VĚDECKÉ SPISY VYSOKÉHO UČENÍ TECHNICKÉHO V BRNĚ

Edice Habilitační a inaugurační spisy, sv. 425
ISSN 1213-418X

Róbert Lórencz

EFEKTIVNÍ HARDWAROVÁ IMPLEMENTACE MULTIPLIKATIVNÍ MODULÁRNÍ INVERZE NAD GF(p) PRO KRYPTOGRAFICKÁ PRIMITIVA

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

doc. Ing. Róbert Lórencz, CSc.

EFEKTIVNÍ HARDWAROVÁ IMPLEMENTACE MULTIPLIKATIVNÍ MODULÁRNÍ INVERZE NAD GF(p) PRO KRYPTOGRAFICKÁ PRIMITIVA

EFFICIENT HARDWARE IMPLEMENTATION OF MULTIPLICATIVE MODULAR INVERSION OVER GF(p) FOR CRYPTOGRAPHIC PRIMITIVES

TEZE PŘEDNÁŠKY K PROFESORSKÉMU JMENOVACÍMU ŘÍZENÍ V OBORU VÝPOČETNÍ TECHNIKA A INFORMATIKA

VUTIUM

BRNO 2012

KLÍČOVÁ SLOVA
kryptografie; kryptografická primitiva; modulární aritmetika; binární rozšířený Euklidův algoritmus; multiplikativní modulární inverze; hardware; ASIC; FPGA

KEYWORDS
cryptography; cryptographical primitives; modular arithmetic; binary extended Euclidean algorithm; multiplicative modular inversion; hardware; ASIC; FPGA

ISBN 978-80-214-4522-2
ISSN 1213-418X

Obsah

Představení autora

doc. Ing. Róbert Lórencz, CSc.

Narozen:	10. 8. 1957, Prešov, SR
Vzdělání:	1976 Gymnázium, Prešov, SR
	1981 FEL ČVUT Praha, Ing. (Sdělovací technika)
	1991 ÚM SAV, CSc. (Meracia a výpočtová technika)
Habilitace:	2005 FEL ČVUT, doc. (Výpočetní technika a informatika)
Zaměstnání:	1982 NC KU Praha (výzkumný pracovník)
	1983 FÚ SAV Bratislava (výzkumný pracovník)
	1990 L. M. Universität Mnichov (postdoc)
	1992 Škoda auto a.s. (systémový analytik)
	1995 FEI TU Košice (výzkumný pracovník)
	1998 FEL ČVUT (akademický pracovník)
	2009 – dosud FIT ČVUT (akademický pracovník)

Zahraniční pobyty a stáže: ICTP Terst, Itálie (1 měsíc), GSI Darmstadt, Německo (1 měsíc), L. M. Universität Mnichov, Nemecko (1 rok), DESY Hamburg, Německo (2 roky)

Vědecká činnost

Autor nebo spoluautor 8 článků v časopisech s IF, 4 článků ve vědeckých časopisech bez IF, 40 příspěvků na mezinárodních konferencích, desítek příspěvků na národních konferencích, 1 US patentu a 2 tuzemských patentů. Recenzní činnost pro 2 IEEE časopisy, 2 zahraniční časopisy s IF a zahraniční grantovou agenturu. Práce byly dosud citovány víc než 130× jinými autory a z toho 44× v publikacích uvedených v databázi WoS (SCI citace). Přehled vybraných řešených výzkumných projektů:

- 1990-91 TAPS projekt, GSI Darmstadt, Německo (člen řešitelského týmu).
- 1996-98 Extrakce parametrů pomoci SPICE modelů, FEC DESY, Německo (řešitel).
- 2002-03 Spolupráce odborných vysokých škol v boji státu s počítačovou kriminalitou. Nejzávažnější bezpečnostní rizika, grant MV ČR (člen řešitelského týmu).
- 2005-11 Výzkumný záměr, Výzkum perspektivních informačních a komunikačních technologií, (člen řešitelského týmu v kategorii D1).
- 2007-10 Problematika kybernetických hrozeb z hlediska bezpečnostních zájmů ČR, grant MV ČR (spoluřešitel).
- 2012-14 Studium vlastností residuální aritmetiky pro řešení soustav lineárních rovnic, GA ČR, (řešitel).

Pedagogická činnost

Zavedení nových předmětů, přednášky v předmětech: Architektura počítačových systémů, Bezpečnost přenosu a zpracování dat, Aplikovaná kryptografie, Pokročilá kryptologie, Strojová kód a data, Aplikovaná numerická matematika a částečně předmět Bezpečnost a hardware v studijních programech FEL a FIT ČVUT. Zavedení a garance oboru Informační technologie a oboru Počítačová bezpečnost v bakalářském resp. magisterském studijním programu Informatika na FIT ČVUT. Autor skript. Vedení 4 obhájených disertačních prací, členství v oborových komisích oboru a programu doktorského studia na FEL a FIT (místopředseda ORP) ČVUT. Garant, řešitel a člen řešitelského týmů 6 projektů na rozvoj výuky, podporu doktorského studia a rozvoj infrastruktury pro doktorské studium v objemu cca 2 mil. Kč.

Další aktivity

Proděkan pro vědu a výzkum, vedoucí katedry počítačových systémů a vedoucí výzkumné skupiny Aplikovaná numerická matematika a kryptografie na FIT ČVUT. Soudní znalec. Člen vědecké rady FIT a FEL ČVUT, člen redakční rady zahraničního časopisu (ACM Slovakia).

1 Úvod

S rozšiřováním služeb a produktů v oblasti informačních technologií (IT) pro veřejnost, komerční sektor a státní správu rostou k tomu úměrně i požadavky na cenovou přístupnost, spolehlivost, a také bezpečnost těchto služeb a produktů. Právě bezpečnost je čím dále tím více jedním z nejdůležitějších a nejsledovanějších parametrů v IT. To souvisí především s právní, sociální a ekonomickou stránkou používání IT. Bezpečnost při používání IT technologií zahrnuje technologická — systémová a organizační opatření. Návrh a používání jednotlivých kryptografických systémů, které splňují bezpečnostní pravidla prostředí, ve kterém jsou nasazeny, mají své konkrétní metodiky, doporučení a normy.

Implementace kryptografických systémů a jejich primitiv v dedikovaném hardwaru (např. u koprocesorů) je častokrát optimální řešení, které nejlépe zohledňuje většinu požadavků na ně kladených. Návrh takových hardwarových systémů by měl splňovat co nejvíce z následujících parametrů: malá časová složitost, malá prostorová složitost, nízká spotřeba, odolnost systému vůči kryptoanalýze pomocí vyzařování postranními kanály, bezpečné uchovávání dat, spolehlivost, jednoduché rozhraní, malá cena. Bezpečnostní hledisko systémů závisí hlavně na výběru kryptografických primitiv a jejich výpočetních algoritmů. Algoritmy používané v kryptografických primitivech provádí výpočty ve valné většině případů v modulární aritmetice nad tělesem GF(p), nebo v principiálně příbuzných aritmetikách, které umožňují snadnou implementaci některých výpočetních operací v hardware, a také software (např. polynomiální báze, Montgomeryho báze). Požadavek efektivního vykonávání operací kryptografických primitiv implementovaných v hardware je ekvivalentní požadavku dosažení menších hodnot tzv. implementačního koeficientu — „time×area product – T×A“. Tento koeficient je součinem časové a prostorové složitosti implementovaného algoritmu a jeho nižší hodnota znamená „úspěšnější“ implementaci v dané technologii. S nižším koeficientem T×A souvisí i nižší spotřeba a nižší cena kryptosystému. Nezanedbatelný je vliv nízké hodnoty tohoto koeficientu také na bezpečnost, a to v souvislosti s vyzařováním informace postranními kanály. V této souvislosti je zřejmé, že nižší energetická a časová náročnost výpočtu (nižší T×A) dává méně možností pro odposlech důležitých informací pomocí postranních kanálů v průběhu provozu kryptografického systému. Nízké T×A lze také dosáhnout použitím moderních technologických bází jako jsou ASIC (Application Specific Integrated Circuit) a FPGA (Field Programmable Gate Array), ale použití vhodných algoritmů respektujících vlastnosti technologických bází zůstává i nadále velmi důležitým nástrojem pro dosažení malé hodnoty T×A.

Výpočetně nejsložitější operací v modulární aritmetice je multiplikativní modulární inverze — MMI (Modular Multiplicative Inverse). Tato operace se používá v mnohých kryptografických primitivech, např. RSA (Rivest-Shamir-Adleman) a DSA (Digital Signature Algorithm) a v posledních několika letech zejména v kryptografii využívající vlastnosti eliptických křivek — EC (Elliptic Curve). Na rozdíl od kryptografie založené na EC — ECC (Elliptic Curve Cryptography), kde MMI je v algoritmu násobícího bod na EC v afinních souřadnicích počítána téměř polovinu výpočetního času, je v ostatních kryptografických primitivech MMI počítána jednou, dvakrát (RSA, DSA, ...), nebo maximálně několikrát, případ RSA-CRT (RSA — Chinese Remainder Theorem). Využití EC v kompletních asymetrických kryptografických schématech (výměna klíčů, digitální podpis, atd.) je již dlouhodobě aktuální z mnoha důvodů. Jednak principy ECC nabízí větší odolnost vůči různým

typům útoků, a dále v ECC pro stejnou úroveň bezpečnosti stačí několikanásobně menší počet bitů klíčů než u jiných typů kryptografických systémů (např. RSA), což má za následek menší prostorovou složitost implementace. Nezanedbatelnou výhodou ECC je také možnost efektivní hardwarové implementace, která upřednostňuje používání principů ECC v aplikacích, kde je důležitá velmi nízká spotřeba energie, jako jsou koprocesory různých čipových karet a také aplikace jako elektronické pasy.

