissioned blockchain projects such as Corda [14], variants of Hyperledger [34], and Quorum [29].

CDBC solutions are often designed as multi-layer projects [36]. Wholesale CBDC targets communication of financial institutions and inter-bank settlements. Retail CBDC includes accessibility to the general public or their customers.

2.4 Atomic Swap

A basic atomic swap assumes two parties $\mathbb{A}$ and $\mathbb{B}$ owning crypto-tokens in two different blockchains. $\mathbb{A}$ and $\mathbb{B}$ wish to execute cross-chain exchange atomically and thus achieve a *fairness* property,

[1] As we will see it can be a central bank or even a retail bank, depending on the deployment scenario described in Section 4.2.1. Therefore, depending on the scenario, we well use the term CBDC instance even for a retail bank.

i.e., either both of the parties receive the agreed amount of crypto-tokens or neither of them. First, this process involves an agreement on the amount and exchange rate, and second, the execution of the exchange itself.

In a centralized scenario [49], the approach is to utilize a trusted third party for the execution of the exchange. In contrast to the centralized scenario, blockchains allow us to execute such an exchange without a requirement of the trusted party. The atomic swap protocol [10] enables conditional redemption of the funds in the first blockchain to $\mathbb{B}$ upon revealing of the hash pre-image (i.e., secret) that redeems the funds on the second blockchain to $\mathbb{A}$. The atomic swap protocol is based on two Hashed Time-Lock Contracts (HTLC) that are deployed by both parties in both blockchains.

Although HTLCs can be implemented by Turing-incomplete smart contracts with support for hash-locks and time-locks, for clarity, we provide a description assuming Turing-complete smart contracts, requiring four transactions:

(1) $\mathbb{A}$ chooses a random string x (i.e., a secret) and computes its hash $h(x)$. Using $h(x)$, $\mathbb{A}$ deploys $HTLC_{\mathbb{A}}$ on the first blockchain and sends the agreed amount to it, which later enables anybody to do a conditional transfer of that amount to $\mathbb{B}$ upon calling a particular method of $HTLC_{\mathbb{A}}$ with $x = h(x)$ as an argument (i.e., hash-lock). Moreover, $\mathbb{A}$ defines a time-lock, which, when expired, allows $\mathbb{A}$ to recover funds into her address by calling a dedicated method: this is to prevent aborting of the protocol by another party.

(2) When $\mathbb{B}$ notices that $HTLC_{\mathbb{A}}$ has been already deployed, she deploys $HTLC_{\mathbb{B}}$ on the second blockchain and sends the agreed amount there, enabling a conditional transfer of that amount to $\mathbb{A}$ upon revealing the correct pre-image of $h(x)$ ($h(x)$ is visible from already deployed $HTLC_{\mathbb{A}}$). $\mathbb{B}$ also defines a time-lock in $HTLC_{\mathbb{B}}$ to handle abortion by $\mathbb{A}$.

(3) Once $\mathbb{A}$ notices deployed $HTLC_{\mathbb{B}}$, she calls a method of $HTLC_{\mathbb{B}}$ with revealed x, and in turn, she obtains the funds on the second blockchain.

(4) Once $\mathbb{B}$ notices that x was revealed by $\mathbb{A}$ on the second blockchain, she calls a method of $HTLC_{\mathbb{A}}$ with x as an argument, and in turn, she obtains the funds on the first blockchain.

If any of the parties aborts, the counter-party waits until the time-lock expires and redeems the funds.

2.5 Merkle Tree

A Merkle tree [48] is a data structure based on the binary tree in which each leaf node contains a hash of a single data block, while each non-leaf node contains a hash of its concatenated children. Hence, the root node provides a tamper-evident integrity snapshot of the tree contents. A Merkle tree enables efficient membership verification (with logarithmic time/space complexity) using the Merkle *proof*. To enable a membership verification of element x_i in the list X, the Merkle tree supports the following operations:

$\mathbf{MkRoot(X) \rightarrow Root}$: an aggregation of all elements of the list X by a Merkle tree, providing a single value *Root*.

$\mathbf{MkProof(x_i, X) \rightarrow \pi^{mk}}$: a Merkle proof generation for the ith element x_i present in the list of all elements X.

$\pi^{\mathbf{mk}}\mathbf{.Verify(x_i, Root) \rightarrow \{T, F\}}$: verification of the Merkle proof π^{mk}, witnessing that x_i is included in the list X that is aggregated by the Merkle tree with the root hash *Root*.

2.6 History Tree

A Merkle tree has been primarily used for proving membership. However, Crosby and Wallach [22] extended its application for an append-only tamper-evident log, named a *history tree*. In history tree, leaf nodes are added in an append-only fashion while it enables to produce incremental proofs witnessing that arbitrary two versions of the tree are consistent. The history tree brings a versioned computation of hashes over the Merkle tree, enabling to prove that different versions (i.e., commitments) of a log, with distinct root hashes, make consistent claims about the past. The history tree L supports the following operations:

$\mathbf{L.add(x) \rightarrow C_j}$**:** appending of the record x to L, returning a new commitment C_j that represents the most recent value of the root hash of the history tree.

$\mathbf{L.IncProof(C_i, C_j) \rightarrow \pi^{inc}}$**:** an incremental proof generation between two commitments C_i and C_j, where $i \leq j$.

$\mathbf{L.MemProof(i, C_j) \rightarrow \pi^{mem}}$**:** a membership proof generation for x_i from the commitment C_j, where $i \leq j$.

$\pi^{\mathbf{inc}}\mathbf{.Verify(C_i, C_j) \rightarrow \{T, F\}}$**:** verification of the incremental proof π^{inc}, witnessing that the commitment C_j contains the same history of records $x_k, k \in \{0, \ldots, i\}$ as the commitment C_i, where $i \leq j$.

$\pi^{\mathbf{mem}}\mathbf{.Verify(i, x_i, C_j) \rightarrow \{T, F\}}$**:** verification of the membership proof π^{mem}, witnessing that x_i is the ith record in the jth version of L, fixed by the commitment C_j, $i \leq j$.

$\pi^{\mathbf{inc}}\mathbf{.ReduceRoot() \rightarrow C_j}$**:** a reduction of the commitment C_j from the incremental proof π^{inc} that was generated by *L.IncProof*(C_i, C_j).

2.7 Aquareum

Aquareum [33] is a centralized ledger that is based on a combination of a trusted execution environment (TEE) with a public blockchain platform (see our other submission with ID #83). It provides a publicly verifiable non-equivocating censorship-evident private ledger. Aquareum is integrated with a Turing-complete virtual machine (instantiated by eEVM [50]), allowing arbitrary transaction processing logic, such as transfers or client-specified smart contracts. In other words, Aquareum provides most of the blockchain features while being lightweight and cheap in contrast to them. Nevertheless, Aquareum does not provide extremely high availability (such as blockchains) due to its centralized nature, which is, however, common and acceptable for the environment of CBDC.

The overview of Aquareum is depicted in Figure 1 (where parts in red are our modifications and are irrelevant for the current description). In Aquareum, clients $\mathbb{C}$s submit transactions to operator $\mathbb{O}$ (1), who executes them in protected TEE enclave $\mathbb{E}$ (4) upon fetching a few data of the ledger L with the partial state containing only concerned accounts (2). $\mathbb{E}$ outputs updated state of affected client accounts with execution receipts and *a version transition pair* of L (5) that is periodically submitted to the smart contract $\mathbb{S}$ deployed on a public blockchain (7). $\mathbb{S}$ verifies $\mathbb{E}$'s signature and the consistency of the previous version of L with $\mathbb{S}$'s local snapshot (8)

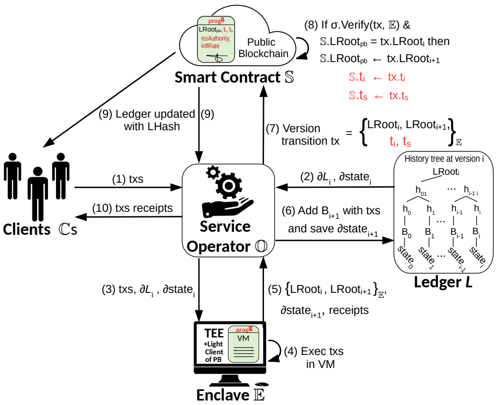

prog$\mathbb{S}$
LRoot$_{pb}$, t_i, t_s
issAuthority, inflRate
Public Blockchain
Smart Contract $\mathbb{S}$
(8) If σ.Verify(tx, $\mathbb{E}$) &
$\mathbb{S}$.LRoot$_{pb}$ = tx.LRoot$_i$ then
$\mathbb{S}$.LRoot$_{pb}$ ← tx.LRoot$_{i+1}$
$\mathbb{S}.t_i$ ← tx.t_i
$\mathbb{S}.t_s$ ← tx.t_s
(9) Ledger updated with LHash
(9)
(7) Version transition tx = $\{$LRoot$_i$, LRoot$_{i+1}$, t_i, $t_s\}_{\mathbb{E}}$
(1) txs
(10) txs receipts
Clients $\mathbb{C}$s
Service Operator $\mathbb{O}$
(2) ∂L_i , ∂state$_i$
(6) Add B$_{i+1}$ with txs and save ∂state$_{i+1}$
History tree at version i
LRoot$_i$
h_{01}
$h_{i-1\ i}$
h_0
h_1
h_{i-1}
h_i
B_0
B_1
B_{i-1}
B_i
state$_0$
state$_1$
state$_{i-1}$
state$_i$
Ledger L
(3) txs, ∂L_i , ∂state$_i$
(5) $\{$LRoot$_i$, LRoot$_{i+1}\}_{\mathbb{E}}$, ∂state$_{i+1}$, receipts
TEE +Light Client of PB
prog$\mathbb{E}$
VM
(4) Exec txs in VM
Enclave $\mathbb{E}$

Figure 1: Architecture of Aquareum with our modifications in red.

before updating the snapshot to a new version. Note that snapshot is represented by the root hash (i.e., $LRoot$) of the history tree of L.

3 PROBLEM DEFINITION

Our goal is to propose a CBDC approach that respects the features proposed in DEA manifesto [26] released in 2022, while on top of it, we assume other features that might bring more benefits and guarantees. First, we start with a specification of the desired features related to a single instance of CBDC that we assume is operated by a single entity (further a bank or its operator) that maintains its ledger. Later, we describe desired features related to multiple instances of CBDC that co-exist in the ecosystem of wholesale and/or retail CBDC.[2] In both cases, we assume that a central bank might not be a trusted entity. All features that respect this assumption are marked with asterisk * and are considered as requirements for such an attacker model.

3.1 Single Instance of CBDC

When assuming a basic building block of CBDC – a single bank's CBDC working in an isolated environment from the other banks – we specify the desired features of CBDC as follows:

Correctness of Operation Execution*: The clients who are involved in a monetary operation (such as a transfer) should be guaranteed with a correct execution of their operation.

Integrity*: The effect of all executed operations made over the client accounts should be irreversible, and no "quiet" tampering of the data by a bank should be possible. Also, no conflicting transactions can be (executed and) stored by the CBDC instance in its ledger.

Verifiability*: This feature extends integrity and enables the clients of CBDC to obtain easily verifiable evidence that the ledger they interact with is internally correct and consistent. In particular, it means that none of the previously inserted transactions was neither modified nor deleted.

Non-Equivocation*: From the perspective of the client's security, the bank should not be able to present at least two inconsistent views on its ledger to (at least) two distinct clients who would accept such views as valid.

Censorship Evidence*: The bank should not be able to censor a client's request without leaving any public audit trails proving the censorship occurrence.

Transparent Token Issuance*: Every CBDC-issued token should be publicly visible (and thus audit-able) to ensure that a bank is not secretly creating token value "out-of-nothing," and thus causing uncontrolled inflation. The transparency also holds for burning of existing tokens.

High Performance: A CBDC instance should be capable of processing a huge number of transactions per second since it is intended for daily usage by thousands to millions of people.

Privacy: All transfers between clients as well as information about the clients of CBDC should remain private for the public and all other clients that are not involved in particular transfers. However, a bank can access this kind of information and potentially provide it to legal bodies, if requested.

3.2 Multiple Instances of CBDC

In the case of multiple CBDC instances that can co-exist in a common environment, we extend the features described in the previous listing by features that are all requirements:

Interoperability*: As a necessary prerequisite for co-existence of multiple CBDC instances, we require them to be mutually interoperable, which means that tokens issued by one bank can be transferred to any other bank. For simplicity, we assume that all the CBDC instances are using the unit token of the same value within its ecosystem.[3] At the hearth of interoperability lies atomicity of supported operations. Atomic interoperability, however, requires means for accountable coping with censorship and recovery from stalling. We specify these features in the following.

Atomicity*: Any operation (e.g., transfer) between two interoperable CBDC instances must be either executed completely or not executed at all. As a consequence, no new tokens can be created out-of-nothing and no tokens can be lost in an inter-bank operation. Note that even if this would be possible, the state of both involved instances of CBDC would remain internally consistent; therefore, consistency of particular instances (Section 3.1) is not a sufficient feature to ensure atomicity within multiple interoperable CBDC instances. This requirement is especially important due to trustless assumption about particular banks, who might act in their benefits even for the cost of imposing the extreme inflation to the whole system.[4]

Inter-CBDC Censorship Evidence*: Having multiple instances of CBDC enables a different way of censorship, where one CBDC (and its clients) might be censored within some inter-CBDC operation with another

[2] Note that we will propose two deployment scenarios (see Section 4.2.1), one for the wholesale environment and the second one for the retail environment of multiple retail banks interacting with a single central bank.

[3] On the other hand, conversions of disparate CBDC-backed tokens would be possible by following trusted oracles or oracle networks.

[4] For example, if atomicity is not enforced, one bank might send the tokens to another bank, while not decreasing its supply due to pretended operation abortion.

CBDC instance, precluding them to finish the operation. Therefore, there should exist a means how to accountably detect this kind of censorship as well.