Vzhledem k tomu, že klíčovou operací při použití ECC je operace MMI, závisí na jejím efektivním výpočtu výběr souřadnicového systému v ECC, a také i celková koncepce používání ECC v daném kryptografickém systému. V souvislosti s častým výskytem MMI v ECC je hodnota parametru $T \times A$, jak celkového, tak také pro MMI častokrát vodítkem při rozhodování o metodě přístupu při implementaci ECC pro daný kryptografický systém. Cílem této práce je představení některých efektivních implementací MMI v hardwarových technologických bázích FPGA a ASIC. Samotné výpočetní algoritmy MMI jsou založeny na rozšířeném Euklidově algoritmu — EEA (Extended Euclidean Algorithm). Dílčí cíle se věnují následujícím oblastem:

- Stručný přehled vlastností EEA a jeho použití pro výpočet MMI.
- Popis základních binárních algoritmů pro výpočet MMI s použitím EEA.
- Uvedení nového LS (Left-Shift) binárního algoritmu a jeho implementace.
- Popis upraveného algoritmu pro výpočet Montgomeryho MMI.

2 Vlastnosti rozšířeného Euklidova algoritmu a jeho použití pro výpočet multiplikativní modulární inverze

V jednotlivých částech této kapitoly je uveden základní matematický aparát pro popis Euklidova algoritmu, a dále jeho rozšířené a binární verze. Věty jsou uvedeny bez důkazů, které nalezneme spolu s dalšími informacemi v [15, 26, 45].

2.1 Multiplikativní modulární inverze v kryptografii

Základní aritmetické operace sčítání, odečítání, násobení a operace Multiplikativní Modulární Inverze – MMI prováděné v modulární aritmetice nad Galoisovým tělesem GF(p), kde p je prvočíslo, jsou součástí operací různých algoritmů, a to jak v symetrické, tak také v asymetrické kryptografii. V asymetrické kryptografii jsou to například: RSA algoritmus [44, 46], Diffie-Hellman algoritmus výměny klíče [11], algoritmus digitálního podpisu DSA [42], a také v ECC [27, 40]. Výpočetní složitost současně známých algoritmů pro výpočet MMI, a to nejen nad tělesem GF(p) je stále větší než složitost jiných základních operací [26, 29]. Na druhé straně je frekvence výskytu výpočtu MMI v kryptografických primitivech menší než výskyt jiných operací. Jsou ale speciální algoritmy, ve kterých se výpočet MMI často vyskytuje, například v kryptografii veřejného klíče, urychlení operace umocňování pomocí takzvaného „addition-subtraction chains“ [13, 28, 26], nebo při výpočtu násobení bodů na EC nad tělesem GP(p) v ECC [27, 40]. Je zřejmé, že v takových případech i malé vylepšení výpočtu MMI má signifikantní vliv na celkovou efektivitu výpočtu.

Hardwarová implementace některých algoritmů pro výpočet MMI v různých technologiích jako jsou ASIC a FPGA může znamenat velmi výhodné splnění požadavků kladených na daný kryptografický systém obsahující kryptografická primitiva s MMI. Je to především požadavek dosažení co nejmenšího parametru T×A, což má za následek nižší energetickou náročnost zařízení, a tak větší odolnost vůči kryptoanalýze pomocí postranních kanálů a větší bezpečnost. Nižší T×A má vliv i na nízkou cenu, a tedy i větší dostupnost pro co nejširší okruh uživatelů.

Byly vyvinuty různé přístupy pro výpočet MMI, a to jak pro software, tak také pro hardware [4, 12, 15, 16, 22, 25, 26, 51]. Nejznámější a nejpoužívanější algoritmus pro výpočet MMI je založen na rozšířeném Euklidově algoritmu — EEA. Jeho modifikace pro hardwarovou implementaci se nazývají binárními algoritmy MMI založenými na EEA. Při řešení hardwarové architektury kryptografických systémů s MMI je častokrát výhodné vybudovat výpočetní jednotku na bázi vybrané architektury určené pro výpočet MMI. Je to důsledek toho, že MMI má nejsložitější nejen časovou, ale i prostorovou složitost, a tak ostatní operace (sčítání, odečítání a násobení) můžou byt dobudovány již na hardware MMI. Vzhledem ke složitosti výpočtu inverze jsou algoritmy některých hardwarových implementací (výjimečně také softwarových) a jejich operace založené na různých matematických bázích s cílem vyhnout se výpočtu inverze. Příkladem může být normální báze nad tělesem F_{2^m}, kde je výpočet inverzního prvku prováděn pomocí malé Fermatovy věty algoritmem Itoh-Teechai-Tsujii [24].

Cílem této práce je představení dvou binárních algoritmů pro výpočet MMI založených na EEA s jejich efektivní implementací v technologických bázích FPGA a ASIC. První je takzvaný „Left-Shift — LS“ binární algoritmus a druhý je upravený algoritmus pro výpočet Montgomeryho

inverze takzvaný „Subtraction-free Almost Montgomery Inverse — SF-AMI“. U obou algoritmů je kladen velký důraz na dosažení co nejmenší časové a současně prostorové složitosti (parametr T×A). Architektury těchto algoritmů mohou být vhodným základem pro realizaci dalších aritmetických operací [23, 34].

2.2 Multiplikativní modulární inverze — matematický základ

V této podkapitole je uveden stručný přehled základních matematických pojmů, definic a vět týkajících se multiplikativní modulární inverze. Dále platí, že označení $|a|_m$, kde a a m jsou celá čísla a $m > 1$, znamená nejmenší nezáporný zbytek a modulo m.

Definice 2.1 *Když m je prvočíslo a když $b \in \mathcal{Z}_m, b \neq 0$, potom existuje jedinečné celé číslo $c \in \mathcal{Z}_m$, pro které platí:*

$$|c\,b|_m = |b\,c|_m = 1. \tag{1}$$

Číslo c nazýváme multiplikativní inverzí b *modulo m nebo* multiplikativní modulární inverzí *a píšeme*

$$c = |b^{-1}|_m = b^{-1} \bmod m.$$

V případě, že m není prvočíslo, $(\mathcal{Z}_m, +, \cdot)$ není tělesem, a nenulové prvky mohou mít, ale nemusí mít MMI. Pro existenci MMI platí následující věta a její důsledek.

Věta 2.1 *Nechť $b \in \mathcal{Z}_m$. Potom existuje nějaké jedinečné celé číslo $c \in \mathcal{Z}_m$, které odpovídá rovnici (1) tehdy a jen tehdy pokud $|b|_m \neq 0$ and $\gcd(b, m) = 1$.*

Důsledek 2.1 *Když $b \in \mathcal{Z}_m$ je nenulové, potom existuje jedinečné $|b^{-1}|_m \in \mathcal{Z}_m$ tehdy a jen tehdy když b a m jsou nesoudělná.*

Znamená to, že $(\mathcal{Z}_m, +, \cdot)$ je vždy konečný komutativní okruh a když m je prvočíslo, je také konečným tělesem isomorfním ke Galoisovmu tělesu GF(m). Když $|b^{-1}|_m$ existuje, definujeme dělení modulo m následovně:

Definice 2.2

$$\left|\frac{a}{b}\right|_m = |a|b^{-1}|_m|_m.$$

Před uvedením výpočtu MMI s použitím EEA uvedeme definici *největšího společného dělitele* (*Greatest Common Divisor — GCD*) dvou celých čísel (byl již použit ve větě 2.1).

Definice 2.3 *GCD dvou celých čísel a a b (obou nenulových) je největší kladné celé číslo d, které dělí $|a|$ a $|b|$ a označujeme $\gcd(a, b) = d$.*

GCD splňuje následující podmínky:

1. $\gcd(0,\, b) = |b|$ pro $b \neq 0$,
2. $\gcd(a,\, b) = \gcd(b,\, a) = \gcd(|a|,\, |b|) = \gcd(a,\, b + ka)$, kde $k \in \mathcal{Z}$.

2.3 Euklidův algoritmus

Uvažujme Euklidův algoritmus — EA pro nalezení GCD dvou různých celých čísel. Podle definice a podmínek pro GCD bez újmy na obecnosti předpokládáme, že a a b jsou nezáporná celá čísla, ne však obě nulová. Základní metodou pro popsání EA dvou celých čísel $a > b > 0$ je využití *vlastností dělení*. Pokud a dělíme b, potom existují celá čísla q a r, kde $q > 0$ a $0 \leq r < b$ taková, že $a = bq + r$, kde q je kvocient a r je zbytek. Opakovaným použitím tohoto předpisu získáme systém rovnic a nerovnic:

$$\begin{array}{lcll} a & = & b\,q_1 + r_1 & 0 < r_1 < b, \\ b & = & r_1\,q_2 + r_2 & 0 < r_2 < r_1, \\ r_1 & = & r_2\,q_3 + r_3 & 0 < r_3 < r_2, \\ & & \vdots & \vdots \\ r_{n-2} & = & r_{n-1}\,q_n + r_n & 0 < r_n < r_{n-1}, \\ r_{n-1} & = & r_n\,q_{n+1}. & \end{array} \tag{2}$$

kde $r_n \neq 0$, ale $r_{n+1} = 0$. Sekvenční vykonávání (2) se nazývá *Euklidův algoritmus — EA*. Nejmenší nenulový zbytek r_n v (2) má následující vlastnosti:

Věta 2.2 $\gcd(a,\, b) = \gcd(r_1,\, r_2) = \cdots = \gcd(r_{n-1},\, r_n) = \gcd(r_n,\, 0) = r_n$.

EA lze vyjádřit v jednoduché a pro výpočet užitečné formě:

ALGORITMUS 1 — EUKLIDŮV ALGORITMUS

Input: $a,\, b \in \mathcal{Z}$ and $a > b > 0$
Output: $\gcd(a,\, b)$

1. **while** $(b > 0)$
2. $\quad r := |a|_b$
3. $\quad a := b$
4. $\quad b := r$
5. **return** $\gcd(a, b) := a$

2.3.1 Časová složitost Euklidova algoritmu

Počet dělících kroků EA byl tématem mnoha výzkumných prací. Lameho věta [30] dává odhad počtu dělení potřebných pro nalezení GCD pomocí EA. Pochopení a důkaz této věty je založen na skutečnosti, že dva po sobě jdoucí členy Fibonacciho posloupnosti mají maximální počet dělení [45], protože podíl ve všech krocích kromě posledního je 1 (vyplývá z definice Fibonacciho čísel).

Věta 2.3 (Lameho) *Počet dělení n potřebných k nalezení GCD dvou celých čísel $a > b > 0$ použitím EA je menší než pětinásobek počtu dekadických číslic b, tj. $n \leq 5k$, kde $k = \lfloor \log_{10} b \rfloor$.*

Důsledek 2.2 *GCD dvou kladných celých čísel $a > b > 0$ může být nalezen s použitím $O((\log_2 a)^3)$ bitových operací. Protože z Lameho věty 2.3 lze odvodit, že $O((\log_2 a))$ dělení je potřebných pro nalezení $\gcd(a, b)$ pomocí EA a tak každé dělení je vykonáno pomocí $O((\log_2 a)^2)$ bitových operací, je*

$\gcd(a,\, b)$ *vypočítáno s maximálně* $O((\log_2 a)^3)$ *bitovými operacemi. Příklad efektivní implementace GCD algoritmu je v [43].*

2.3.2 Rozšířený Euklidův algoritmus

Následující věta uvádí důležitou vlastnost GCD:

Věta 2.4 *Když* $a > b > 0$ *a* $a, b \in \mathcal{Z}$*, potom* $\gcd(a,\, b)$ *je nejmenší kladné číslo* d *takové, že*

$$d = a\,x + b\,y, \quad \textit{kde } x,\, y \in \mathcal{Z}.$$

Je zřejmé, že celá čísla x a y nejsou jediná, protože, pro nejaké $t \in \mathcal{Z}$, $d = a\,(x + b\,t) + b\,(y - a\,t)$. Také ne každá lineární kombinace a a b vyhovuje rovnici pro dané d, protože když $d = a\,x + b\,y$ potom $(k\,d) = a\,(k\,x) + b\,(k\,y)$, pro nějaké $k \in \mathcal{Z}$.

Můžeme najít celá čísla x a y, která splňují rovnici $d = a\,x + b\,y$ s použitím (2) a získáme postupně zbytky $r_1,\, r_2,\, \ldots,\, r_{n+1}$ jako funkce a a b.

$$\begin{array}{rcll}
r_1 &=& a + b\,(-q_1) & 0 < r_1 < b \qquad q_1 = \lfloor a/b \rfloor, \\
r_2 &=& b + r_1\,(-q_2) & 0 < r_2 < r_1 \qquad q_2 = \lfloor b/r_1 \rfloor, \\
&=& a\,(-q_2) + b\,(1 + q_1\,q_2) & \\
r_3 &=& r_1 + r_2(-q_3) & 0 < r_3 < r_2 \qquad q_3 = \lfloor r_1/r_2 \rfloor, \\
&=& a(1 + q_2\,q_3) + b(-q_1 - q_3 - q_1\,q_2\,q_3) & \\
&\vdots & & \\
r_n &=& r_{n-2} + r_{n-1}\,(-q_n) & 0 < r_n < r_{n-1} \quad q_n = \lfloor r_{n-2}/r_{n-1} \rfloor, \\
&=& a\,x + b\,y & \\
0 &=& r_{n-1} + r_n\,(-q_{n+1}) & \qquad\qquad\quad q_{n+1} = \lfloor r_{n-1}/r_n \rfloor \\
&=& a\,u + b\,v\,. &
\end{array} \tag{3}$$

Výpočet (3) můžeme také vyjádřit ve formě tabulky 1. Z tabulky je vidět ve druhém sloupci posloup-

Tabulka 1: Výpočet r_n, x, a y

	a	1	0
	b	0	1
q_1	r_1	1	$-q_1$
q_2	r_2	$-q_2$	$1 + q_1\,q_2$
q_3	r_3	$1 + q_2\,q_3$	$-q_1 - q_3 - q_1\,q_2\,q_3$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
q_n	r_n	x	y
q_{n+1}	0	u	v.

nost generovaných zbytků (2), v prvním sloupci jsou koeficienty $q_1,\, q_2,\, \ldots,\, q_{n+1}$, ve třetím sloupci

jsou celá čísla, která násobí a, a ve čtvrtém sloupci jsou celá čísla, která násobí b. EA algoritmus prováděný v této podobě je takzvaný *Rozšířený EA (Extended Euclidean Algorithm — EEA)*. EEA lze zapsat v následujícím pseudokódu:

ALGORITMUS 2 — ROZŠÍŘENÝ EUKLIDŮV ALGORITMUS

Input: $a,\ b \in \mathcal{Z}$ a $a > b > 0$
Output: $\gcd(a,\ b)$ a $x,\ y \in \mathcal{Z}$ takové, pro která platí $\gcd(a,\ b) = x\,a + y\,b$

1. $f_1 := 1,\ f_2 := 0,\ g_1 := 0,\ g_2 := 1,\ r_1 := a,\ r_2 := b$
2. **while** $(r_2 > 0)$
3. $\quad q := \lfloor r_1 / r_2 \rfloor$
4. $\quad f := f_2,\ g := g_2,\ r := r_2$
5. $\quad r_2 := r_1 - q\,r_2,\ f_2 := f_1 - q\,f_2,\ g_2 := g_1 - q\,g_2$
6. $\quad f_1 := f,\ g_1 := g,\ r_1 := r$
7. **return** $\gcd(a,\ b) := r$**,** $x := f$**,** $y := g$

Použitím věty 2.3 a důsledku 2.2 je možné snadno ukázat, že počet dělení v EEA je stejný jako počet dělení v EA, tj. $O(\log_2 a)$, a výpočet GCD podle EEA vyžaduje $O(\log_2 a)^3$ bitových operací.

2.3.3 Euklidův algoritmus pro výpočet multiplikativní modulární inverze

Pro výpočet MMI modulo m nějakého celého čísla b na konečném komutativním okruhu $(\mathcal{Z}_m,\ +,\ \cdot\,)$ nebo na konečném tělese $(\mathcal{Z}_p,\ +,\ \cdot\,)$ (pokud $m = p$ a p je prvočíslo), je možné použít výpočet, který je prováděn v (3) a tabulce 1. Z důsledku 2.1 $|b^{-1}|_m$ existuje pro $b \neq 0$ pokud $\gcd(m,\ b) = 1$.

Věta 2.5 *Když* $\gcd(m,\ b) = 1$ *a když* $1 = m\,x + b\,y$*, potom* $|b^{-1}|_m = |y|_m$.

Je tomu tak, protože $1 = |m\,x + b\,y|_m = |b\,y|_m = |b\,|y|_m|_m$ a z definice plyne $|y|_m = |b^{-1}|_m$.

ALGORITMUS 2 může být také použit pro výpočet podílu $|c/b|_m$ modulo m, pokud inicializační hodnota $g_2 := c$. Potom je výsledek $|c\,b^{-1}|_m = |y|_m$. Modulární redukce každého mezivýsledku g_2 (řádek 5) může být vykonávána v každém iteračním kroku, tj. $g_2 := |g_1 - q\,g_2|_m$. Časová složitost takového algoritmu je stejná jako složitost ALGORITMU 2.

2.3.4 Binární Euklidův algoritmus

První modifikovaný EA určený pro použití v binární aritmetice byl uveden [49]. Tento algoritmus vykonával výpočty bez použití operace dělení a byl založen jen na operacích sčítání/odečítaní, testování parity a dělení sudých čísel, což odpovídalo posuvu čísel doprava v binárním vyjádření. Binární algoritmy jsou založeny na několika následujících pravidlech týkajících se dvou celých čísel a a b založených na definičních pravidlech GCD – viz podkapitolu 2.3.

1. Pro sudá a, b je $\gcd(a,b) = 2\gcd(a/2, b/2)$, neboť $k\gcd(a,b) = \gcd(k\,a, k\,b)$ pro $k \in \mathcal{Z}, k \geq 0$.
2. Když a je sudé a b je liché, potom $\gcd(a,\ b) = \gcd(a/2,\ b)$.