Inter-CBDC Censorship Recovery*: If the permanent censorship happens and is indisputably proven, it must not impact other instances of CBDC, including the ones that the inter-CBDC operations are undergoing. Therefore, the interoperable CBDC environment should provide a means to recover from inter-CBDC censorship of unfinished operations.

Identity Management of CBDC Instances*: Since we assume that CBDC instances are trustless, in theory, there might emerge a fake CBDC instance, pretentding to act as a valid one. To avoid this kind of situation, it is important for the ecosystem of wholesale CBDC to manage identities of particular valid CBDC instances in a secure manner.

3.3 Adversary Model

The attacker can be represented by the operator of a bank or the client of a bank, and their intention is to break functionalities that are provided by the features described above. Next, we assume that the adversary cannot undermine the cryptographic primitives used, the blockchain platform, and the TEE platform deployed.

4 PROPOSED CBDC APPROACH

We propose a holistic approach for the ecosystem of wholesale and/or retail CBDC, which aims at meeting the features described in Section 3. To accomplish these features, we leverage interesting properties stemming from a combination of a public blockchain (with smart contract platform) and TEE. Such a combination was proposed for various purposes in related work (see Section 7), out of which the use case of generic centralized ledger Aquareum [33] is most convenient to build on. Therefore, we utilize Aquareum as a building block for a single instance of CBDC, and we make a few CBDC-specific modifications to it, enhancing its transparency and functionality. Our modifications are outlined in Figure 1 by red color, while the details of them (especially changes in programs of smart contract and enclave) will be described in this section. First, we start by a description of a single CBDC instance and then we extend it to a fully interoperable environment consisting of multiple CBDC instances.

Note that in this paper, we focus solely on the transfer of tokens operation within the context of CBDC interoperability. However, our approach could be extended to different operations, involving inter-CBDC smart contract invocations. Also, note that to distinguish between smart contracts on a public blockchains and smart contracts running in TEE, we will denote latter as **micro contracts** (or μ-contracts). Similarly, we denote transactions sent to TEE as **micro transactions** (or μ-transactions) and blocks created in the ledger of CBDC instance as **micro blocks** (or μ-blocks).

4.1 A CBDC Instance

Alike in Aquareum, the primary entity of each CBDC instance is its operator $\mathbb{O}$ (i.e., a bank), who is responsible for (1) maintaining the ledger L, (2) running the TEE enclave $\mathbb{E}$, (3) synchronization of the L's snapshot to a public blockchain with smart contract $\mathbb{IPSC}$ ($\mathbb{I}$ntegrity $\mathbb{P}$reserving $\mathbb{S}$mart $\mathbb{C}$ontract), (4) resolving censorship requests, and (5) a communication with clients $\mathbb{C}$s.

4.1.1 *Token Issuance.*

On top of Aquareum's $\mathbb{S}$, our $\mathbb{IPSC}$ contains snapshotting of the total issued tokens t_i by the current CBDC instance and the total supply t_s available at the instance for the purpose of transparency in token issuance (and potentially even burning). Therefore, we extend the $\mathbb{E}$-signed version transition pair periodically submitted to $\mathbb{IPSC}$ by these two fields that are relayed to $\mathbb{IPSC}$ upon snapshotting L (see red text in Figure 1). Notice that $t_i = t_s$ in the case of a single instance since the environment of the instance is isolated.

An Inflation Bound. Although snapshotting the total tokens in circulation is useful for the transparency of token issuance, $\mathbb{O}$ might still hyper-inflate the CBDC instance. Therefore, we require $\mathbb{O}$ to guarantee a maximal inflation rate i_r per year, which can be enforced by $\mathbb{IPSC}$ as well as $\mathbb{E}$ since the code of both is publicly visible and attestable. The i_r should be adjusted to a constant value by $\mathbb{O}$ at the initialization of $\mathbb{IPSC}$ and verified every time the new version of L is posted to $\mathbb{IPSC}$; in the case of not meeting the constrain, the new version would not be accepted at $\mathbb{IPSC}$. However, another possible option is that the majority vote of $\mathbb{C}$s can change i_r even after initialization. Besides, $\mathbb{E}$ also enforces i_r on t_i and does not allow $\mathbb{O}$ to issue yearly more tokens than defined by i_r. Nevertheless, we put the inflation rate logic also into $\mathbb{IPSC}$ for the purpose of transparency.

4.1.2 *Initialization.*

First, $\mathbb{E}$ with program $prog^{\mathbb{E}}$ (see Algorithm 5 of Appendix) generates and stores two key pairs, one under Σ_{pb} (i.e., $SK_{\mathbb{E}}^{pb}$, $PK_{\mathbb{E}}^{pb}$) and one under Σ_{tee} (i.e., $SK_{\mathbb{E}}^{tee}$, $PK_{\mathbb{E}}^{tee}$). Then, $\mathbb{O}$ generates one key pair under Σ_{pb} (i.e., $SK_{\mathbb{O}}^{pb}$, $PK_{\mathbb{O}}^{pb}$), which is then used as the sender of a transaction deploying $\mathbb{IPSC}$ with program $prog^{\mathbb{IPSC}}$ (see Algorithm 6 of Appendix) at public blockchain with parameters $PK_{\mathbb{E}}^{pb}$, $PK_{\mathbb{E}}^{tee}$, $PK_{\mathbb{O}}^{pb}$, t_i, and i_r. Then, $\mathbb{IPSC}$ stores the keys in parameters, sets the initial version of L by putting $LRoot_{pb} \leftarrow \perp$, and sets the initial total issued tokens and the total supply, both to t_i.[5]

Client Registration. A client $\mathbb{C}$ registers with $\mathbb{O}$, who performs know your customer (KYC) checks and submits her public key $PK_{pb}^{\mathbb{C}}$ to $\mathbb{E}$. Then, $\mathbb{E}$ outputs an execution receipt about the successful registration of $\mathbb{C}$ as well as her access ticket $t^{\mathbb{C}}$ that will serve for potential communication with $\mathbb{IPSC}$ and its purpose is to avoid spamming $\mathbb{IPSC}$ by invalid requests. In detail, $t^{\mathbb{C}}$ is the $\mathbb{E}$-signed tuple that contains $PK_{pb}^{\mathbb{C}}$ and optionally other fields such as the account expiration timestamp. Next, $\mathbb{C}$ verifies whether her registration (proved by the receipt) was already snapshotted by $\mathbb{O}$ at $\mathbb{IPSC}$.

4.1.3 *Normal Operation.*

$\mathbb{C}$s send *μ-transactions* (writing to L) and *queries* (reading from L) to $\mathbb{O}$, who validates them and relays them to $\mathbb{E}$, which processes them within its virtual machine (Aquareum uses eEVM [50]). Therefore, L and its state are modified

[5] Among these parameters, a constructor of $\mathbb{IPSC}$ also accepts the indication whether an instance is allowed to issue tokens. This is, however, implicit for the single instance, while restrictions are reasonable in the case of multiple instances.

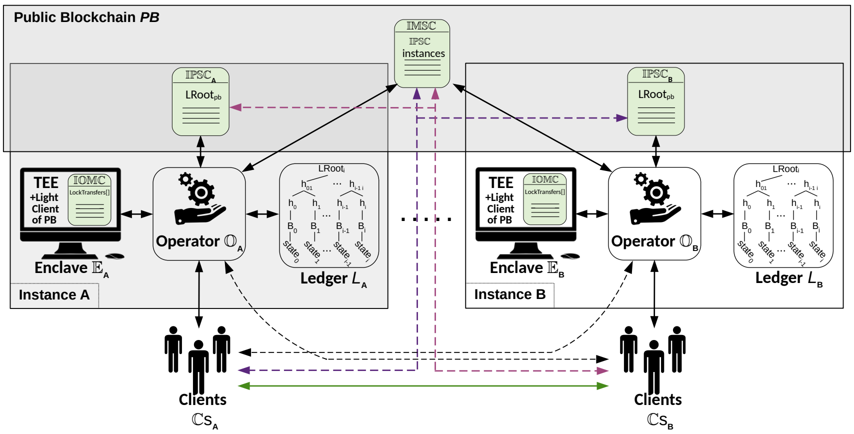


Figure 2: Overview of our CBDC architecture supporting interoperability among multiple CBDC instances (i.e., banks). The schema depicts two instances, where each of them has its own centralized ledger L modified in a secure way through TEE of $\mathbb{E}$, while its integrity is ensured by periodic integrity snapshots to the integrity preserving smart contract ($\mathbb{IPSC}$) in a public blockchain PB. Each CBDC instance is registered in the identity management smart contract $\mathbb{IMSC}$ of a public blockchain, serving as a global registry of bank instances. A client who makes an inter-bank transfer communicates with her bank and the counter-party bank utilizing interoperability micro contracts ($\mathbb{IOMC}$), running in the TEE. Any censored request of a client is resolved by $\mathbb{IPSC}$ of a particular bank and can be initiated by its client or a counter-party client.

in a trusted code of $\mathbb{E}$, creating a new version of L, which is represented by the root hash $LRoot$ of the history tree. Note that program $prog^{\mathbb{E}}$ is public and can be remotely attested by $\mathbb{C}$s (or anybody). $\mathbb{O}$ is responsible for a periodic synchronization of the most recent root hash $LRoot_{cur}$ (i.e., snapshotting the current version of L) to $\mathbb{IPSC}$, running on a public blockchain PB. Besides, $\mathbb{C}$s use this smart contract to resolve censored transactions and queries, while preserving the privacy of data.

4.1.4 ***Censorship Resolution.*** $\mathbb{O}$ might potentially censor some *write* transactions or *read* queries of $\mathbb{C}s$. However, these can be resolved by Aquareum's mechanism as follows. If $\mathbb{C}$'s μ-transaction μ-tx is censored by $\mathbb{O}$, $\mathbb{C}$ first creates $PK^{tee}_{\mathbb{E}}$-encrypted $\mu\text{-}etx$ (to ensure privacy in PB), and then she creates and signs a transaction containing $\mathbb{C}'s$ access ticket $t^{\mathbb{C}}$ and $\mu\text{-}etx$. $\mathbb{C}$ sends this transaction to $\mathbb{IPSC}$, which verifies $t^{\mathbb{C}}$ and stores $\mu\text{-}etx$, which is now visible to $\mathbb{O}$ and the public. Therefore, $\mathbb{O}$ might relay $\mu\text{-}etx$ to $\mathbb{E}$ for processing and then provide $\mathbb{E}$-signed execution receipt to $\mathbb{IPSC}$ that publicly resolves this censorship request. On the other hand, if $\mathbb{O}$ were not to do it, $\mathbb{IPSC}$ would contain an indisputable proof of censorship by $\mathbb{O}$ on a client $\mathbb{C}$.

4.2 Multiple CBDC Instances

The conceptual model of our interoperable CBDC architecture is depicted in Figure 2. It consists of multiple CBDC instances (i.e., at least two), whose $\mathbb{C}s$ communicate in three different ways: (1) directly with each other, (2) in the instance-to-instance fashion through the infrastructure of their $\mathbb{O}$ as well as counterpart's $\mathbb{O}$, (3) through PB with $\mathbb{IPSC}$ of both $\mathbb{O}$s and a global registry $\mathbb{IMSC}$ managing identities of instances.

For simplified description, in the following we assume the transfer operation where a local CBDC instance in Figure 2 is A (i.e., the sender of tokens) and the external one is B (i.e., the receiver of tokens). To ensure interoperability, we require a communication channel of local clients $\mathbb{C}s_A$ to external clients $\mathbb{C}s_B$ (the green arrow), the local operator $\mathbb{O}_A$ (the black arrow), and the external operator $\mathbb{O}_B$ (the black dashed arrow). In our interoperability protocol Π^T (described later in Section 4.3), external $\mathbb{C}s_B$ use the channel with the local operator $\mathbb{O}_A$ only for obtaining incremental proofs of L_A's history tree to verify inclusion of some μ-transactions in L_A. However, there might arise a situation in which $\mathbb{O}_A$ might censor such queries, therefore, we need to address it by another communication channel – i.e., the public blockchain PB.

Censorship of External Clients. We allow external clients $\mathbb{C}s_B$ to use the same means of censorship resolution as internal clients of a single CBDC instance (see Section 4.1.4). To request a resolution of a censored query, the external $\mathbb{C}_B$ uses the access ticket $t^{\mathbb{C}_B}$ at $\mathbb{IPSC}_A$, which is issued by $\mathbb{E}_A$ in the first phase of Π^T.

Identification of Client Accounts. To uniquely identify $\mathbb{C}$'s account at a particular CBDC instance, first it is necessary to specify the globally unique identifier of the CBDC instance. The best candidate is the blockchain address of the $\mathbb{IPSC}$ in PB since it is publicly visible and unique in PB (and we denote it by $\mathbb{IPSC}$). Then, the identification of $\mathbb{C}$'s relevant account is a pair $\mathbb{C}^{ID} = \{PK^{\mathbb{C}}_{pb} || \ \mathbb{IPSC}\}$. Note that $\mathbb{C}$ might use the same $PK^{\mathbb{C}}_{pb}$ for the registration at multiple CBDC instances (i.e., equivalent of having accounts in multiple banks); however, to preserve better privacy, making linkage of $\mathbb{C}$'s instances more difficult, we recommend $\mathbb{C}$s to have dedicated key pair for each instance.

4.2.1 ***Identity Management of CBDC Instances.*** To manage identities of all CBDC instances in the system, we need a global registry of their identifiers – $\mathbb{IPSC}$ addresses. For this purpose, we use the Identity Management Smart Contract ($\mathbb{IMSC}$) deployed in PB (see program $prog^{\mathbb{IMSC}}$ in Algorithm 1). We propose $\mathbb{IMSC}$ to be managed in either decentralized or centralized fashion, depending on the deployment scenario described below.

Deployment scenarios

Decentralized Scheme. In the decentralized scheme, the enrollment of a new CBDC instance must be approved by a majority vote of the already existing instances. This might be convenient for interconnecting central banks from various countries/regions.