3. Když a a b jsou kladná a lichá, potom $a - b$ je sudé, $|a - b| < \max(a, b)$, a $\gcd(a, b) = \gcd(|a - b|/2, b)$.

4. $\gcd(a, 0) = |a|$.

Pokud respektuje tato pravidla, potom binární EA algoritmus pro výpočet GCD může vyjádřit následujícím pseudokódem.

ALGORITHM 3 — BINÁRNÍ EUKLIDŮV ALGORITMUS

Input: $a, b \in \mathcal{Z}$ a $a > b > 0$
Output: $\gcd(a, b)$

1. $k := 0$
2. **while** (a and b even)
3. $\quad a := a/2, b := b/2, k := k + 1$
4. **while** $(a > 0)$
5. $\quad$ **if** (a even) **then**
6. $\quad\quad a := a/2$
7. $\quad$ **else if** (b even) **then**
8. $\quad\quad b := b/2$
9. $\quad$ **else**
10. $\quad\quad x := (a - b)$
11. $\quad\quad$ **if** $(x \geq 0)$ **then**
12. $\quad\quad\quad a := x$
13. $\quad\quad$ **else**
14. $\quad\quad\quad b := -x$
15. **return** $\gcd(a, b) := 2^k\, b$

2.3.5 Časová složitost binárního Euklidova algoritmu

Přesné určení chování ALGORITMU 3 se jeví jako obtížný problém. Existuje mnoho studií věnovaných tomuto tématu. Přehled různých přístupů je představen v [26]. Hlavní pozornost je věnována počtů dělení, tj. počtů pravých posunů (řádek 6. a 8.) a počtu odčítání (řádek 10) v iterační smyčce ALGORITMU 3. V [26] je popsána studie o časové složitosti ALGORITMU 3 pomocí vhodného modelu. V případě, že $a, b \in \mathcal{Z}$, které jsou nezávislé a rovnoměrně rozloženy v rozmezí $1 \leq a, b < 2^n$, kde n je počet bitů slova, se průměrné hodnoty počtů pravých posunů H a počtů odečítání S dají přibližně vyjádřit:

$$S \approx 0.7n + O(1), \qquad H \approx 1.41n + O(1)\,. \tag{4}$$

Podle výsledků dalších testů (miliony náhodných vstupů pro $N \leq 30$) můžeme psát:

$$S \approx 0.7n + 0.5, \qquad H \approx 1.41n + 2.7\,. \tag{5}$$

Teoretická analýza ALGORITTMU 3 in [3] predikuje, že S a H jsou asymptoticky rovny $2N/\beta$ and $4n/\beta$ za předpokladu, že $1 \leq a,\, b < 2^n$, kde $2/\beta \approx 0.70597$ [3]. Za předpokladu, že odečítání trvají déle než posuny, a posuny mohou být prováděny současně se zápisem výsledků operací odčítání, můžeme považovat za „nejhorší“ situaci, takovou kdy vstupní hodnoty a a b vyžadují nejvíce odčítání. Podle [26] je maximální počet je odčítání přesně $\max(\lfloor \log_2 a \rfloor, \lfloor \log_2 b \rfloor) + 1$.

2.3.6 Rozšířený binární Euklidův algoritmus

Rozšířený binární Euklidův algoritmus je odvozen s respektováním pravidel pro algoritmus EEA uvedených v podkapitole 2.3.2 a pravidel pro binární podobu EA uvedených v podkapitole 2.3.4. Tento algoritmus si můžeme vyjádřit pomocí následujícího pseudokódu.

ALGORITMUS 4 — ROZŠÍŘENÝ BINÁRNÍ EUKLIDŮV ALGORITMUS

Input: $a,\, b \in \mathcal{Z}$ a $a > b > 0$
Output: $\gcd(a,\, b) = x\, a + y\, b$ a $x,\, y \in \mathcal{Z}$

1. $k := 0, f_1 := 1, f_2 := 0, g_1 := 0, g_2 := 1$
2. **while** (a even $\wedge$ b even)
3. $\quad a := a/2, b := b/2, k := k + 1$
4. **while** $(a > 0)$
5. $\quad$ **if** (a even) **then**
6. $\quad\quad a := a/2$
7. $\quad\quad$ **if** (f_1 even $\wedge$ g_1 even) **then**
8. $\quad\quad\quad f_1 := f_1/2,\, g_1 := g_1/2$
9. $\quad\quad$ **else**
10. $\quad\quad\quad f_1 := (f_1 + b)/2, g_1 := (g_1 - a)/2$
11. $\quad$ **else if** (b even) **then**
12. $\quad\quad b := b/2$
13. $\quad\quad$ **if** (f_2 even $\wedge$ g_2 even) **then**
14. $\quad\quad\quad f_2 := f_2/2,\, g_2 := g_2/2$
15. $\quad\quad$ **else**
16. $\quad\quad\quad f_2 := (f_2 + b)/2,\, g_2 := (g_2 - a)/2$
17. $\quad$ **else**
18. $\quad\quad c := (a - b),\, d := f_1 - f_2,\, e := g_1 - g_2$
19. $\quad\quad$ **if** ($c \geq 0$) **then**
20. $\quad\quad\quad a := c,\, f_1 := d,\, g_1 := e$
21. $\quad\quad$ **else**
22. $\quad\quad\quad b := -c,\, f_2 := -d,\, g_2 := -e$
23. **return** $\gcd(a,\, b) := 2^k\, b$ and $x := f_2,\, y := g_2$

Podrobný popis tohoto algoritmu je v [33]. Pokud nejmenší nenulový zbytek $r_n = 1$ a $k = 0$, pak $\gcd(a',\, b') = 1$ and $y = |b'^{-1}|_{a'}$. Je zřejmé, že výpočet MMI může být vykonán bez průběžného

výpočtu koeficientu f_i. Takové algoritmy jsou uvedeny v [26, 31, 32]. Tyto algoritmy řeší problém výskytu záporných hodnot koeficientu g_i převedením na kladné číslo přičítáním a ke g_i.

Výpočet MMI $|b^{-1}|_a$ podle ALGORITMU 4 může být vykonán i pokud inverze existuje. Znamená to, že $\gcd(a,\, b) = 1$ pro a a b komutativního okruhu $(\mathcal{Z}_a,\, +,\, \cdot\,)$ nebo, že a a b představují prvky konečného tělesa $(\mathcal{Z}_p,\, +,\, \cdot\,)$ pokud $a = p$ a p je prvočíslo.

3 Efektivní binární Euklidovy algoritmy

V této kapitole jsou uvedeny princip a implementace dvou efektivních binárních algoritmů pro výpočet MMI založených na EA. Prvním algoritmem je takzvaný „Left-Shift — LS“ binární algoritmus a druhým je takzvaný „Subtraction-free AMI — SF“ algoritmus, který je odvozen z algoritmu pro výpočet MMI v takzvané Montgomeryho bázi.

3.1 „Left-Shift“ binární Euklidův algoritmus

Tato kapitola se zabývá novým binárním LS algoritmem pro MMI. Tento algoritmus byl publikován v [32], a dále jeho principy byly patentovány v tuzemsku [36] a v USA [38]. Výpočet MMI pomocí dalších algoritmů, jako jsou algoritmy [25, 26], které byly popsány v kapitole 3.1, mají větší časovou složitost a v konečném důsledku také větší hodnotu implementačního parametru T×A než LS algoritmus. Základní principiální rozdíl ve výpočtu MMI v prezentovaném algoritmu oproti jiným doposud známým algoritmům je v průběžném (v každém iteračním kroku algoritmu) násobení mocninou 2 dílčích hodnot binárního EEA. Znamená to, že obsah jednotlivých registrů, který se má násobit, se posouvá doleva. Na základě toho je také pojmenován *Left-Shift algoritmem — LS algoritmem*. Na druhou stranu, jiné algoritmy pro výpočet MMI, které posouvají v každé iteraci dílčí výsledky doprava, jsou takzvané *Right-Shift algoritmy — RS algoritmy*.

3.1.1 LS algoritmus

Nový přístup k výpočtu MMI se vyhýbá nevýhodám algoritmů v [25, 26]. Tyto nevýhody představují hlavně vysoký počet odčítání a testů, po kterých následuje korekce v případě záporných hodnot výsledků a také operace redukce záporných a lichých čísel. V případě výpočtu MMI pomocí algoritmu v [25] je nutné provést tzv. odložené dělení ve druhé fázi algoritmu. LS algoritmus, který tyto nevýhody odstraňuje, je uveden dále v pseudokódu jako ALGORITMUS 5.

ALGORITMUS 5 — LEFT-SHIFT ALGORITMUS

Input: $a \in [1,\, p-1]$ and p
Output: $r \in [1,\, p-1]$, kde $r = a^{-1} \pmod p$, c_u, c_v
and $0 < c\_v + c\_u \leq 2n$

1. $u := p, v := a, r := 0, s := 1$
2. $c\_u = 0$, $c\_v = 0$
3. while($u \neq \pm 2^{c\_u}$ & $v \neq \pm 2^{c\_v}$)
4. if (u_n,u_{n-1}= 0) *or* (u_n, $u_{n-1} = 1$ & OR(u_{n-2}, …, u_0) = 1) then
5. if ($c\_u \geq c\_v$) then
6. $u := 2u$, $r := 2r$, $c\_u := c\_u + 1$
7. else
8. $u := 2u$, $s := s/2$, $c\_u := c\_u + 1$
9. else if (v_n, $v_{n-1} = 0$) *or* (v_n, $v_{n-1} = 1$ & OR(v_{n-2}, …, v_0) = 1) then
10. if ($c\_v \geq c\_u$) then

11. $v := 2v$, $s := 2s$, $c\_v := c\_v + 1$
12. else
13. $v := 2v$, $r := r/2$, $c\_v := c\_v + 1$
14. else
15. if ($v_n = u_n$) then
16. *oper* = „−“
17. else
18. *oper* = „+“
19. if ($c\_u \leq c\_v$) then
20. $u := u$ *oper* v, $r := r$ *oper* s
21. else
22. $v := v$ *oper* u, $s := s$ *oper* r
23. if ($v = \pm 2^{c}–^{v}$) then
24. $r := s$, $u_n := v_n$
25. if ($u_n = 1$) then
26. if ($r < 0$) then
27. $r := -r$
28. else
29. $r := p - r$
30. if ($r < 0$) then
31. $r := r + p$
32. return r, c_u, and c_v.

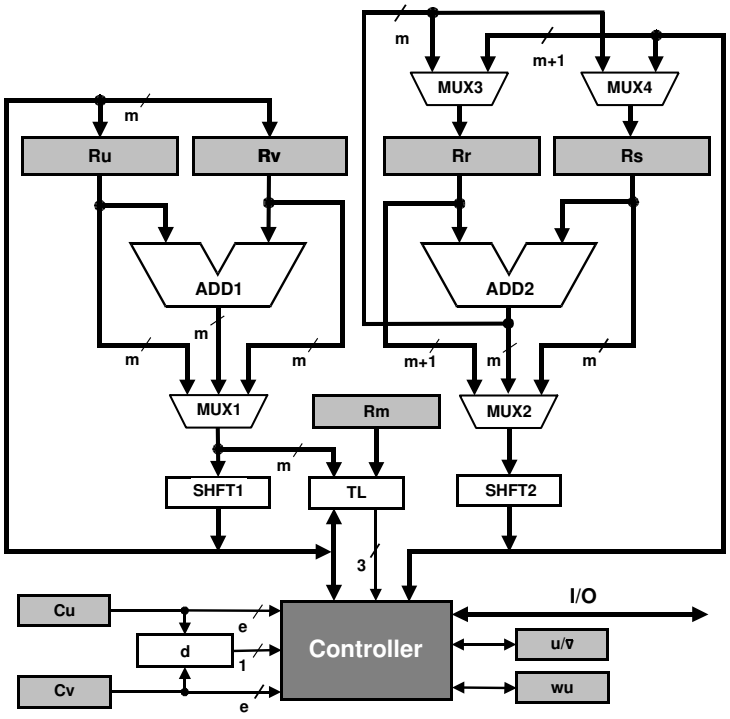
Z uvedeného je zřejmé, že zatímco jiné přístupy musí odstraňovat záporné nebo liché hodnoty mezivýsledků během výpočtu MMI, ALGORITMUS 5 se tomu vyhýbá. RS binární algoritmy pro výpočet MMI zahrnuji minimálně jednu operaci sčítání v každé iteraci pro eliminaci výskytů záporných a lichých mezivýsledků v průběhu počítání MMI. ALGORITMUS 5 eliminuje redundantní operace sčítání/odčítání, tj. je lepší než ostatní algoritmy vzhledem k menšímu počtu operací sčítání/odčítání během výpočtu MMI. Tato skutečnost má velmi významný vliv na časovou složitost výpočtu MMI v případě velkých čísel, které se často vyskytují v kryptografických primitivech. Budeme-li předpokládat, že časová složitost operací posunu zůstává konstantní, ale časová složitost operací sčítání/odčítání se zvyšuje z důvodu časového zpoždění přenosového řetězce sčítačky/odčítačky přibližně $\lceil \log_2 n \rceil$krát (n je počet bitů slova), pak celková doba složitosti výpočtu MMI, pro velká čísla, silně závisí na počtu operací sčítání/odčítání ve výpočetním procesu. V důsledku tohoto, teoreticky ze statistických analýz chování, se u ALGORITMU 5 při výpočtu MMI pro velká počítačová slova (více než 256 bitů) ukazuje, že tento algoritmus je alespoň dvakrát rychlejší než jiné RS binární algoritmy pro výpočet MMI.

Další důležitou vlastnosti ALGORITMU 5, který má bezprostřední vliv na časovou složitost výpočtu MMI, je eliminace tzv. „opravných“ odečítání záporných výsledků, které se vyskytují v iterační smyčce každého RS algoritmu. V některých hardwarových implementacích RS binárních algoritmů se toto řeší pro urychlení výpočtu přidáním dalších odčítaček. S další odčítačkou můžeme provést reciproční paralelní operaci k operaci odečítání reprezentující test, a dále pro výpočet použít kladný

výsledek jedné ze dvou odčítaček. Taková řešení ale zhoršují hodnotu prostorové složitosti u hardwarových implementací. Pokud nechceme zhoršovat prostorovou složitost hardwarové implementace, musíme v případě negativního výsledku vykonat operaci odečítání navíc. V hardwarové implementaci ALGORITMU 5 jsou tyto problémy odstraněny zavedením reprezentace záporných čísel pomocí doplňkového kódu. Znamená to, že záporné hodnoty v průběhu výpočtu jsou „povoleny“ a řízení eliminačních datových toků u algoritmů je vyřešeno sofistikovanějším způsobem při respektování hodnot přenosových bitů u sčítání/odčítání.

3.1.2 Hardwarová architektura LS algoritmu

Protože ALGORITMUS 5 je binárním algoritmem, je jeho implementace v hardwaru téměř přímočará. ALGORITMUS 5 může být převeden do následující hardwarové architektury:



Obrázek 1: Hardwarova architektura pro ALGORITMUS 5

Pro průběžné uchovávání mezivýsledků slouží 4 hlavní registry R_u, R_v, R_r a R_s, výpočtům sčítáním/odčítáním jsou věnovány sčítačky ADD1 a ADD2 a posuvy — násobení a dělení 2 jsou vykonávány pomocí propojení SHIFT1 a SHIFT2. Vzájemně jsou tyto jednotky propojeny datovými sběrnicemi a multiplexery (MUXx) respektující datové toky v průběhu výpočtu. Celý výpočet řídí řídící jednotka (Controller) za účastí pomocných registrů (R_m, C_u, C_v, u/$\bar{v}$ a vu) a pomocné logiky (TL a d). Podrobný popis tohoto zapojení je v [32, 38, 36].

Implementace ALGORITMU 5 odpovídající hardwarové architektuře na obrázku 1 má průměrnou časovou složitost operací výpočtu pro MMI uvedenou v tabulce 2, kde n je počet bitů počítačového

slova. Hodnoty počtu operací posuvu a operací sčítání/odčítání odpovídá studiím uvedených v části 2.3.5 a hodnotám koeficientů v rovnicích (4) a (5). Z tabulky lze snadno odvodit, že v případě stejné

Průměrný počet operací	ALGORITMUS 5	ALGORITMUS 4
Sčítání/odčítání	$0.7n$	$0.7n$
Posuv	$1.4n$	$1.4n$
Test	0	$0.35n$
Korekce záporných hodnost	0	$0.35n$

Tabulka 2: Průměrné časové složitosti operací pro ALGORITMUS 5 a ALGORITMUS 4

doby pro všechny operace může být ALGORITMUS 5 jen o cca 25% rychlejší než ALGORITMUS 4. V případě, že se operace posuvu provádí současně se zápisem výsledné hodnoty po sčítání/odčítání, je zrychlení cca 33%. Jak již bylo uvedeno v této části, provádění operací sčítání/odečítání nemusí mít stejnou časovou složitost jako operace posuvů. Proto, pokud jsme schopni vykonávat operace posuvu rychleji než operace sčítání/odečítání, může být ALGORITMUS 5 až $2\times$ rychlejší než ALGORITMUS 4, a to v závislosti na délce sčítačky/odčítačky, tj. na délce kritické cesty, která je v tomto případě tvořena přenosovým řetězcem.

3.2 „Subtraction-free AMI“ algoritmus

V této části bude popsán takzvaný „Subtraction-free AMI — SF-AMI“ algoritmus, který je odvozen z první fáze algoritmu pro výpočet MMI v Montgomeryho bázi. Reprezentace zbytkových tříd v takzvané Montgomeryho bázi (Montgomery Domain — MD) bylo uvedeno v [41]. Hlavním důvodem použití MD je zrychlení modulárního násobení. To je zvláště důležité v kryptografických primitivech s dlouhými počítačovými slovy a velkým počtem násobení (např. RSA), kdy urychlení násobení značně urychlí celý výpočet. Před samotným uvedením algoritmu SF-AMI pro výpočet MMI v MD stručně popíšeme matematický základ MD pro binární násobení a inverzi.