The enrollment requires creating a request entry at $\mathbb{IMSC}$ (i.e., $newJoinRequest()$) by a new instance specifying the address of its $\mathbb{IPSC}_{new}$ and $PK_{PB}^{\mathbb{O}_{new}}$. Then, the request has to be approved by voting of existing instances. Prior to voting (i.e., $approveJoinRequest()$), the existing instances should first verify a new instance by certain legal processes as well as by technical means: do the remote attestation of $prog_{new}^{\mathbb{E}}$, verify the inflation rate i_r and the initial value of total issued tokens t_i in $\mathbb{IPSC}$, etc. Removing of the existing instance also requires the majority of all instances, who should verify legal conditions prior to voting.

Centralized Scheme. So far, we were assuming that CBDC instances are equal, which might be convenient for interconnection of central banks from different countries. However, from the single-country point-of-view, there usually exist only one central bank, which might not be interested in decentralization of its competences (e.g., issuing tokens, setting inflation rate) among multiple commercial banks. We respect this and enable our approach to be utilized for such a use case, while the necessary changes are made to $\mathbb{IMSC}_c$ (see Algorithm 2), allowing to have only one CBDC authority that can add or delete instances of (commercial) banks, upon their verification (as outlined above). The new instances can be adjusted even with token issuance capability and constraints on inflation, which is enforced within the code of $\mathbb{E}$ as well as $\mathbb{IPSC}$.

4.2.2 ***Token Issuance.*** With multiple CBDC instances, $\mathbb{C}$s and the public can obtain the total value of issued tokens in the ecosystem of CBDC and compare it to the total value of token supply of all instances. Nevertheless, assuming only two instances A and B, the value of t_s snapshotted by $\mathbb{IPSC}_A$ might not reflect the recently executed transfers to instance B that might have already made the snapshot of its actual L_B version to $\mathbb{IPSC}_B$, accounting for the transfers. As a consequence, given a set of instances, the value of the aggregated t_s should always be greater or equal than the corresponding sum of t_i:

$$t_i^A + t_i^B \quad \leq \quad t_s^A + t_s^B. \tag{1}$$

We can generalize it for N instances known by $\mathbb{IMSC}$ as follows:

$$\sum_{\forall X \ \in \ \mathbb{IMSC}} t_i^X \quad \leq \quad \sum_{\forall X \ \in \ \mathbb{IMSC}} t_s^X. \tag{2}$$

4.2.3 ***Inflation Rate.*** In contrast to a single CBDC instance, multiple independent instances must provide certain guarantees about inflation not only to their clients, but also to each other. For this purpose, the parameter inflation rate i_r is adjusted to a constant value in the initialization of $\mathbb{IPSC}$ and checked before the instance is approved at $\mathbb{IMSC}$.

If one would like to enable the update of i_r at CBDC instances, a majority vote at $\mathbb{IMSC}$ on a new value could be utilized (or just the vote of authority in the case of centralized scenario). Nevertheless, to support even fairer properties, $\mathbb{C}$s of a particular instance might vote on the value of i_r upon its acceptance by $\mathbb{IOMC}$ and before it is propagated to $\mathbb{IPSC}$ of an instance. Then, based on the new value of $\mathbb{IPSC}.i_r$, $\mathbb{E}.i_r$ can be adjusted as well (i.e., upon the validation by the light client of $\mathbb{E}$). However, the application of such a mechanism might depend on the use case, and we state it only as a possible option that can be enabled in our approach.

Algorithm 1: $prog_d^{\mathbb{IMSC}}$ of decentralized $\mathbb{IMSC}$

```
▷ DECLARATION OF TYPES AND VARIABLES:
  $msg$: a current transaction that called $\mathbb{IMSC}$,
  struct InstanceInfo {
      $operator : PK_{\mathbb{O}}^{PB}$ of the instance's $\mathbb{O}$,
      $isApproved$: admission status of the instance,
      $approvals \leftarrow [\,]$ : $\mathbb{O}$s who have approved the instance creation (or deletion),
  }
  $instances[\,]$: a mapping of $\mathbb{IPSC}$ to InstanceInfo,
▷ DECLARATION OF FUNCTIONS:
function $Init(\mathbb{IPSC}s[\,], \mathbb{O}s[\,])$ public          ▷ Initial instances are implicitly approved.
    assert $|\mathbb{IPSC}s| = |\mathbb{O}s|$ ;
    for $i \leftarrow 0;\ i \leq |\mathbb{O}s|;\ i \leftarrow i + 1$ do
        $instances[\mathbb{IPSC}s[i]] \leftarrow$ InstanceInfo$(\mathbb{O}s[i], True, [\,])$;

function $newJoinRequest(\mathbb{IPSC})$ public
    assert $instances[\mathbb{IPSC}] = \perp$;                      ▷ The instance must not exist yet.
    $instances[\mathbb{IPSC}] \leftarrow$ InstanceInfo$(msg.sender, False, [\,])$;

function $approveJoinRequest(\mathbb{IPSC}_{my}, \mathbb{IPSC}_{new})$ public
    assert $instances[\mathbb{IPSC}_{my}].operator = msg.sender$;          ▷ Sender's check.
    assert $instances[\mathbb{IPSC}_{my}].isApproved$;     ▷ The sending $\mathbb{O}$ has valid instance.
    assert $!instances[\mathbb{IPSC}_{new}].isApproved$;   ▷ The new instance is not approved.
    $r \leftarrow instances[\mathbb{IPSC}_{new}]$;
    $r.approvals[msg.sender] \leftarrow True$;       ▷ The sender acknowledges the request.
    if $|r.approvals| > \lfloor |instances|/2 \rfloor$ then
        $r.isApproved \leftarrow True$;                        ▷ Majority vote applies.
        $r.approvals \leftarrow [\,]$;               ▷ Switch this field for a potential deletion.

function $approveDelete(\mathbb{IPSC}_{my}, \mathbb{IPSC}_{del})$ public
    assert $instances[\mathbb{IPSC}_{my}].operator = msg.sender$;          ▷ Sender's check.
    assert $instances[\mathbb{IPSC}_{my}].isApproved$;     ▷ The sending $\mathbb{O}$ has valid instance.
    assert $instances[\mathbb{IPSC}_{del}].isApproved$;         ▷ An instance to delete must be
     approved.
    $r \leftarrow instances[\mathbb{IPSC}_{del}]$;
    $r.approvals[msg.sender] \leftarrow True$;       ▷ The sender acknowledges the request.
    if $|r.approvals| > \lfloor |instances|/2 \rfloor$ then
        delete $r$;
```

Algorithm 2: $prog_c^{\mathbb{IMSC}}$ of centralized $\mathbb{IMSC}$

```
▷ DECLARATION OF TYPES AND VARIABLES:
  $msg$: a current transaction that called $\mathbb{IMSC}$,
  $authority$: $\mathbb{IPSC}$ of the authority bank,
  $authority^{\mathbb{O}}$: $PK_{pb}^{\mathbb{O}}$ of $\mathbb{O}$ at authority bank,
  $instances[\,]$: a mapping of $\mathbb{IPSC}$ to $PK_{\mathbb{O}}^{PB}$,
▷ DECLARATION OF FUNCTIONS:
function $Init(\mathbb{IPSC})$ public                 ▷ Initial instances are implicitly approved.
    $authority^{\mathbb{O}} \leftarrow msg.sender$;
    $authority \leftarrow \mathbb{IPSC}$;

function $addInstance(\mathbb{IPSC}_{new},\ \mathbb{O}_{new})$ public
    assert $msg.sender = authority^{\mathbb{O}}$ ;        ▷ Only the authority can add instances.
    assert $instances[\mathbb{IPSC}_{new}] = \perp$;             ▷ The instance must not exist yet.
    $instances[\mathbb{IPSC}_{new}] \leftarrow \mathbb{O}_{new}$;

function $delInstance(\mathbb{IPSC}_{del})$ public
    assert $msg.sender = authority^{\mathbb{O}}$ ;     ▷ Only the authority can delete instances.
    delete $instances[\mathbb{IPSC}_{del}]$;
```

4.2.4 ***Interoperability.*** The interoperability logic itself is provided by our protocol Π^T that utilizes $\mathbb{I}$nter$\mathbb{O}$perability $\mathbb{M}$icro $\mathbb{C}$ontracts $\mathbb{IOMC}^S$ and $\mathbb{IOMC}^R$, which serve for sending and receiving tokens, respectively. Therefore, in the context of $\mathbb{E}$-isolated environment these μ-contracts allow to mint and burn tokens, reflecting

Algorithm 3: $prog^{\mathbb{IOMC}^S}$ **of sending** $\mathbb{IOMC}^S$

```
▷ Declaration of types and variables:
  $\mathbb{E}$,                                     ▷ The reference to $\mathbb{E}_A$ of sending party.
  $msg$,                                   ▷ The current $\mu$-transaction that called $\mathbb{IOMC}^S$.
  struct LockedTransfer {
      $sender$,                            ▷ Sending client $\mathbb{C}_A$.
      $receiver$,                          ▷ Receiving client $\mathbb{C}_B$.
      $receiver\mathbb{IPSC}$,             ▷ The $\mathbb{IPSC}$ contract address of the receiver's instance.
      $amount$,                            ▷ Amount of tokens sent.
      $hashlock$,                          ▷ Hash of the secret of the sending $\mathbb{C}_A$.
      $timelock$,                          ▷ A timestamp defining the end of validity of the transfer.
      $isCompleted$,                       ▷ Indicates whether the transfer has been completed.
      $isReverted$,                        ▷ Indicates whether the transfer has been canceled.
  },
  $transfers \leftarrow []$,               ▷ Initiated outgoing transfers (i.e., LockedTransfer).
  const $timeout^{HTLC} \leftarrow 24h$,   ▷ Set the time lock for e.g., 24 hours.
▷ Declaration of functions:
function sendInit(receiver, receiverIPSC, hashlock) public payable
    assert $msg.value > 0$;                                   ▷ Checks the amount of tokens.
    $timelock \leftarrow timestamp.now() + timeout^{HTLC}$;
    $t \leftarrow$ LockedTransfer($msg.sender, receiver, receiver\mathbb{IPSC}$,
        $msg.value, hashlock, timelock, False, False$);      ▷ A new receiving transfer.
    $transfers.append(t)$;
    Output ("sendInitialized", $transferID \leftarrow |transfers| - 1$));

function sendCommit(transferID, secret, extTransferID) public
    assert $transfers[transferID] \neq \perp$;               ▷ Check the existence of locked transfer.
    $t \leftarrow transfers[transferID]$;
    assert $t.hashlock = h(secret)$;                          ▷ Check the secret.
    assert $!t.isCompleted \wedge !t.isReverted$;             ▷ Test if the transfer is still pending.
    $t.isCompleted \leftarrow True$;
    burn t.amount;                                            ▷ Burn tokens.
    $\mathbb{E}.t_s \leftarrow \mathbb{E}.t_s - t.amount$;    ▷ Decrease the total supply of the instance.
    Output ("sendCommitted", $transferID, extTransferID, t.receiver$,
      $t.receiver\mathbb{IPSC}, t.amount$);

function sendRevert(transferID) public
    assert $transfers[transferID] \neq \perp$;               ▷ Check the existence of locked transfer.
    $t \leftarrow transfers[transferID]$;
    assert $!t.isCompleted \wedge !t.isReverted$;             ▷ Test the transfer is still pending.
    assert $t.timelock \leq timestamp.now()$;                 ▷ Check the HTLC expiration.
    $transfer(t.amount, t.sender)$;                           ▷ Returning tokens back to the sender.
    $t.isReverted \leftarrow True$;
    Output("sendReverted", $transferID$);
```

the changes in t_s after sending or receiving tokens between CBDC instances. Both μ-contracts are deployed in $\mathbb{E}$ by each $\mathbb{O}$ as soon as the instance is created, while $\mathbb{E}$ records their addresses that can be obtained and attested by $\mathbb{C}$s. We briefly review these contracts in the following, and we will demonstrate their usage in Section 4.3.

The Sending $\mathbb{IOMC}^S$. The sending $\mathbb{IOMC}^S$ (see Algorithm 3) is based on Hash Time LoCks (HTLC), thus upon initialization of transfer by *hashlock* provided by $\mathbb{C}_A$ (i.e., $hashlock \leftarrow h(secret)$) and calling *sendInit*$(hashlock, \ldots)$, $\mathbb{IOMC}^S$ locks transferred tokens for the timeout required to complete the transfer by *sendCommit*$(secret, \ldots)$. If tokens are not successfully transferred to the recipient of the external instance during the timeout, they can be recovered by the sender (i.e., *sendRevert*$()$).[6] If tokens were sent successfully from $\mathbb{C}_A$ to $\mathbb{C}_B$, then instance A burns them within *sendCommit*$()$ of $\mathbb{IOMC}^S$ and deducts them from t_s. Note that deducting t_s is a special operation that cannot be executed within standard μ-contracts, but $\mathbb{IOMC}$ contracts are exceptions and can access some variables of $\mathbb{E}$.

The Receiving $\mathbb{IOMC}^R$. The receiving $\mathbb{IOMC}^R$ (see Algorithm 4) is based on Hashlocks (referred to as HLC) and works pairwise with sending $\mathbb{IOMC}^S$ to facilitate four phases of our interoperable transfer protocol Π_T (described below). After calling $\mathbb{IOMC}^{\mathbb{S}}.sendInit()$, incoming initiated transfer is recorded at $\mathbb{IOMC}^R$ by *receiveInit*$(hashlock, \ldots)$. Similarly, after executing token deduction at instance A (i.e., $\mathbb{IOMC}^{\mathbb{S}}.sendCommit(secret, \ldots)$), incoming transfer is executed at $\mathbb{IOMC}^R$ by *receiveCommit*$(secret, \ldots)$ that mints tokens to $\mathbb{C}_B$ and increases t_s. Similar to $\mathbb{IOMC}^S$, minting tokens and increasing t_s are special operations requiring access to $\mathbb{E}$, which is exceptional for $\mathbb{IOMC}$. The overview of Π_T is depicted in Figure 3.

Algorithm 4: $prog^{\mathbb{IOMC}^R}$ **of receiving** $\mathbb{IOMC}^R$

```
▷ Declaration of types and variables:
  $\mathbb{E}$,                                     ▷ The reference to $\mathbb{E}_B$ of receiving party.
  struct LockedTransfer {
      $sender$,                            ▷ Sending client $\mathbb{C}_A$.
      $sender\mathbb{IPSC}$,               ▷ The IPSC contract address of the sender's instance.
      $receiver$,                          ▷ Receiving client $\mathbb{C}_B$.
      $amount$,                            ▷ Amount of transferred tokens.
      $hashlock$,                          ▷ Hash of the secret of the sending $\mathbb{C}_A$.
      $isCompleted$,                       ▷ Indicates whether the transfer has been completed.
  },
  $transfers \leftarrow []$,               ▷ Initiated incoming transfers (i.e., LockedTransfer).
▷ Declaration of functions:
function receiveInit(sender, senderIPSC, hashlock, amount) public
    assert $amount > 0$;
    $t \leftarrow$ LockedTransfer($sender, sender\mathbb{IPSC}, msg.sender, amount$,
                  $hashlock, False$);                          ▷ Make a new receiving transfer entry.
    $transfers.append(t)$;
    Output("receiveInitialized", $transferID \leftarrow |transfers| - 1$);

function receiveCommit(transferID, secret) public
    assert $transfers[transferID] \neq \perp$;               ▷ Check the existence of transfer entry.
    $t \leftarrow transfers[transferID]$;
    assert $t.hashlock = h(secret)$;                          ▷ Check the secret.
    assert $!t.isCompleted$;                                  ▷ Check whether the transfer is pending.
    $\mathbb{E}.mint(this, t.amount)$;                        ▷ Call $\mathbb{E}$ to mint tokens on $\mathbb{IOMC}_R$.
    $\mathbb{E}.t_s \leftarrow \mathbb{E}.t_s + t.amount$ ;   ▷ Increase the total supply of the instance.
    $transfer(t.amount, t.receiver)$;                         ▷ Credit tokens to the recipient.
    $t.isCompleted \leftarrow True$;
    Output("receiveCommited", $transferID$);
```

4.3 Interoperable Transfer Protocol Π^T

In this section we outline our instance-to-instance interoperable transfer protocol Π^T for inter-CBDC transfer operation, which is inspired by the atomic swap protocol (see Section 2.4), but in contrast to the exchange-oriented approach of atomic swap, Π^T focuses only on one-way atomic transfer between instances of the custodial environment of CBDC, where four parties are involved in each transfer – a sending $\mathbb{C}_A$ and $\mathbb{O}_A$ versus a receiving $\mathbb{C}_B$ and $\mathbb{O}_B$. The goal of Π^T is to eliminate any dishonest behavior by $\mathbb{C}s$ or $\mathbb{O}$s that would incur token duplication or the loss of tokens.

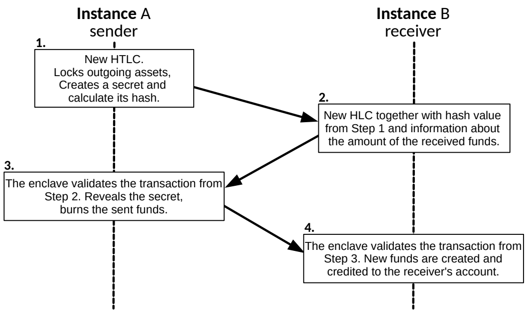

Instance A
sender
Instance B
receiver
1.
New HTLC.
Locks outgoing assets,
Creates a secret and
calculate its hash.
2.
New HLC together with hash value
from Step 1 and information about
the amount of the received funds.
3.
The enclave validates the transaction from
Step 2. Reveals the secret,
burns the sent funds.
4.
The enclave validates the transaction from
Step 3. New funds are created and
credited to the receiver's account.

Figure 3: Overview of the protocol Π^T*, consisting of 4 phases.*

[6] Note that setting a short timeout might prevent the completion of the protocol.

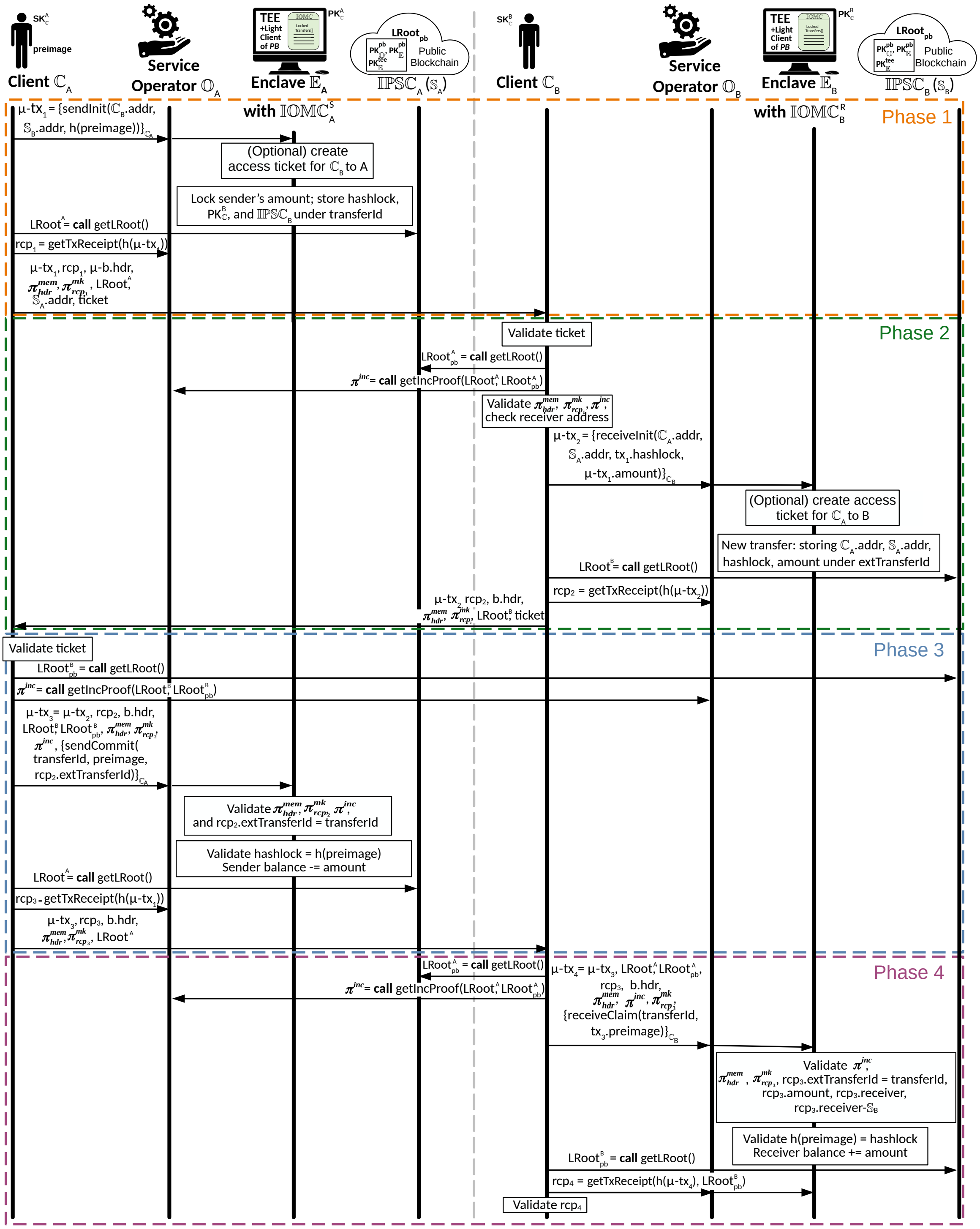

SK$^A_\mathbb{C}$
preimage
Client $\mathbb{C}_A$
Service Operator $\mathbb{O}_A$
TEE +Light Client of PB
IOMC
Locked Transfers[]
PK$^A_\mathbb{C}$
Enclave $\mathbb{E}_A$
LRoot$_{pb}$
PK$^{pb}_\mathbb{O}$, PK$^{pb}_\mathbb{E}$
PK$^{tee}_\mathbb{E}$
Public Blockchain
$\mathbb{IPSC}_A$ ($\mathbb{S}_A$)
SK$^B_\mathbb{C}$
Client $\mathbb{C}_B$
Service Operator $\mathbb{O}_B$
PK$^B_\mathbb{C}$
Enclave $\mathbb{E}_B$
$\mathbb{IPSC}_B$ ($\mathbb{S}_B$)
with $\mathbb{IOMC}^S_A$
with $\mathbb{IOMC}^R_B$
Phase 1
µ-tx$_1$ = {sendInit($\mathbb{C}_B$.addr, $\mathbb{S}_B$.addr, h(preimage))}$_{\mathbb{C}_A}$
(Optional) create access ticket for $\mathbb{C}_B$ to A
Lock sender's amount; store hashlock, PK$^B_\mathbb{C}$, and $\mathbb{IPSC}_B$ under transferId
LRoot^A= call getLRoot()
rcp$_1$ = getTxReceipt(h(µ-tx$_1$))
µ-tx$_1$, rcp$_1$, µ-b.hdr, π^{mem}_{hdr}, $\pi^{mk}_{rcp_1}$, LRoot^A, $\mathbb{S}_A$.addr, ticket
Phase 2
Validate ticket
LRoot$^A_{pb}$ = call getLRoot()
π^{inc}= call getIncProof(LRoot^A, LRoot$^A_{pb}$)
Validate π^{mem}_{hdr}, $\pi^{mk}_{rcp_1}$, π^{inc}, check receiver address
µ-tx$_2$ = {receiveInit($\mathbb{C}_A$.addr, $\mathbb{S}_A$.addr, tx$_1$.hashlock, µ-tx$_1$.amount)}$_{\mathbb{C}_B}$
(Optional) create access ticket for $\mathbb{C}_A$ to B
New transfer: storing $\mathbb{C}_A$.addr, $\mathbb{S}_A$.addr, hashlock, amount under extTransferId
LRoot^B= call getLRoot()
rcp₂ = getTxReceipt(h(µ-tx$_2$))
µ-tx$_2$, rcp₂, b.hdr, π^{mem}_{hdr}, $\pi^{mk}_{rcp_2}$, LRoot^B, ticket
Phase 3
Validate ticket
LRoot$^B_{pb}$ = call getLRoot()
π^{inc}= call getIncProof(LRoot^B, LRoot$^B_{pb}$)
µ-tx$_3$= µ-tx$_2$, rcp₂, b.hdr, LRoot^B, LRoot$^B_{pb}$, π^{mem}_{hdr}, $\pi^{mk}_{rcp_2}$, π^{inc}, {sendCommit(transferId, preimage, rcp₂.extTransferId)}$_{\mathbb{C}_A}$
Validate π^{mem}_{hdr}, $\pi^{mk}_{rcp_2}$, π^{inc}, and rcp₂.extTransferId = transferId
Validate hashlock = h(preimage) Sender balance -= amount
LRoot^A= call getLRoot()
rcp₃ = getTxReceipt(h(µ-tx$_3$))
µ-tx$_3$, rcp₃, b.hdr, π^{mem}_{hdr}, $\pi^{mk}_{rcp_3}$, LRoot^A
Phase 4
LRoot$^A_{pb}$ = call getLRoot()
π^{inc}= call getIncProof(LRoot^A, LRoot$^A_{pb}$)
µ-tx$_4$= µ-tx$_3$, LRoot^A, LRoot$^A_{pb}$, rcp₃, b.hdr, π^{mem}_{hdr}, π^{inc}, $\pi^{mk}_{rcp_3}$, {receiveClaim(transferId, tx$_3$.preimage)}$_{\mathbb{C}_B}$
Validate π^{inc}, π^{mem}_{hdr}, $\pi^{mk}_{rcp_3}$, rcp₃.extTransferId = transferId, rcp₃.amount, rcp₃.receiver, rcp₃.receiver-$\mathbb{S}_B$
Validate h(preimage) = hashlock Receiver balance += amount
LRoot$^B_{pb}$ = call getLRoot()
rcp₄ = getTxReceipt(h(µ-tx$_4$), LRoot$^B_{pb}$)
Validate rcp₄

Figure 4: The details of the proposed interoperability protocol Π^T.

To execute Π^T it is necessary to inter-connect $\mathbb{E}$s of two instances involved in a transfer. However, $\mathbb{E}$ does not allow direct communication with the outside world, and therefore it is necessary to use an intermediary. One solution is to involve $\mathbb{O}$s but they might be overwhelmed with other activities, updating the ledger by executing μ-transactions, and moreover, they might not have direct incentives to execute inter-CBDC transfers. Therefore, we argue that in contrast to the above option, involving $\mathbb{C}s$ as intermediaries has two advantages: (1) elimination of the synchronous communication overhead on $\mathbb{O}$s and (2) enabling $\mathbb{C}$s to have a transparent view about the status of the transfer and take action if required. In the following, we describe phases of Π^T in detail (see also Figure 4).

Phase 1 – Client $\mathbb{C}_A$ Initiates the Protocol

The client $\mathbb{C}_A$ creates a μ-tx$_1$ with the amount being sent, which invokes the `sendInit()` of $\mathbb{IOMC}_A$ with arguments containing the address of the external client $\mathbb{C}_B$, the address of $\mathbb{IPSC}_B$ (denoted as $\mathbb{S}_B$ in Figure 4 for brevity), and the hash of the secret that is created by $\mathbb{C}_A$. $\mathbb{C}_A$ sends signed μ-tx$_1$ to $\mathbb{O}_A$ who forwards it to the $\mathbb{E}_A$. Before executing the μ-tx$_1$, $\mathbb{E}_A$ ensures that the external recipient (i.e., $\mathbb{C}_B$) has the access ticket already issued and valid, enabling her to post censorship resolution requests to $\mathbb{IPSC}_A$ (if needed). The access ticket should be valid for at least the entire period defined by the HTLC of $\mathbb{IOMC}_A$. In the next step, a μ-tx$_1$ is executed by $\mathbb{E}_A$, creating a new transfer record with $transferId$ in $\mathbb{IOMC}_A$. During the execution, $\mathbb{C}_A$'s tokens are transferred (and thus locked) to the $\mathbb{IOMC}_A$'s address. $\mathbb{C}_A$ waits until the new version of L_A is snapshotted to $\mathbb{IPSC}_A$, and then obtains $LRoot^A$ from it. Then $\mathbb{C}_A$ asks $\mathbb{O}_A$ for the execution receipt rcp_1 of μ-tx$_1$ that also contains a set of proofs (π^{mem}_{hdr}, $\pi^{mk}_{rcp_1}$) and the header of the μ-block that includes μ-tx$_1$. In detail, π^{mem}_{hdr} is the inclusion proof of the μ-block b in the current version of L_A; $\pi^{mk}_{rcp_1}$ is the Merkle proof proving that rcp_1 is included in b (while rcp_1 proves that μ-tx$_1$ was executed correctly). The mentioned proofs and the receipt are provided to $\mathbb{C}_B$, who verifies that μ-tx$_1$ was executed and included in the L_A's version that is already snapshotted to $\mathbb{IPSC}_A$, thus irreversible (see below).