3.2.1 Montgomeryho binární násobení

Nejdříve definujeme velmi jednoduše takzvané m-residuum:

Definice 3.1 *Číslo $\bar{a} = |a\,R|_m$ je takzvané m-*residuum čísla *a a množina čísel $\bar{a}$ je množinou čísel takzvaného Motgomeryho oboru čísel (Montgomery domain — MD), kde číslo R je nejmenší mocnina základu poziční číselné soustavy, pro které platí $R > m$.*

Definice 3.2 *Montgomeryho součin dvou m-residuí $\bar{a}$ a $\bar{b}$ je definován jako m-residuum*

$$\bar{c} = |\bar{a}\,\bar{b}\,R^{-1}|_m.$$

Pokud $c = |a\,b|_m$, lze ukázat, že:

$$\bar{c} = |c\,R|_m = |a\,b\,R|_m = |a\,R\,b\,R\,R^{-1}|_m = |\bar{a}\,\bar{b}\,R^{-1}|_m$$

Montgomeryho násobení, které provádí operaci $\bar{c} = |\bar{a}\,\bar{b}\,R^{-1}|_m$, můžeme použít pro výpočet konverze $\bar{c}$ na c, tedy inverzní mapování m-reziduí.

Věta 3.1 *Nechť* $\bar{a} \in [1,\, m-1]$ *je* m*-residuum,* $\bar{b} = 1$*. Potom podle definice 3.2 platí:*

$$a = |\bar{a} \cdot 1 \cdot R^{-1}|_m = |a\,R\,R^{-1}|_m = a.$$

Výpočet m-reziduí $\bar{a}$ z a můžeme provést znovu pomocí Montgomeryho násobení, pokud máme předpočítánu hodnotu $|R^2|_m$. Pak můžeme vyjádřit konverzi m-residua čísla $\bar{a}$ na a takto:

Věta 3.2 *Nechť* $a \in [1,\, m-1]$ *je celé číslo a* $b = |R^2|_m$*. Potom, z definice 3.2 plyne:*

$$\bar{a} = |a\,b\,R^{-1}|_m = |a\,R^2\,R^{-1}|_m = |a\,R|_m.$$

Z předchozího je vidět, že pomocí Montgomeryho násobení můžeme efektivně vykonávat operace konverze a samotné násobení v MD. Algoritmus Montgomeryho násobení je vhodný zejména pro hardware, ale také pro software [29]. Hardwarová implementace může používat různé obměny překrývání násobení a redukce[28, 41].

MD v hardware má R mocninu dvou, potom v binární reprezentaci je operace modulo $R = 2^n$, kde n je počet bitů slova, prováděná efektivně. Předpokládejme, že $R = 2^n$ a nechť $\bar{a} = (\bar{a}_{n-1}\,\bar{a}_{n-2}\,\cdots\,\bar{a}_0)_2$, kde $\bar{a}_i$ je 0 or 1 a $\bar{a} = \sum_{i=0}^{n-1} \bar{a}_i\,2^i$. Nechť $0 \leq \bar{b} < m$. Nyní můžeme popsat základní binární Montgomeryho algoritmus pro násobení pomocí pseudokódu ALGORITMEM 6:

ALGORITMUS 6 — BINÁRNÍ MONTGOMERYHO ALGORITMUS PRO NÁSOBENÍ

Input: m-residua $\bar{a}$, $\bar{b}$ and m, n
Output: $|\bar{a}\,\bar{b}\,R^{-1}|_m$

1. $s := 0, i := 0$
2. **while** $(i < n)$
3. $\quad x := x + \bar{a}_i\,\bar{b}$
4. $\quad x := (x + x_0 m)/2$
5. $\quad i := i + 1$
6. **if** $(x \geq m)$ **then**
7. $\quad x := x - m$
8. **return** $|\bar{a}\,\bar{b}\,R^{-1}|_m := x$

Z algoritmu je patrné, že posuv mezivýsledku je prováděn doprava (na rozdíl od násobicích algoritmů v celočíselné doméně) a redukce modulo m se vykonávána podle testování nejméně významného bitu mezivýsledku x v řádku 5. Následně je výsledek posunut doprava (dělení 2). Toto nabízí navrhovat velmi výkonné násobičky založené na proudovém zpracování nebo násobičky používající tzv. „carry-save“ sčítačky, a to hlavně proto, že je odstraněno čekání na přenos při výpočtu, který řídí provádění redukce modulo m.

Existuje mnoho aplikací Montgomeryho násobení, a to hlavně v kryptografii [1, 2, 5]. Různé architektury jsou realizovány v technologiích ASIC a FPGA za účelem navržení efektivních kryptografických systémů [9, 10, 39, 48, 50].

3.2.2 Montgomeryho multiplikativní modulární inverze

Část 2.3.4 popisuje RS binární EA ALGORITMUS 3, který počítá GCD dvou celých čísel a a b s průběžným půlením, tj. posunem vpravo a odečtením. Související ALGORITMUS 4 v části 3.1 počítá MMI $|b^{-1}|_a$. V ALGORITMU 4 jsou jak liché, tak sudé hodnoty půleny. Půlení lichých hodnot f_i nebo g_i je prováděno tak, že se nejdříve přičte hodnota b k f_i, a pak odečte hodnota a od g_i. Výpočet MMI může být proveden bez výpočtu koeficientu f_i. Takové algoritmy jsou v [26, 31, 32]. V těchto algoritmech je půlení liché hodnoty g_i znovu provedeno tak, že se nejprve přičte modul a ke g_i, kde a je z předpokladu liché. Výskyt záporných hodnot g_i je řešen konverzí do kladných čísel přičtením hodnoty a. Další možností je odložit půlení až na konec iterativního výpočtu GCD. V tomto případě algoritmus výpočtu MMI nejprve vypočítá $|b^{-1}\,2^k|_a$, kde k je počet odložených půlení, pak je provedeno půlení k krát modulo a. Jeden z výsledků takového výpočtu je MMI v MD, tj. $|b^{-1}\,2^n|_a = |\overline{b}^{-1}|_a$, kde n je počet bitů a a $\overline{b} = |b\,2^n|_m$ je m-residuum.

Na základě tohoto pozorování lze rozdělit výpočet MMI do dvou částí. V první fázi se počítá tzv. „téměř Montgomeryho inverze — Almost Montgomery Inverse — AMI“. Výsledkem této fáze je GCD a a b a za předpokladu, že $\gcd(a,\,b) = 1$, mezihodnota $|b^{-1}\,2^k|_a$, kde k je počet iterací. Druhá fáze půlí mezihodnotu hodnotu $k - n$ krát modulo a, a potom neguje výsledek. Je-li $\gcd(a,\,b) = 1$, je konečný výsledek Montgomeryho MMI $|b^{-1}\,2^n|_a = |\overline{b}^{-1}|_a$. Takový algoritmus můžeme vyjádřit pseudokódem:

ALGORITMUS 7 — MONTGOMERYHO MULTIPLIKATIVNÍ MODULÁRNÍ INVERZE

Input: $a, b \in \mathcal{Z}$, $a > b > 0$, a je liché a n je počet bitů a
Output: $\gcd(a,\,b) > 1$ nebo $|b^{-1}\,2^n|_a = |\overline{b}^{-1}|_a$

První fáze – Almost Montgomeryho Inverze – AMI
1. $u := a, v := b, r := 0, s := 1, k := 0$
2. **while** $(v > 0)$
3. **if** (u even) **then**
4. $u := u/2, s := 2s$
5. **else if** (v even) **then**
6. $v := v/2, r := 2r$
7. **else if** $(u > v)$ **then**
8. $u := (u - v)/2, r := r + s, s := 2s$
9. **else**
10. $v := (v - u)/2, s := r + s, r := 2r$
11. $k := k + 1$
12. **if** $u \neq 1$ **then return** "Not relatively prime"
13. **if** $r \geq a$ **then**
14. $r := r - a$

Druhá fáze
15. **while** $(k > n)$

16. **if** r even **then**
17. $r := r/2$
18. **else**
19. $r := (r + a)/2$
20. $k := k - 1$
21. **return** $|b^{-1}\, 2^n|_a := a - r$

První fáze tohoto algoritmu je prováděna bez půlení, a také bez konverze záporných hodnot r a s do kladných hodnot. To je založeno na skutečnosti, že r a s jsou přepsány pouze kladnou hodnotou $r+s$. To je možné díky skutečnosti, že $r,\ s$ jsou sumou buď dvou záporných hodnot, anebo dvou kladných hodnot, a pak v iteračním procesu můžeme vždy vzít kladný výsledek jako absolutní hodnoty, které jsou zapsány do r, nebo do s (viz [33]).

3.2.3 Vlastnosti Montgomeryho algoritmu pro multiplikativní modulární inverzi

Následující věta odhaduje počet iterací první fáze, AMI, Montgomeryho algoritmu pro výpočet MMI.

Věta 3.3 *Když $a > b > 0$, $\gcd(a,\, b) = 1$ a a je liché, potom počet iterací první fáze* ALGORITMU 7 *je minimálně n a maximálně $2n$, kde n počet bitů a.*

Průměrná hodnota počtu iterací první fáze ALGORITMU 7 je $1.4n$. Průměrná hodnota posuvu je potom také $1.4n$ a průměrná hodnota sčítání 0.7 ve shodě s (4) a (5). Tyto hodnoty také souhlasí s hodnotami počtu sčítání/odčítání ALGORITMU 5, viz tabulka 2.