Phase 2 – $\mathbb{C}_B$ Initiates Receive

First, $\mathbb{C}_B$ validates an access ticket to $\mathbb{IPSC}_A$ using the enclave $\mathbb{E}_A$'s public key accessible in that smart contract. Next, $\mathbb{C}_B$ obtains the root hash $LRoot^A_{pb}$ of L_A to ensure that $\mathbb{C}_B$'s received state has been already published in $\mathbb{IPSC}_A$, and thus contains μ-tx$_1$. After obtaining $LRoot^A_{pb}$, $\mathbb{C}_B$ forwards it along with the root $LRoot^A$ obtained from $\mathbb{C}_A$ to $\mathbb{O}_A$, who creates an incremental proof π^{inc} of $\langle LRoot^A, LRoot^A_{pb}\rangle$. Once the proof π^{inc} has been obtained and validated, the protocol can proceed to validate the remaining proofs sent by the client $\mathbb{C}_A$ along with verifying that the receiving address belongs to $\mathbb{C}_B$. Next, $\mathbb{C}_B$ creates μ-tx_2, invoking the method `receiveInit()` with the arguments: the address of $\mathbb{C}_A$ obtained from μ-tx_1,[7] the address $\mathbb{IPSC}_A.addr$ of $\mathbb{C}_A$'s instance, the hash value of the secret, and the amount of crypto-tokens being sent. $\mathbb{C}_B$ sends μ-tx_2 to $\mathbb{O}_B$, who forwards it to $\mathbb{E}_B$ for processing. During processing of μ-tx_2, $\mathbb{E}_B$ determines whether the external client (from its point of view – i.e., $\mathbb{C}_A$) has an access ticket issued with a sufficiently long validity period; if not, one is created. Subsequently, $\mathbb{E}_B$ creates a new record in $\mathbb{IOMC}_B$ with $extTransferId$. Afterward, $\mathbb{C}_B$ retrieves the $LRoot^B$ from L_B and requests the execution receipt rcp_2 from $\mathbb{O}_B$, acknowledging that the μ-tx_2 has been executed. Finally, $\mathbb{C}_B$ sends a message $\mathbb{C}_A$ with μ-tx_2 and cryptographic proofs π^{mem}_{hdr}, $\pi^{mk}_{rcp_2}$, the execution receipt of μ-tx_2, the block header b in which the μ-tx_2 was included, $LRoot^B$ (i.e., the root value of L_B after μ-tx_2 was executed), and the valid client access ticket for $\mathbb{C}_A$.

Phase 3 – Confirmation of Transfer by $\mathbb{C}_A$

First, $\mathbb{C}_A$ validates the received access ticket to $\mathbb{IPSC}_B$. Next, $\mathbb{C}_A$ obtains the snapshotted root hash $LRoot^B_{pb}$ of L_B from $\mathbb{IPSC}_B$. As in the previous phases, it is necessary to verify that the version of L_B that includes μ-tx_2 is represented by $LRoot^B_{pb}$ (thus is irreversible). Next, both root hashes ($LRoot^B$ and $LRoot^B_{pb}$) are sent to the external operator $\mathbb{O}_B$, which produces the incremental proof π^{inc} from them. Next, $\mathbb{C}_A$ creates μ-tx_3 that consists of invoking the `sendCommit()` method at $\mathbb{E}_A$ with the arguments containing the published secret (i.e., $preimage$) and the record identifier of the transfer at local instance (i.e., $transferId$) as well as the external one (i.e., $extTransferId$). Along with the invocation of `sendCommit()`, μ-tx_3 also wraps π^{inc} with its versions ($LRoot^B_{pb}$ and $LRoot^B$), μ-tx_2, its execution receipt rcp_2 with its Merkle proof $\pi^{mk}_{rcp_2}$, $b.hdr$ – the header of the block that included μ-tx_2, and its membership proof π^{mem}_{hdr} of L_B. Next, $\mathbb{C}_A$ sends μ-tx_3 to $\mathbb{E}_A$ through $\mathbb{O}_A$. During the execution of μ-tx_3, $\mathbb{E}_A$ validates the provided proofs and the equality of transfer IDs from both sides of the protocol. Note that to verify π^{mem}_{hdr}, $\mathbb{E}_A$ uses its light client to L_B. $\mathbb{E}_A$ then validates whether $\mathbb{C}_A$'s provided secret corresponds to the hashlock recorded in the 1st phase of the protocol, and if so, it burns the sent balance of the transfer.

Next, $\mathbb{C}_A$ waits until the new version of L_A is snapshotted to $\mathbb{IPSC}_A$, and then obtains $LRoot^A$ from it. Then $\mathbb{C}_A$ asks $\mathbb{O}_A$ for the execution receipt rcp_3 of μ-tx$_3$ that also contains a set of proofs (π^{mem}_{hdr}, $\pi^{mk}_{rcp_3}$) and the header of the μ-block that includes μ-tx$_3$. The proofs have the same interpretation as in the end of the 1st phase. The mentioned proofs and the receipt are provided to $\mathbb{C}_B$, who verifies that μ-tx$_1$ was executed and included in the L_A's version that is already snapshotted to $\mathbb{IPSC}_A$, thus irreversible.

Phase 4 – Acceptance of Tokens by $\mathbb{C}_B$

After receiving a message from client $\mathbb{C}_A$, the client $\mathbb{C}_B$ obtains $LRoot^A_{pb}$ from $\mathbb{IPSC}_A$ and then requests the incremental proof between versions $\langle LRoot^A, LRoot^A_{pb}\rangle$ from $\mathbb{O}_A$. Then, $\mathbb{C}_B$ creates μ-tx_4 invoking the `receiveClaim()` function at $\mathbb{E}_B$ with $transferId$ and the disclosed secret by $\mathbb{C}_A$ as the arguments. Moreover, μ-tx_4 contains remaining items received from $\mathbb{C}_A$. Then, μ-tx_4 is sent to $\mathbb{O}_B$, who forwards it to $\mathbb{E}_B$. During the execution of μ-tx_4, $\mathbb{E}_B$ verifies the provided proofs, the equality of transfer IDs from both sides of the protocol, the amount being sent, and the receiver of the transfer (i.e., $\mathbb{C}_B \,||\, \mathbb{IPSC}_B$). Note that to verify π^{mem}_{hdr}, $\mathbb{E}_B$ uses its light client to L_A. $\mathbb{E}_A$ then validates whether $\mathbb{C}_A$'s provided secret corresponds to the hashlock recorded in the 2nd phase of the protocol, and if so,

[7] Note that we assume that the address is extractable from the signature.

it mints the sent balance of the transfer on the receiver's account $\mathbb{C}_B$. Finally, $\mathbb{C}_B$ verifies that μ-tx_4 was snapshotted at $\mathbb{IPSC}_B$, thus is irreversible. In detail, first $\mathbb{C}_B$ obtains $LRoot^B_{pb}$ from $\mathbb{IPSC}_B$ and then asks $\mathbb{O}_B$ to provide her with the execution receipt rcp_4 of μ-tx_4 in the version of L_B that is equal or newer than $LRoot^B_{pb}$. Then, $\mathbb{C}_B$ verifies rcp_4, which completes the protocol.

5 IMPLEMENTATION & EVALUATION

The work is built on a *proof-of-concept* implementation of the decentralized smart contract platform Aquareum in C++ and Intel SGX technology for enclave instantiation. The IPSC contract on the public blockchain is constructed using the Solidity programming language and is prepared for deployment on the Ethereum network. The enclave employs the OpenEnclave SDK development tool,[8], which is compatible with several TEE technologies and OS systems. Aquareum incorporates the Ethereum virtual machine – EVM, in its stripped-down, minimalist version eEVM.[9]

5.0.1 *Implementation Details.*

The C++ written client application enables the clients to execute internal and external (i.e., between two instance) transfer operations as well as invoking internal and external functions of micro contracts. The operator component is represented by the C++ written server implemented as a concurrent non-blocking application that processes messages from clients. So far, the PoC of the server enables to process three types of messages: transaction execution, client registration, query for IOMC contract addresses.

5.1 Evaluation

We used *Ganache*[10] and *Truffle*,[11] to develop $\mathbb{IOMC}$, $\mathbb{IPSC}$, and $\mathbb{IMSC}$ contracts. In addition, using the *Pexpect*[12] tool, we tested the intercommunication of the implemented components and validated the correctness of the implemented interoperability protocol. The tool enabled the parallel execution and control of numerous programs (in this case, multiple Aquareum instances and client programs) to check the correctness of the expected output.

The computational cost of executing the operations defined in $\mathbb{IOMC}$ and $\mathbb{IMSC}^X$ contracts is presented in Table 1, Table 2, and Table 3.[13] We optimized our implementation to minimize the storage requirements of smart contract platform. On the other hand, it is important to highlight that $\mathbb{IOMC}^X$ μ-contracts are executed on a private ledger corresponding to the instance of CBDC, where the cost of gas is minimal or negligible as compared to a public blockchain. Other experiments are the subject of our future work.

6 SECURITY ANALYSIS

In this section, we analyze our approach in terms of security-oriented features and requirements specified in Section 3. In particular, we focus on resilience analysis of our approach against adversarial actions that the malicious CBDC instance (i.e., its operator O) or malicious client (i.e., C) can perform to violate the security requirements.

Function	constructor	sendInitialize	sendCommit	sendRevert
Deployment	901 509	160 698	64 629	60 923
Execution	653 689	134 498	42 717	39 523

Table 1: The cost of deployment and invocation of functions in the sending $\mathbb{IOMC}^S$ μ-contract in gas units (CBDC private ledger).

Function	constructor	receiveInit	receiveClaim	fund
Deployment	716 330	139 218	61 245	23 168
Execution	509 366	112 762	39 653	1 896

Table 2: The cost of deployment and invocation of functions in the receiving $\mathbb{IOMC}^R$ μ-contract in units of gas (CBDC private ledger).

Function	constructor	newJoinRequest	approveRequest	isApproved
Deployment	830 074	48 629	69 642	0
Execution	567 838	25 949	45 554	0

Table 3: The invocation cost of functions in $\mathbb{IMSC}$ smart contract in units of gas (Ethereum public blockchain).

6.1 Single Instance of CBDC

THEOREM 1. *(Correctness of Operation Execution) O is unable to modify the full state of L in a way that does not respect the semantics of VM deployed in $\mathbb{E}$ of CBDC instance.*

JUSTIFICATION. The update of the L's state is performed exclusively in $\mathbb{E}$. Since $\mathbb{E}$ contains trusted code that is publicly known and remotely attested by $\mathbb{C}$s, O cannot tamper with this code. □

THEOREM 2. *(Integrity) O is unable to modify the past records of L, and no conflicting transactions can be stored in L.*

JUSTIFICATION. All extensions of L are performed within trusted code of $\mathbb{E}$ (see Theorem 1), while utilizing the history tree [22] as a tamper evident data structure, which enables us to make only such incremental extensions of L that are consistent with L's past. □

THEOREM 3. *(Verifiability) O is unable to unnoticeably modify or delete a transaction tx that was previously inserted to L, if sync with $\mathbb{IPSC}$ was executed anytime afterward.*

JUSTIFICATION. Since tx was correctly executed (Theorem 1) as a part of the block b_i in a trusted code of $\mathbb{E}$, $\mathbb{E}$ produced a signed version transition pair $\{h(L_{i-1}), h(L_i), t_i, t_s\}_{\mathbb{E}}$ of L from the version $i-1$ to the new version i that corresponds to L with b_i included. O could either sync L with $\mathbb{IPSC}$ immediately after b_i was appended or she could do it n versions later. In the first case, O published

[8] https://openenclave.io/sdk/

[9] Microsoft's Enclave EVM is available at https://github.com/microsoft/eEVM.

[10] https://github.com/trufflesuite/ganache-cli

[11] https://github.com/trufflesuite/truffle

[12] https://github.com/pexpect/pexpect

[13] Note that we do not provide the gas measurements for $\mathbb{IPSC}$ since these are almost the same as in Aquareum [33].

$\{h(L_{i-1}), h(L_i), t_i, t_s\}_{\mathbb{E}}$ to $\mathbb{IPSC}$, which updated its current version of L to i by storing $h(L_i)$ into $\mathbb{IPSC}.LRoot_{pb}$. In the second case, n blocks were appended to L, obtaining its $(i+n)$th version. $\mathbb{E}$ executed all transactions from versions $(i+1), \ldots, (i+n)$ of L, while preserving correctness (Theorem 1) and integrity (Theorem 2). Then $\mathbb{E}$ generated a version transition pair $\{h(L_{i-1}), h(L_{i+n}), t_i, t_s\}_{\mathbb{E}}$ and O posted it to $\mathbb{IPSC}$, where the current version of L was updated to $i+n$ by storing $h(L_{i+n})$ into $\mathbb{IPSC}.LRoot_{pb}$. When any $\mathbb{C}$ requests tx and its proofs from O with regard to publicly visible $\mathbb{IPSC}.LRoot_{pb}$, she might obtain a modified tx' with a valid membership proof $\pi^{mem}_{hdr_i}$ of the block b_i but an invalid Merkle proof $\pi^{mk}_{tx'}$, which cannot be forged. □ In the case of tx deletion, O provides $\mathbb{C}$ with the tampered full block b'_i (maliciously excluding tx) whose membership proof $\pi^{mem}_{hdr'_i}$ is invalid – it cannot be forged. □

Theorem 4. *(Non-Equivocation) Assuming L synced with $\mathbb{IPSC}$: O is unable to provide two distinct $\mathbb{C}$s with two distinct valid views on L.*

Justification. Since L is periodically synced with publicly accessible $\mathbb{IPSC}$, and $\mathbb{IPSC}$ stores only a single current version of L (i.e., $\mathbb{IPSC}.LRoot_{pb}$), all $\mathbb{C}s$ share the same view on L. □

Theorem 5. *(Censorship Evidence) O is unable to censor any request (transaction or query) from $\mathbb{C}$ while staying unnoticeable.*

Justification. If $\mathbb{C}$'s request is censored by CBDC's operator O, $\mathbb{C}$ can ask for a resolution of the request through public $\mathbb{IPSC}$. O observing the request might either ignore it and leave the indisputable proof of censorship at $\mathbb{IPSC}$ or she might submit the request to $\mathbb{E}$ and obtain an enclave-signed proof witnessing that a request was processed (hence have not remained censored) – this proof is submitted to $\mathbb{IPSC}$, whereby publicly resolving the request. □

Theorem 6. *(Privacy) C is unable to obtain plain text of μ-transactions of other $\mathbb{C}$s even during the censorship resolution.*

Justification. μ-transactions are sent to $\mathbb{O}$ in TLS-encrypted messages. In the case of censorship resolution, submitted μ-transactions by $\mathbb{C}$ to public $\mathbb{IPSC}$ are encrypted by $\mathbb{E}$'s public key $PK^{tee}_{\mathbb{E}}$. □

Theorem 7. *(Transparent Token Issuance) O is unable to issue or burn any tokens without leaving a publicly visible evidence.*

Justification. All issued tokens of CBDC are publicly visible at $\mathbb{IPSC}$ since each transaction posting a new version transition pair also contains $\mathbb{E}$-signed information about the current total issued tokens t_i and total supply of the instance t_s,[14] while t_i was updated within the trusted code of $\mathbb{E}$. The information about t_i is updated at $\mathbb{IPSC}$ along with the new version of L. Note that the history of changes in total issued tokens t_i can be parsed from all transactions updating version of L published by O to PB. □

6.2 Multiple Instances of CBDC

In the following, we assume two CBDC instances A and B.

Theorem 8. *(Atomic Interoperability I) Neither O_A (operating A) nor O_B (operating B) is unable to steal any tokens during the inter-bank CBDC transfer.*

Justification. Atomic interoperability is ensured in our approach by adaptation of atomic swap protocol for all inter-bank transfers, which enables us to preserve the wholesale environment of CBDC in a consistent state (respecting Equation 2). In detail, the transferred tokens from CBDC instance A to instance B are not credited to B until A does not provide the indisputable proof that tokens were deducted from a relevant A's account. This proof confirms irreversible inclusion of tx_3 (i.e., $\mathbb{E}_A.sendCommit()$ that deducts account of A's client) in A's ledger and it is verified in 4th stage of our protocol by the trusted code of $\mathbb{E}_B$.

In the case that O_A would like to present B with integrity snapshot of L_A that was not synced to $\mathbb{IPSC}_A$ yet, B will not accept it since the 4th phase of our protocol requires O_B to fetch the recent $\mathbb{IPSC}_A.LRoot_{pb}$ and verify its consistency with A-provided $LRoot$ as well as inclusion proof in PB; all executed/verified within trusted code of $\mathbb{E}_B$. □

Theorem 9. *(Atomic Interoperability II) Colluding clients C_A and C_B of two CBDC instances cannot steal any tokens form the system during the transfer operation of our protocol.*

Justification. If the first two phases of our protocol have been executed, C_A might potentially reveal the *preimage* to C_B without running the 3rd phase with the intention to credit the tokens at B while deduction at A had not been executed yet. However, this is prevented since the trusted code of $\mathbb{E}_B$ verifies that the deduction was performed at A before crediting the tokens to C_B – as described in Theorem 8. □

Theorem 10. *(Inter-CBDC Censorship Evidence) O_A is unable to unnoticeably censor any request (transaction or query) from $\mathbb{C}_B$.*

Justification. If $\mathbb{C}_B$'s request is censored by O_A, $\mathbb{C}_B$ can ask for a resolution of the request through public $\mathbb{IPSC}_A$ since $\mathbb{C}_B$ already has the access ticket to instance A. The access ticket is signed by $\mathbb{E}_A$ and thus can be verified at $\mathbb{IPSC}_A$. Hence, the censorship resolution/evidence is the same as in Theorem 5 of a single CBDC instance. □

Theorem 11. *(Inter-CBDC Censorship Recovery) A permanent inter-CBDC censorship by O_A does not cause an inconsistent state or permanently frozen funds of undergoing transfer operations at any other CBDC instance – all initiated and not finished transfer operations can be recovered from.*

Justification. If O_A were to censor $\mathbb{C}_B$ in the 2nd phase of our protocol, no changes at ledger L_B would be made. If O_A were to censor $\mathbb{C}_B$ in the 4th phase of our protocol, L_B would contain an initiated transfer entry, which has not any impact on the consistency of the ledger since it does not contain any locked tokens. □

If O_B were to censor $\mathbb{C}_A$ in the 3rd phase of our protocol, A would contain some frozen funds of the initiated transfer. However, these funds can be recovered back to $\mathbb{C}_A$ upon a recovery call of $\mathbb{E}_A$ after

[14] Note that in the case of single CBDC instance $t_i = t_s$

a recovery timeout has passed. Note that after tokens of $\mathbb{C}_A$ have been recovered and synced to $\mathbb{IPSC}_A$ in *PB*, it is not possible to finish the 4th stage of our protocol since it requires providing the proof that tokens were deducted at *A* and such a proof cannot be constructed anymore. The same holds in the situation where the sync to $\mathbb{IPSC}_A$ at *PB* has not been made yet – after recovery of tokens, $\mathbb{E}_A$ does not allow to deduct the same tokens due to its correct execution (see Theorem 1). □

Theorem 12. *(Identity Management of CBDC Instances I) A new (potentially fake) CBDC instance cannot enter the ecosystem of wholesale CBDC upon its decision.*

Justification. To extend the list of valid CBDC instances (stored in IMSC contract), the majority vote of all existing CBDC instances must be achieved through public voting on IMSC. □

Theorem 13. *(Identity Management of CBDC Instances II) Any CBDC instance (that e.g., does not respect certain rules for issuance of tokens) might be removed from the ecosystem of CBDC by majority vote.*

Justification. A publicly visible voting about removal of a CBDC instance from the ecosystem is realized by IMSC contract that resides in *PB*, while each existing instance has a single vote. □

6.3 Security of TEE

We assume that its TEE platform employed is secure. However, previous research indicated that this might not be the case in practical implementations of TEE, such as SGX that was vulnerable to memory corruption attacks [8] as well as side channel attacks [13, 45, 54, 67]. A number of software-based defense and mitigation techniques have been proposed [13, 18, 31, 60, 62] and some vulnerabilities were patched by Intel at the hardware level [35]. Nevertheless, we note that our approach is TEE-agnostic thus can be integrated with other TEEs such as ARM TrustZone or RISC-V architectures (using Keystone-enclave [28] or Sanctum [21]).

Another class of SGX vulnerabilities was presented by Cloosters et al. [20] and involved incorrect application designs enabling arbitrary reads and writes of protected memory. Since the authors did not provide public with their tool (and moreover it does not support Open-enclave SDK), we did manual inspection of our code and did not find any of the concerned vulnerabilities. Another work was done by Borrello et al. [12] and involves more serious micro-architectural flaws in chip design. Intel has already released microcode and SGX SDK updates to fix the issue.

6.4 Public Blockchain & Finality

Many blockchain platforms suffer from accidental forks (i.e., availability-favored blockchains in terms of CAP theorem), which temporarily create parallel inconsistent blockchain views. To mitigate this phenomenon, it is recommended to wait a certain number of block confirmations after a given block is created before considering it irreversible with overwhelming probability. This waiting time (a.k.a., time to finality) influences the non-equivocation property of our approach, inheriting it from the underlying blockchain platform. Most availability-favored blockchains have a long time to finality, e.g., ∼3mins in Bitcoin [56], ∼3mins in Ethereum [**?**], ∼2mins in Cardano [37]. However, consistency-favored blockchains in terms of the CAP theorem have a short time to finality, e.g., HoneyBadgerBFT [51], Algorand [30], Hyperledger Besu [34]. The selection of the underlying blockchain platform should respect low time to finality in the critical environment of CBDC, and thus employ a consistency-favored public blockchain.

7 RELATED WORK

In this section, we first review various approaches to interoperability and CBDC. Moreover, since our protocol is designed using a combination of TEE and the blockchain, we revise the most relevant solutions and stress the novelty of our approach, which combines several unique features.

7.1 Blockchain Interoperability

Cross-chain interoperability is one of the most desirable yet challenging features to be designed and developed in blockchains, affecting the impact and usability of the solution [6, 52, 58, 69].

Cross-chain communication protocols define the process of synchronization between different chains of the same blockchain, e.g., by the use of sidechains. Additionally, cross-blockchain communication protocols, such as Interledger Protocol [63], allow interaction of different blockchains. While the cross-chain solutions can be employed by the native constructs such as atomic swap, the cross-blockchain protocols require adoption of the solution. Blockchain interoperability solutions can be categorized into three groups according to the principle they are based on and the type of chains that are supported [6].

Public connectors. Public connectors are a set of approaches that focuses on cryptocurrency systems and their transactions. This includes the sidechains, relays, notary schemes, and hash time locks [24, 52, 55].

Blockchain of blockchains. Blockchain of blockchains focuses on application specific-solutions. The example is Polkadot [6, 15] – a network for cross-blockchain interoperability. In Polkadot network, multiple parallelized globally-coherent chains (parachains) are connected via bridges that represent a specific type of parachain. Bridges also serve as a gateway for communication with external networks, such as Bitcoin.

Hybrid Connectors. Hybrid solutions create an abstraction layer over the blockchain ecosystem and provide a unified API for interaction between blockchain and applications [46]. Examples are trusted relays or blockchain migrators. The interoperability requires validators present in both the source and target blockchains. The validators collect cross-chain transactions and ensure that they are delivered [68].

The proposed solution in this paper contain a custom one-way atomic swap protocol that utilizes hash time lock contracts. Such swaps are settled on public blockchain *PB*. It is also expected that *PB* used for the synchronization of clients and CBDC instances deploys a single blockchain technology. The usability of the proposed solution targets the financial institutions such as banks, leveraging its potential use in CBDC projects. The protocol does not specify a middleware layer providing API or the use of gateway chains.

Therefore, it can be categorized as a public connector that augments and combines the features provided by individual solutions in the same category.

7.2 CBDC Projects

While most CBDC projects are still in their early stages, some well-known proposals are reaching maturity level [71]. For instance, Project Jasper [17] was one of the initial prototypes for inter-bank payments using blockchain technology. Project Ubin [53] appeared with the aim of clearing and settling of payments and securities efficiently by using several blockchain technologies and smart contracts. Project E-krona [4] was designed, among others, to enable fast transactions between domestic and cross-border entities. Stella [39] is another well-known project that uses permissioned blockchain technology to enable cross border operations as well as confidentiality protection. The mBridge project [9] (initially named Inthanon-LionRock) prototype is built by ConsenSys on Hyperledger Besu. The prototype encompasses several jurisdictions and aims at creating a cross-border payment infrastructure that improves on key pain points, including high cost, low speed, and operational complexities. Finally, Project Khokha [64] was designed for efficient, confidential inter-bank transactions.

Despite the maturity of some projects, research on CBDC technology is still in its infancy. In addition, the road to creating a native interoperable protocol that can be used regardless of the underlying blockchain technology still requires further exploration and is one of the main objectives of this article. Compared to other CBDC projects, our approach is the first protocol combining TEE and blockchain to bring interesting security and privacy features, accompanied by external interoperability. In detail, our protocol guarantees a set of features such as integrity, non-equivocation (i.e., we provide snapshots to public blockchain to avoid reverts and forks of the local CBDC ledgers), correctness (i.e., the EVM is executed in an enclave which can be remotely attested), and censorship evidence. Since the designed protocol addresses inter-bank communication and payment settlements, it can be potentially integrated as a part of the above-mentioned wholesale CBDC projects. The advantages of the retail CBDC approach towards individual clients are also preserved by the privacy support, censorship evidence and mitigation of malicious approach described in Section 6. The general approach is also invariant towards token differences introduced by different projects with regard to the public blockchain.

7.3 Combining Blockchain and TEE

The combination of Trusted Execution Environment (TEE) technologies and blockchain has gained increased attention in the past few years. Hybridchain [70] is an architecture for confidentiality-preserving in permissioned blockchain. Such architecture extends the enclave memory of TEE that allows blockchain applications running in TEE to securely store transaction records outside of TEE. Ekiden [19] is a blockchain-agnostic solution that offloads smart contract execution to TEE enclaves. Teechain [44] focuses on the Bitcoin network and enables the secure execution of transactions in TEE, enhancing the scalability of the network. Fastkitten enables extended functionality in the Bitcoin network by using Turing-complete smart contracts executed via TEE-enabled operators [23].

However, solutions combining interoperability with TEE-based blockchains are still in their infancy. Only a few authors have explored this such as Bellavista et al. [7], and Lan et al. [43], which are the works most similar to ours. More concretely, Bellavista et al. [7] explore the use of a relay scheme based on TEE to provide blockchain interoperability in the context of collaborative manufacturing and supply chains. Lan et al. [43] aim to preserve confidentiality in interoperable cross-chain platforms and propose a protocol to ensure privacy-preserving communications among them. Nevertheless, our approach is the first one designing a functional protocol for interoperable CBDC, considering features such as the ones mentioned in Section 3.