Nechť funkce $AMI(a,\, b) = |b^{-1}\, 2^k|_a$ představují první fázi ALGORITMU 7 — AMI a funkce $MMMI(a,\, b) = |b^{-1}\, 2^n|_a = |\overline{b}^{-1}|_a$ počítá celý ALGORITMUS 7, kde n je počet bitů v a a $\overline{b} = |b\, 2^n|_a$ je m-residuu. MMMI lze přímo použít pro výpočet inverze v MD s čísly z celočíselné báze. Totéž platí pro výpočet celočíselné inverze s MD operandy, protože platí:

$$|MMI(m,\, \overline{b})|_m = |b^{-1}\, 2^{-n}\, 2^n|_m = |b^{-1}|_m.$$

Pro výpočet inverze v MD s operandy z MD, vyžaduje druhá fáze ALGORITMU 7 modifikaci. Namísto dělení výsledku AMI 2^{k-n}, musí to být násobení hodnotou 2^{2n-k}, to je násobení $2n - k$ krát s 2 modulo m ($2n - k$ posuvů doprava modulo m).

$$|AMI(m,\, \overline{b}) \cdot 2^{2n-k}|_m = |(b^{-1}\, 2^{-n}\, 2^k)2^{2n-k}|_m = |\overline{b}^{-1}|_m$$

Tyto tři situace jsou zde uvedeny:

$$\begin{aligned} |\overline{b}^{-1}|_m &= |MMI(m,\, b)|_m \\ |b^{-1}|_m &= |MMI(m,\, \overline{b})|_m \\ |\overline{b}^{-1}|_m &= |AMI(m,\, b) \cdot 2^{2n-k}|_m. \end{aligned}$$

3.2.4 SF-AMI algoritmus

ALGORITMUS 7 — AMI počítající první fázi Montgomeryho MMI má nedostatky v podobě operace testování. Modifikovaná první část ALGORITMU 7 — AMI, takzvaný „Subtraction-free AMI — SF-AMI“, tento nedostatek odstraňuje. Pseudokód tohoto algoritmu je uveden v první části ALGORITMU 8.

ALGORITMUS 8 - SUBTRACTION-FREE MONTGOMERY MMI

Input: $a, b \in \mathcal{Z}$, $a > b > 0$, a je liché a n je počet bitů a
Output: $\gcd(a,\, b) > 1$ nebo $|b^{-1}\, 2^n|_a = |\overline{b}^{-1}|_a$

První fáze — Subtraction-Free AMI — SF-AMI
1. $u \leftarrow (-p)$, $v \leftarrow a$, $r \leftarrow 0$, $s \leftarrow 1$, $k \leftarrow 0$
2. **while** (1)
3. **if** ($u_{\mathrm{LSB}} == 0$) **then**
4. $u \leftarrow u/2, s \leftarrow 2s$
5. **else if** ($v_{\mathrm{LSB}} == 0$)
6. $v \leftarrow v/2, r \leftarrow 2r$
7. **else**
8. $x = u + v, y = r + s$
9. **if** ($x == 0$) **then** $r \leftarrow s$, **goto** 15.
10. **if** ($CARRY(x) == 0$) **then**
11. $u \leftarrow x/2, r \leftarrow y, s \leftarrow 2s$
12. **else**
13. $v \leftarrow x/2, s \leftarrow y, r \leftarrow 2r$
14. $k \leftarrow k + 1$

Druhá fáze
15. **while** ($k > n$)
16. **if** r even **then**
17. $r \leftarrow r/2$
18. **else**
19. $r \leftarrow (r + a)/2$
20. $k \leftarrow k - 1$
21. **return** $|b^{-1}\, 2^n|_a \leftarrow r$

Označení „==“ představuje srovnání rovnosti. Symboly „=“ a „←“ představují přiřazení hodnoty proměnné; ale v hardware „=“ znamená, že hodnota je přítomna na výstupu kombinačního obvodu, zatímco „←“ indikuje, že hodnota je zapsána do registru s příchodem další náběžné hrany hodinového signálu.

V některých publikovaných algoritmech [16, 17, 18, 19, 20, 21, 25, 47] může být AMI přepsána jako u ALGORITMU 8. V této verzi se algoritmus ukončí, jakmile se $u == v$ a inverze je získána z s

místo r. Ve srovnání s algoritmy [25, 47], proměnné u, r a s nejsou aktualizovány v poslední iteraci, hodnota k je menší o 1, a s je vždy menší než p a není potřebná následná operace redukce. Operace v řádcích 9–10 algoritmu v [47] jsou tak zbytečné a není potřeba další bit v r.

Odčítání $u-v$ a $CARRY(x)$ test v řádcích 8–10 představují $u > v$ test v [25, 47]. Jestliže $u > v$, pak výsledek je vydělen 2 a uložen do u (řádek 11). V opačném případě se vypočítá $v - u$, výsledek se vydělí 2 a je uložen do v (řádek 13). Pokud $u == v$ algoritmus skončí. V hardwaru může tento postup být proveden dvěma způsoby: První možností je použít dvě odčítačky, které provádějí $u-v$ a $v-u$ paralelně a vezme se jen kladný výsledek. Tento přístup se používá např. v [16, 17, 18, 19, 21]. Druhou možností je použít jednu odčítačku a v případě záporného výsledku uskutečnit negaci nebo výpočet $u - v$, což znamená další hodinový takt navíc. Pokud je výsledek záporný, je provedena korekce, která znamená znovu jeden hodinový takt navíc. Tato metoda se používá například v [20].

ALGORITMUS 8 je modifikací Kaliskiho algoritmu [25], který nevykonává korekce po „nezdařilých“ výpočtech. V hardwaru jsou proměnné x a y výstupy sčítaček. Další proměnné představují registry. Obsah u je vždy záporný po celou dobu výpočtu a je reprezentován v doplňkovém kódu. $CARRY(x)$ je výstup přenosu sčítačky, která počítá $x = u + v$; když $x < 0$, potom $|u| > v$, potom $CARRY(x) == 0$ a 1 v opačném případě.

Po dobu provádění AMI ALGORITMU 8 je $-p \leq u < 0$ a $0 \leq v,\ r,\ s < p$. Z toho důvodu není nutné uložit příslušné znaménkové bity (nikdy se nemění), a tedy n-bitové registry a datové sběrnice jsou dostatečné. $n$bitová sčítačka je také dostačující, při sčítání kladných v a záporných u je přenos z $n$tého řádu roven přenosu z $(n-1)$ního řádu a jeho doplněk představuje znaménko výsledku. Ve srovnání s jinými architekturami [16, 17, 21], které provádějí každou iteraci za jeden hodinový cyklus se dvěma odčítačkami pro výpočet $u - v$ respektive $v - u$, tak v případě AMI ALGORITMU 8 dvě odčítačky mohou být nahrazeny jedinou odčítačkou. Tímto způsobem se snižuje prostorová složitost.

Ve srovnání s architekturami, které potřebují hodinový takt navíc, vždy když $u < v$ [20], AMI ALGORITMUS 8 vykonává výpočet v méně hodinových cyklech. Průměrné zrychlení AMI ALGORITMU 8 je 1.25 [35]. V případě dlouhých slov, kdy sčítání a odčítání se provádí v několika hodinových cyklech, může průměrné zrychlení dosáhnout teoretické hodnoty 1.5. ALGORITMUS 8 má podobnou hardwarovou implementační architekturu jako LS algoritmus 1. Průměrná časová složitost operací výpočtu je uvedena v tabulce 3, kde n je počet bitů počítačového slova.

Průměrný počet operací	ALGORITMUS 8	ALGORITMUS 7
Sčítání/odčítání — AMI	$0.7n$	$0.7n$
Posuv — AMI	$1.4n$	$1.4n$
Test — AMI	0	$0.35n$
Sčítání — 2. fáze	$0.2n$ – $0.3n$	$0.2n$–$0.3n$
Posuv — 2. fáze	$0.4n$ – $0.6n$	$0.4n$–$0.6n$

Tabulka 3: Průměrné časové složitosti operací pro ALGORITMUS 8 a 7

3.3 Hardwarová implementace LS a SF-AMI algoritmů

V této části uvedeme příklady hardwarových implementací LS a SF-AMI algoritmů. Pro hardwarovou implementaci ALGORITMU 5 (LS algoritmus) byla zvolena technologie ASIC. Oproti FPGA technologii nabízí lépe využít vlastnosti LS algoritmu. Jedná se především o možnost lepšího přizpůsobení základním elementů ASIC pro konstrukci architektury z obrázku 1. Technologie FPGA jednak neumožňuje použití jemné granularity stavebních elementů, na rozdíl od technologie ASIC, a jednak dedikované přenosové cesty v FPGA strukturách znemožňují využití vlastností LS algoritmu, tj. využití malého počtu sčítání/odčítání pro návrh vícetaktových sčítaček/odčítaček. V tabulce 4 je srovnání časových složitosti implementaci LS algoritmu, ALGORITMU 5 a algoritmu pro Montgomeryho inverzi, ALGORITMU 7, v technologii ASIC 0.18μ pomocí nástroje pro syntézu Synopsys DC [8]. Návrh byl proveden pro různé délky bitů n počítačového slova. Z tabulky je vidět, že LS algoritmus je ≈ 1.3 krát rychlejší než ALGORITMUS 7. Je nutno podotknout, že plocha pro implementaci LS ALGORITMU 5 byla cca o 33% větší než plocha pro implementaci ALGORITMU 7.

n	**Montgomeryho MMI** [μs]	**LS MMI** [μs]	**Zrychlení**
128	0.76	0.56	1.36
160	0.98	0.75	1.31
192	1.17	0.88	1.33
256	1.66	1.20	1.38

Tabulka 4: Srovnání časové složitosti hardwarových implementaci v technologii ASIC 0.18μ

Dalším příkladem je hardwarová implementace první fáze algoritmu pro Montgomeryho inverze (ALGORITMU 7) — AMI a SF-AMI, 1. fáze ALGORITMUS 8, a to jednak v technologii FPGA, tak i ASIC [37]. Výsledky implementace pro FPGA jsou uvedeny v tabulce 5. Tyto výsledky jsou také v [6] . Implementace byla provedena ve VHDL, s použitím nástroje Xilinx ISE 6.3i pro FPGA Xilinx Virtex2.

n	**SF-AMI**	**AMI**	**Vylepšení** [%]
64	5.3	7.5	41
128	9.2	15.2	65
162	18	23.5	31
256	31.3	38.9	24

Tabulka 5: Srovnání implementačního parametru T×A [slices×μs] pro SF-AMI a AMI algoritmů v FPGA

Výsledky implementace pro ASIC jsou uvedeny v tabulce 6. Implementace byla provedena ve VHDL, s použitím nástroje Synplify ASIC, technologie TSMC 0.18u.