8 DISCUSSION

As seen in Section 7, this is the first blockchain TEE-based interoperable protocol that operates in the context of CBDC. However, our protocol allows modifications if additional requirements were to be fulfilled (i.e., considering the ones defined in Section 3). The latter enables a certain degree of dynamism when adapting the protocol to specific application contexts.

Following the interest of countries in CBDC [5], research on CBDCs and their potential challenges has also been receiving increasing attention in the last years (i.e., the number of contributions has been doubling yearly since 2020 according to Scopus, using the query TITLE-ABS-KEY (((central AND bank AND digital AND currency) OR CBDC) AND challenges). While a profound analysis of state of the art is out of the scope of this paper, we found that authors typically follow two strategies to discuss CBDC and its challenges, namely considering a local perspective (i.e., at a jurisdiction or national level) and adopting a global challenge abstraction. Overall, we considered the most recent reviews and surveys analyzing CBDC and its challenges [2, 16, 40, 59] and other grey literature, such as the Digital Euro Association [25], or the US federal reserve [66], to extract the challenges and represent them according to a high-level hierarchical abstraction. Since one of the aims of our proposal is to provide solutions to as many challenges as possible, we describe, for each challenge, the benefits and features that our proposal provides in Table 4.

9 CONCLUSION

Although the controversy surrounding the coexistence of privacy and CBDC [42], the latter promises a series of benefits, such as transaction efficiency (e.g., by reducing costs and decreasing its finality at the national or international level) and countering financial crime. Moreover, CBDC complements current financial services by offering broader opportunities. Nevertheless, the corresponding regulations should carefully manage these new opportunities, ensuring they do not restrict citizens' rights. Note that novel functionalities enforced in financial transactions, such as token expiration dates, negative interest rates for token holders (i.e., in an attempt to stimulate the economy in recession periods) or tokens whose validity is tied to a specific subset of goods (e.g., enforcing that part of the salary is spent on energy or healthcare), could either be applied for the sustainability of the society or state control in the context of authoritarian regimes.

Topic	Main concerns	Our proposal's contribution
Technology	The design, implementation and maintenance of CBDC's as well as their scalability, resiliency and compatibility with the current financial structure.	Our system is scalable and compatible with current financial system
Monetary Policy	Monetary policy transmission, including interest rates, the value of money, or other tools, should not be hindered by CBDC.	Our system relies on smart contracts to enforce specific policies if required, such as token expiration or token usability.
Financial Stability	The potentially disrupting impact of CBDC on the existing financial system could create new financial vulnerabilities, uncontrolled disintermediation or illicit activities.	The use of blockchain and the policies translated into the system should be audited and verified. Our system is compatible with the latter and other policies in the above layers.
Legal Framework	The legal framework for CBDC needs to comply with existing laws and regulations, including consumer protection, anti-money laundering, and countering the financing of terrorism.	The system is compatible with auditability layers compliant with current legal and regulations
Interoperability	Ensuring interoperability by guaranteeing that CBDCs are compatible with other countries monetary policies and promoting cross-border cooperation and standardisation.	Our protocol is interoperable by design and ensures the system remains in monetary equilibrium since no new tokens are created. The potential use of oracles enables further operations with different currencies beyond current ones, promoting cross-border cooperation and additional capabilities.
Security and Privacy	CBDC needs to ensure robustness to prevent cyberattacks and unauthorised access to data by guaranteeing privacy-preserving mechanisms of transactions and personal information.	Our proposal is robust and preserves the privacy of transactions since all the transactions are encrypted. We provide various security properties, such as atomicity, verifiability, integrity, non-equivocation, correctness of execution, censorship evidence, and others.
User Adoption and Inclusion	CBDCs will need to provide access to banking services to different populations. Users will require a behaviour change, acceptance and trust.	The use of our system is transparent to other layers, so it does not introduce any burden. TEE technologies enable trustable platforms, and our protocol allows verifiable censorship resolution.

Table 4: High-level abstraction of CBDC's challenges and how our proposal contributes to them. In some cases, our proposal slightly interferes with these challenges since many only apply to other CBDC ecosystem layers.

Given the above circumstances, we provide the design and implementation of the protocol that uses a custom adaptation of atomic swap and is executed by any pair of CBDC instances to realize a one-way transfer, resolving interoperability over multiple instances of semi-centralized CBDC. Our protocol guarantees a series of properties such as verifiability, atomicity of inter-bank transfers, censorship resistance, and privacy. Our contributions result in a step forward toward enriching the capabilities of CBDC and their practical deployment.

Future work will closer study token issuance management through protocol directives, perform more extensive evaluation, and propose interoperable execution of smart contracts between CBDC instances.

ACKNOWLEDGMENTS

This work was supported by the FIT BUT internal project FIT-S-23-8151. This work was also supported by the European Commission under the Horizon Europe Programme, as part of the LAZARUS project (http://lazarus-he.eu/ Grant Agreement no. 101070303). Fran Casino was supported by the Government of Catalonia with the Beatriu de Pinós programme (Grant No. 2020 BP 00035), and by AGAUR with the 2021-SGR-00111 project. The content of this article does not reflect the official opinion of the European Union. Responsibility for the information and views expressed therein lies entirely with the authors.

REFERENCES

[1] Mohammed AlShamsi, Mostafa Al-Emran, and Khaled Shaalan. 2022. A Systematic Review on Blockchain Adoption. *Applied Sciences* 12, 9 (2022), 4245.

[2] W.O. Alwago. 2022. Is the Renminbi a Global Currency in the Making? Globalization of Digital yuan. 67, 4 (2022), 553–566.

[3] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. 2013. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, Vol. 13. ACM New York, NY, USA.

[4] Hanna Armelius, Gabriela Guibourg, Stig Johansson, and Johan Schmalholz. 2020. E-krona design models: pros, cons and trade-offs. *Sveriges Riksbank Economic Review* 2 (2020), 80–96.

[5] Atlantic Council. [n. d.]. Central Bank Digital Currency Tracker. https://www.atlanticcouncil.org/cbdctracker/.

[6] Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia. 2021. A survey on blockchain interoperability: Past, present, and future trends. *ACM Computing Surveys (CSUR)* 54, 8 (2021), 1–41.

[7] Paolo Bellavista, Christian Esposito, Luca Foschini, Carlo Giannelli, Nicola Mazzocca, and Rebecca Montanari. 2021. Interoperable Blockchains for Highly-Integrated Supply Chains in Collaborative Manufacturing. *Sensors* 21, 15 (2021), 4955.

[8] Andrea Biondo, Mauro Conti, Lucas Davi, Tommaso Frassetto, and Ahmad-Reza Sadeghi. 2018. The Guard's Dilemma: Efficient Code-Reuse Attacks Against Intel {SGX}. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 1213–1227.

[9] BIS innovation hub. 2021. Inthanon-LionRock to mBridge: Building a multi CBDC platform for international payments. https://www.bis.org/publ/othp40.htm

[10] Bitcoin Wiki. 2018. Atomic Swap. https://en.bitcoinwiki.org/wiki/Atomic_Swap

[11] Codruta Boar and Andreas Wehrli. 2021. Ready, steady, go?-Results of the third BIS survey on central bank digital currency. (2021).

[12] Pietro Borrello, Andreas Kogler, Martin Schwarzl, Moritz Lipp, Daniel Gruss, and Michael Schwarz. 2022. ÆPIC Leak: Architecturally Leaking Uninitialized Data from the Microarchitecture. In *31st USENIX Security Symposium (USENIX Security 22)*.

[13] Ferdinand Brasser, Srdjan Capkun, Alexandra Dmitrienko, Tommaso Frassetto, Kari Kostiainen, Urs Müller, and Ahmad-Reza Sadeghi. 2017. DR. SGX: hardening SGX enclaves against cache attacks with data location randomization. *arXiv preprint arXiv:1709.09917* (2017).

[14] Richard Gendal Brown, James Carlyle, Ian Grigg, and Mike Hearn. 2016. Corda: an introduction. *R3 CEV, August* 1, 15 (2016), 14.

[15] Jeff Burdges, Alfonso Cevallos, Peter Czaban, Rob Habermeier, Syed Hosseini, Fabio Lama, Handan Kilinc Alper, Ximin Luo, Fatemeh Shirazi, Alistair Stewart, and Gavin Wood. 2020. Overview of Polkadot and its Design Considerations. https://doi.org/10.48550/ARXIV.2005.13456

[16] C. Catalini, A. De Gortari, and N. Shah. 2022. Some Simple Economics of Stablecoins. 14 (2022), 117–135.

[17] James Chapman, Rodney Garratt, Scott Hendry, Andrew McCormack, and Wade McMahon. 2017. Project Jasper: Are distributed wholesale payment systems feasible yet. *Financial System* 59 (2017).

[18] Sanchuan Chen, Xiaokuan Zhang, Michael K Reiter, and Yinqian Zhang. 2017. Detecting privileged side-channel attacks in shielded execution with Déjá Vu. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. 7–18.

[19] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. 2018. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution. *arXiv preprint arXiv:1804.05141* (2018).

[20] Tobias Cloosters, Michael Rodler, and Lucas Davi. 2020. TeeRex: Discovery and Exploitation of Memory Corruption Vulnerabilities in {SGX} Enclaves. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 841–858.

[21] Victor Costan, Ilia Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal hardware extensions for strong software isolation. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 857–874.

[22] Scott A Crosby and Dan S Wallach. 2009. Efficient Data Structures For Tamper-Evident Logging.. In *USENIX Security Symposium*. 317–334.

[23] Poulami Das, Lisa Eckey, Tommaso Frassetto, David Gens, Kristina Hostáková, Patrick Jauernig, Sebastian Faust, and Ahmad-Reza Sadeghi. 2019. {FastKitten}: Practical Smart Contracts on Bitcoin. In *28th USENIX Security Symposium (USENIX Security 19)*. 801–818.

[24] Snoviya Dcunha, Srushti Patel, Shravani Sawant, Varsha Kulkarni, and Mahesh Shirole. 2021. Blockchain Interoperability Using Hash Time Locks. In *Proceeding of Fifth International Conference on Microelectronics, Computing and Communication Systems*, Vijay Nath and J. K. Mandal (Eds.). Springer Singapore, Singapore, 475–487.

[25] Digital Euro Association. [n. d.]. Central Bank Digital Currency. https://home.digital-euro-association.de/cbdc/en.

[26] Digital Euro Association. 2022. The CBDC Manifesto. https://cbdcmanifesto.com/

[27] Jonathan Ehrenfeld. 2022. SWIFT partners with Chainlink for cross-chain crypto transfer project. https://blog.chain.link/smartcon-2022-recap/

[28] Keystone Enclave. 2019. Keystone: An Open Framework for Architecting Trusted Execution Environments. https://keystone-enclave.github.io/.

[29] Thomas Espel, Laurent Katz, and Guillaume Robin. 2017. Proposal for Protocol on a Quorum Blockchain with Zero Knowledge. 2017 (2017).

[30] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP*.

[31] Daniel Gruss, Julian Lettner, Felix Schuster, Olya Ohrimenko, Istvan Haller, and Manuel Costa. 2017. Strong and efficient cache side-channel protection using hardware transactional memory. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 217–233.

[32] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. 2013. Using innovative instructions to create trustworthy software solutions. *HASP@ ISCA* 11 (2013).

[33] Ivan Homoliak and Pawel Szalachowski. 2020. Aquareum: A Centralized Ledger Enhanced with Blockchain and Trusted Computing. arXiv:2005.13339 [cs.CR]

[34] Hyperledger Foundation. 2022. Hyperledger. https://github.com/hyperledger.

[35] Intel. 2018. Resources and Response to Side Channel L1 Terminal Fault. https://www.intel.com/content/www/us/en/architecture-and-technology/l1tf.html

[36] Si Yuan Jin and Yong Xia. 2022. CEV Framework: A Central Bank Digital Currency Evaluation and Verification Framework With a Focus on Consensus Algorithms and Operating Architectures. *IEEE Access* 10 (2022), 63698–63714. https://doi.org/10.1109/ACCESS.2022.3183092

[37] Aggelos Kiayias, Al'exander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO'17*.

[38] Mr John Kiff, Jihad Alwazir, Sonja Davidovic, Aquiles Farias, Mr Ashraf Khan, Mr Tanai Khiaonarong, Majid Malaika, Mr Hunter K Monroe, Nobu Sugimoto, Hervé Tourpe, et al. 2020. A survey of research on retail central bank digital currency. (2020).

[39] Michinobu Kishi. 2019. Project Stella and the impacts of fintech on financial infrastructures in Japan.

[40] V. Koziuk. 2021. Confidence in digital money: Are central banks more trusted than age is matter? 18, 1 (2021), 12–32.

[41] Marc Lacoste and Vincent Lefebvre. 2023. Trusted Execution Environments for Telecoms: Strengths, Weaknesses, Opportunities, and Threats. *IEEE Security & Privacy* (2023).

[42] Christine Lagarde. 2022. High level conference: Towards a legislative framework enabling a digital euro for citizens and businesses. https://www.ecb.europa.eu/press/key/date/2022/html/ecb.sp221107~dcc0cd8ed9.en.html

[43] Ying Lan et al. 2021. TrustCross: Enabling Confidential Interoperability across Blockchains Using Trusted Hardware. In *2021 4th International Conference on Blockchain Technology and Applications* (Xi'an, China) *(ICBTA 2021)*. Association for Computing Machinery, New York, NY, USA, 17–23. https://doi.org/10.1145/3510487.3510491

[44] Joshua Lind, Ittay Eyal, Florian Kelbert, Oded Naor, Peter Pietzuch, and Emin Gün Sirer. 2017. Teechain: Scalable blockchain payments using trusted execution environments. *arXiv preprint arXiv:1707.05454* (2017).

[45] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. 2021. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE.

[46] Mohammad Madine, Khaled Salah, Raja Jayaraman, Yousof Al-Hammadi, Junaid Arshad, and Ibrar Yaqoob. 2021. appXchain: Application-Level Interoperability for Blockchain Networks. *IEEE Access* 9 (2021), 87777–87791. https://doi.org/10.1109/ACCESS.2021.3089603

[47] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. 2013. Innovative instructions and software model for isolated execution. *Hasp@isca* 10, 1 (2013).

[48] Ralph C Merkle. 1989. A certified digital signature. In *Conference on the Theory and Application of Cryptology*. Springer, 218–238.

[49] Silvio Micali. 2003. Simple and fast optimistic protocols for fair electronic exchange. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*. ACM, 12–19.

[50] Microsoft. 2020. Enclave EVM. https://github.com/microsoft/eEVM

[51] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honeybadger of BFT protocols. In *ACM CCS*.

[52] Debasis Mohanty, Divya Anand, Hani Moaiteq Aljahdali, and Santos Gracia Villar. 2022. Blockchain Interoperability: Towards a Sustainable Payment System. *Sustainability* 14, 2 (2022), 913.

[53] Monetary Authority of Singapore. 2016. Project Ubin. https://www.mas.gov.sg/schemes-and-initiatives/Project-Ubin

[54] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. 2020. Plundervolt: Software-based Fault Injection Attacks against Intel SGX. In *Proceedings of the 41st IEEE Symposium on Security and Privacy (S&P'20)*.

[55] Blessing Musungate, Busra Candan, Umut Cabuk, and Gokhan Dalkilic. 2019. Sidechains: Highlights and Challenges. https://doi.org/10.1109/ASYU48272.2019.8946384

[56] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system.

[57] Sergio Luis Náñez Alonso, Miguel Ángel Echarte Fernández, David Sanz Bas, and Jarosław Kaczmarek. 2020. Reasons fostering or discouraging the implementation of central bank-backed digital currency: A review. *Economies* 8, 2 (2020), 41.

[58] Ilham A Qasse, Manar Abu Talib, and Qassim Nasir. 2019. Inter blockchain communication: A survey. In *Proceedings of the ArabWIC 6th Annual International Conference Research Track*. 1–6.

[59] H.M.C.V. Sebastião, P.J.O.R. da Cunha, and P.M.C. Godinho. 2021. Cryptocurrencies and blockchain. Overview and future perspectives. 21, 3 (2021), 305–342.

[60] Jaebaek Seo, Byoungyoung Lee, Seong Min Kim, Ming-Wei Shih, Insik Shin, Dongsu Han, and Taesoo Kim. 2017. SGX-Shield: Enabling Address Space Layout Randomization for SGX Programs.. In *NDSS*.

[61] Vijak Sethaput and Supachate Innet. 2021. Blockchain Application for Central Bank Digital Currencies (CBDC). In *2021 Third International Conference on Blockchain Computing and Applications (BCCA)*. IEEE, 3–10.

[62] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. 2017. T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs.. In *NDSS*.

[63] Vasilios Siris, Pekka Nikander, Spyros Voulgaris, Nikos Fotiou, Dmitrij Lagutin, and George Polyzos. 2019. Interledger Approaches. *IEEE Access* PP (07 2019), 1–1. https://doi.org/10.1109/ACCESS.2019.2926880

[64] South African Reserve Bank. 2018. Project Khokha: Blockchain Case Study for Central Banking in South Africa . https://consensys.net/blockchain-use-cases/finance/project-khokha/

[65] Pramod Subramanyan, Rohit Sinha, Ilia Lebedev, Srinivas Devadas, and Sanjit A Seshia. 2017. A formal foundation for secure remote execution of enclaves. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2435–2450.

[66] US Federal Reserve. [n. d.]. Central Bank Digital Currency. https://www.federalreserve.gov/central-bank-digital-currency.htm.

[67] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 991–1008.

[68] Gang Wang and Mark Nixon. 2021. InterTrust: Towards an Efficient Blockchain Interoperability Architecture with Trusted Services. In *2021 IEEE International Conference on Blockchain (Blockchain)*. 150–159. https://doi.org/10.1109/Blockchain53845.2021.00029

[69] Gang Wang, Qin Wang, and Shiping Chen. 2023. Exploring Blockchains Interoperability: A Systematic Survey. *Comput. Surveys* (2023).

[70] Yong Wang, June Li, Siyu Zhao, and Fajiang Yu. 2020. Hybridchain: A novel architecture for confidentiality-preserving and performant permissioned blockchain using trusted execution environment. *IEEE Access* 8 (2020), 190652–190662.

[71] Tao Zhang and Zhigang Huang. 2021. Blockchain and central bank digital currency. *ICT Express* (2021).

Algorithm 6: The program $prog^{\mathbb{IPSC}}$ of $\mathbb{IPSC}$.

```
▷ Declaration of types and constants:
  CensInfo { $\mu$-$etx$, $\mu$-$equery$, $status$, $edata$ },
  $msg$: a current transaction that called $\mathbb{IPSC}$,
▷ Declaration of functions:
function $Init(PK_{\mathbb{E}}^{pb}, PK_{\mathbb{E}}^{tee}, PK_{\mathbb{O}}, \_i_r, [ia \leftarrow T])$ public
    $PK_{\mathbb{E}}^{tee}[].add(PK_{\mathbb{E}}^{tee})$;          ▷ PK of enclave $\mathbb{E}$ under $\Sigma_{tee}$.
    $PK_{\mathbb{E}}^{pb}[].add(PK_{\mathbb{E}}^{pb})$;            ▷ PK of enclave $\mathbb{E}$ under $\Sigma_{pb}$.
    $PK_{\mathbb{O}}^{pb} \leftarrow PK_{\mathbb{O}}$;             ▷ PK of operator $\mathbb{O}$ under $\Sigma_{pb}$.
    $LRoot_{pb} \leftarrow \perp$;    ▷ The most recent root hash of L synchronized with $\mathbb{IPSC}$.
    $censReqs \leftarrow []$;                ▷ Request that $\mathbb{C}$s wants to resolve publicly.
    $t_s \leftarrow 0$;                          ▷ The total supply of the instance.
    $t_i \leftarrow 0$;                      ▷ The total issued tokens by the instance.
    const $issueAuthority \leftarrow ia$;    ▷ Token issuance capability of the instance.
    const $i_r \leftarrow \_i_r$;                ▷ Max. yearly inflation of the instance.
    const $createdAt \leftarrow timestamp()$; ▷ The timestamp of creation a CBDC instance.
function $snapshotLedger(root_A, root_B, \_t_i, \_t_s, \sigma)$ public
                     ▷ Verify whether msg was signed by $\mathbb{E}$.
    assert $\Sigma_{pb}.verify((\sigma, PK_{\mathbb{E}}^{pb}[\text{-}1]), (root_A, root_B, \_t_i, \_t_s))$;
                     ▷ Snapshot issued tokens and total supply.
    if $issueAuthority$ then
        assert $\_\_meetsInflationRate(\_t_i)$;   ▷ The code is trivial, and we omit it.
        $t_i \leftarrow \_t_i$;
    else
        assert $t_i = \_t_i$;
                 ▷ Verify whether a version transition extends the last one.
    if $LRoot_{pb} = root_A$ then
        $LRoot_{pb} \leftarrow root_B$;                 ▷ Do a version transition of L.
function $SubmitCensTx(\mu\text{-}etx, \sigma_{msg})$ public
             ▷ Called by $\mathbb{C}$ in the case her $\mu$-tx is censored. $\mathbb{C}$ encrypts it by $PK_{\mathbb{E}}^{tee}$.
    accessControl($\sigma_{msg}$, $msg.PK_{\mathbb{C}}^{pb}$);
    $censReqs$.add(CensInfo($\mu$-$etx$, $\perp$, $\perp$, $\perp$));
function $ResolveCensTx(idx_{req}, status, \sigma)$ public
               ▷ Called by $\mathbb{O}$ to prove that $\mathbb{C}$'s $\mu$-tx was processed.
    assert $idx_{req} < |censReqs|$;
    $r \leftarrow censReqs[idx_{req}]$;
    assert $\Sigma_{pb}.verify((\sigma, PK_{\mathbb{E}}^{pb}[\text{-}1]),\ (h(r.\mu\text{-}etx), status))$;
    $r.status \leftarrow status$;
function $SubmitCensQry(\mu\text{-}equery, \sigma_{msg})$ public
               ▷ Called by $\mathbb{C}$ in the case its read query is censored. $\mathbb{C}$ encrypts it by $PK_{\mathbb{E}}^{tee}$.
    accessControl($msg$, $\sigma_{msg}$, $msg.PK_{\mathbb{C}}^{pb}$);
    $censReqs$.add(CensInfo($\perp$, $\mu$-$equery$, $\perp$, $\perp$));
function $ResolveCensQry(idx_{req}, status, edata, \sigma)$ public
             ▷ Called by $\mathbb{O}$ as a response to the $\mathbb{C}$'s censored read query.
    assert $idx_{req} < |censReqs|$;
    $r \leftarrow censReqs[idx_{req}]$;
    assert $\Sigma_{pb}.verify((\sigma, PK_{\mathbb{E}}^{pb}[\text{-}1]), (h(r.\mu\text{-}equery), status, h(edata)))$;
    $r.\{edata \leftarrow edata, status \leftarrow status\}$;
function $ReplaceEnc(PKN_{\mathbb{E}}^{pb}, PKN_{\mathbb{E}}^{tee}, r_A, r_B, \_t_i, \_t_s, \sigma, \sigma_{msg})$ public
                     ▷ Called by $\mathbb{O}$ in the case of enclave failure.
    assert $\Sigma_{pb}.verify((\sigma_{msg}, PK_{\mathbb{O}}^{pb}), msg)$;      ▷ Avoiding MiTM attack.
    $snapshotLedger(r_A, r_B, \_t_i, \_t_s, \sigma)$ ;               ▷ Do a version transition.
    $PK_{\mathbb{E}}^{tee}.add(PKN_{\mathbb{E}}^{tee})$;         ▷ Upon change, $\mathbb{C}$s make remote attestation.
    $PK_{\mathbb{E}}^{pb}.add(PKN_{\mathbb{E}}^{pb})$;
```

Algorithm 5: The program $prog^{\mathbb{E}}$ of enclave $\mathbb{E}$

```
▷ Declaration of types and functions:
  Header { $ID$, $txsRoot$, $rcpRoot$, $stRoot$};
  $\#(r) \longrightarrow v$: denotes the version $v$ of $L$ having $LRoot = r$,
▷ Variables of TEE:
  $SK_{\mathbb{E}}^{tee}, PK_{\mathbb{E}}^{tee}$: keypair of $\mathbb{E}$ under $\Sigma_{tee}$,
  $SK_{\mathbb{E}}^{pb}, PK_{\mathbb{E}}^{pb}$: keypair of $\mathbb{E}$ under $\Sigma_{pb}$,
  $hdr_{last} \leftarrow \perp$: the last header created by $\mathbb{E}$,
  $LRoot_{pb} \leftarrow \perp$: the last root of $L$ flushed to PB's $\mathbb{IPSC}$,
  $LRoot_{cur} \leftarrow \perp$: the root of $L \cup blks_p$ (not flushed to PB),
  $ID_{cur} \leftarrow 1$: the current version of $L$ (not flushed to PB),
  $FH_{cur} \leftarrow []$: the frozen hashes cache of the current $L$'s history tree.
▷ Declaration of functions:
function $Init()$ public
    $(SK_{\mathbb{E}}^{pb}, PK_{\mathbb{E}}^{pb}) \leftarrow \Sigma_{pb}.Keygen()$;
    $(SK_{\mathbb{E}}^{tee}, PK_{\mathbb{E}}^{tee}) \leftarrow \Sigma_{tee}.Keygen()$;
    Output($PK_{\mathbb{E}}^{tee}$, $PK_{\mathbb{E}}^{pb}$);
function $Exec(txs[], \partial st^{old})$ public
    assert $\partial st^{old}.root = hdr_{last}.stRoot$;
    $\partial st^{new}, rcps, txs_{er} \leftarrow \_\_processTxs(txs,\ \partial st^{old})$;
    $\sigma \leftarrow \Sigma_{pb}.sign(SK_{\mathbb{E}}^{pb}, (LRoot_{pb}, LRoot_{cur}))$;
    Output($LRoot_{pb}$, $LRoot_{cur}$, $\partial st^{new}$, $hdr_{last}$, $rcps$, $txs_{er}$, $\sigma$);
function $Flush()$ public
    $LRoot_{pb} \leftarrow LRoot_{cur}$;       ▷ Shift the version of L synchronized with PB.
function $\_\_processTxs(txs[], \partial st^{old})$ private
    $\partial st^{new}, rcps[], txs_{er} \leftarrow$ runVM($txs$, $\partial st^{old}$);          ▷ Run $\mu$-$txs$ in VM.
    $txs \leftarrow txs \setminus txs_{er}$;          ▷ Filter out parsing errors/wrong signatures.
    $hdr \leftarrow$ Header($ID_{cur}, MkRoot(txs), MkRoot(rcps), \partial st^{new}.root$));
    $hdr_{last} \leftarrow hdr$;
    $LRoot_{cur} \leftarrow \_\_newLRoot(hdr)$;
    $ID_{cur} \leftarrow ID_{cur} + 1$;
    return $\partial st^{new}, rcps, txs_{er}$;
function $\_\_newLRoot(hdr)$ private
    $\_\_udpateFH(h(hdr))$;
    return $FH_{cur}.ReduceRoot()$;
                    ▷ Since $FH_{cur} = \pi_{next}^{inc}$, inc. proof. for 1 element commitment.
function $\_\_updateFH(hdrH)$ private
    $FH_{cur}.add(hdrH)$;
    $l \leftarrow \lfloor log_2(ID_{cur}) \rfloor$;
    for $i \leftarrow 2$; $i \leq 2^l$; $i \leftarrow 2i$ do
        if $0 = ID_{cur}$ mod $i$ then
            $FH_{cur}[\text{-}2] \leftarrow h(FH_{cur}[\text{-}2]\ ||\ FH_{cur}[\text{-}1])$
            delete $FH_{cur}[\text{-}1]$;                    ▷ Remove the last element.
```