V případě FPGA implementace je vylepšení T×A o 31% až 65% a v případě ASIC implementace je vylepšení T×A o 40% až 76% v závislosti na bitové délce počítačového slova $n \in \{64, 126, 162, 256\}$.

n	**SF-AMI**	**AMI**	**Vylepšení** [%]
64	0.2	0.3	43
128	0.4	0.7	76
162	0.6	0.8	40
256	1.0	1.4	39

Tabulka 6: Srovnání implementačního parametru T×A [mm^2×ns] pro SF-AMI a AMI algoritmů v ASIC.

Z předchozího srovnání vyplývá, že jak LS algoritmus ALGORITMUS 5, tak SF-AMI algoritmu ALGORITMUS 8 mají ve srovnání s AMI ALGORITMU 7 lepší výkonnostní parametry, a to jednak časovou složitost a jednak menší hodnotu parametru T×A v implementacích v FPGA a v ASIC.

V [7] jsou uvedeny výsledky celkového srovnání ALGORITMU 7 a ALGORITMU 8 v tabulce 7 je uvedená část toho srovnání. Zlepšení parametru T×A v případě ALGORITMU 8 oproti ALGORITMU 7

n	ALGORITMUS 8	ALGORITMUS 7	**Vylepšení** [%]
128	147.2	204.4	39
160	232.1	369.6	69
192	339.1	512.0	51
256	630.1	820.3	30

Tabulka 7: Srovnání implementačního parametru T×A [mm^2×ns] pro ALGORITMUS 8 a 7 v ASIC.

se pohybuje v rozmezí 39% až 69% v závislosti na počtu bitů počítačového slova

Pro srovnávání úspěšnosti hardwarové implementace byl vybrán ALGORITMUS 7, a to z toho důvodu, že v odborných kruzích je tento algoritmus případně jeho deriváty považovány za nejvýkonnější. Přesto ve srovnáních se SF-AMI algoritmem a LS algoritmem má menší výkonnostní parametry. Další výzkum v této oblasti vidíme v návrhu architektur pro tyto algoritmy, které budou zohledňovat ještě více nejdůležitější vlastnosti, a to optimální počet sčítání/odčítání v průběhu výpočtu MMI.

4 Závěr

V této práci byly prezentovány některé výsledky výzkumu efektivní hardwarové implementace algoritmů multiplikativní modulární inverze — MMI. Byl uveden základní matematický aparát týkající se MMI, který počítá nad konečným tělesem F(p) (nebo tělesem GF(p) k němu izomorfním). Dále byly popsány jednak základní a binární Euklidovy algoritmy pro výpočet GCD, a také rozšířené základní a binární verze Euklidova algoritmu pro výpočet MMI. U těchto algoritmů byla diskutována jejich časová složitost.

Hlavní částí práce je představení dvou originálních binárních algoritmů pro výpočet MMI. Oba algoritmy jsou založeny na rozšířeném Euklidově algoritmu. První z nich takzvaný „Left-Shift“ algoritmus se vyznačuje optimálním prováděním výpočtu MMI, a to vzhledem k minimálnímu počtu operaci sčítání/odčítání. Pokud uvažujeme hardwarovou architekturu, která umožňuje provádět sčítání za delší dobu než posuv se stejnou délkou slova, potom v případě velkých počítačových slov může být hardwarová implementace LS algoritmu více než dvakrát rychlejší než jiné implementace pro výpočet MMI. Jako druhý byl uveden takzvaný „Subtraction-free“ algoritmus, který je upraven pro výpočet Montgomeryho MMI. Tento algoritmus odstraňuje vykonávání opravné operace odečítání ve výpočetních iteracích. Tímto bylo dosaženo teoretického zrychlení první fáze modifikovaného algoritmu oproti původnímu o 25% až 50%. Algoritmus „Left-Shift“ byl implementován v ASIC technologické bázi. Tato implementace byla ve srovnání s implementací Montgomeryho MMI cca 1.3 až 1.4 krát rychlejší pro počítačová slova od 128 bitů do 256 bitů. Pro „Subtraction-free“ algoritmus byly provedeny implementace do technologií ASIC a FPGA. Implementace tohoto algoritmu ve srovnání s Montgomeryho MMI implementací měla cca o 25% až 65% lepší parameter T×A a v případě ASIC byl tento parametr v rozmezí cca o 40% až 75% lepší než Montgomeryho MMI implementovaná v ASIC.

V dalším je výzkum v této oblasti soustředěn na návrh hardwarové architektury pro oba algoritmy, která bude zohledňovat jejich hlavní výhodu, a to optimální počet operací sčítání a odčítání. Při úspěšném návrhu je možné se přiblížit k teoretickým hodnotám zrychleni, tj. v případě LS algoritmu pro velká počítačová slova více než dva a pro SF-AMI je to 1.5.

Výsledky uvedené v této práci ukazují možnost vylepšení známých a využívaných implementací pro výpočet multiplikativní modulární inverze, která je součástí mnoha kryptografických primitiv. Vylepšení její časové nebo prostorové složitosti může mít v konečném důsledku značný dopad na implementaci celého kryptografického systému. Každé urychlení ve výpočtu má v takovém případě dopad na mnoho faktorů implementace. Může se zvýšit její cenová dostupnost, může být odolnější vůči zneužití a útokům, tj. může být bezpečnější, může být ekonomičtější atd. Jednou z oblastí, kde je možné naplnit většinu z těchto očekávání, jsou čipové karty s využitím prvků kryptografie eliptických křivek.

Seznam použité literatury

[1] L. Batina, G. Muurling, "Montgomery in Practice: How to Do It More efficiently in Hardware", *CT–RSA 2002, San Jose, CA, USA*, LNCS 2271, pp. 40–52, 2002.

[2] T. Blum, C. Paar, "High-Radix Montgomery Modular Exponentiation on Reconfigurable Hardware", *IEEE Transaction on Computers*, vol. 50, no. 7, pp. 759–764, 2001.

[3] R. P. Brent, "Analysis of the binary Euclidean algorithm", *New Directions and Recent Results in Algorithms and Complexity, Academic Press, New York*, pp. 321–355, 1976.

[4] B. Bruner, A. Curiger, M. Hofstetter, "On Computing Multiplicative Inverse in GF(2^m)", *IEEE Trans. Computer*, vol.42, pp. 1010–1015, 1993.

[5] J. Buček, R. Lórencz R., "Montgomery Multiplication on FPGA with Modified Carry-Save Encoding", *In: ICSES'04 International Conference on Signals and Electronic Systems. Poznań: PTETIS*, pp. 313–315, 2004.

[6] J. Buček, R. Lórencz R., "Comparing Subtraction-Free and Traditional AMI", In Proceedings of the 2006 IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems. Praha: CTU Publishing House, 2006, vol. 1, p. 97-99. ISBN 1-4244-0184-4.

[7] J. Buček, R. Lórencz R., "Comparison of Different Implementations of Montgomery Modular Inverse in Hardware", Nebublikovaná přednásška, Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier, Université Montpellier II, France. 2007-06-21.

[8] J. Buček, R. Lórencz R., "Comparing Left-shift and Montgomery inverse implementations in ASIC", Nebublikovaná přednásška, 7th International Workshop on Cryptographic Architectures Embedded in Reconfigurable Devices - CryptArchi 2009, Praha, ČR. 2009-06-19.

[9] A. Cilardo, A. Mazzeo, L. Romano, G. P. Saggese: "Carry-Save Montgomery Modular Exponentiation on Reconfigurable Hardware", *In: Design, Automation and Test in Europe Conference and Exhibition Designers' Forum (DATE'04)*, pp. 206–211, 2004.

[10] A. Daly, W. Marnane: "Efficient Architectures for Implementing Montgomery Modular Multiplication and RSA Modular Exponentiation on Reconfigurable Logic", *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, pp. 40–49, 2002.

[11] W. Diffie and M. E. Hellman, "New Directions in Cryptography", *IEEE Transactions on Information Theory*, vol. 22, pp. 644–654, Nov. 1976.

[12] J. D. Dworkin, P. M Glaser, M. J. Torla, A. Vadekar, R. J. Lambert, S. A. Vanstone, "Finite Field Inverse Circuit", *US Patent 6,009,450*, Dec. 28, 1999.

[13] Ö. Eğecioğlu and Ç. K. Koç, "Exponentiation Using Canonical recoding", *Theoretical Computer Science*, vol. 129, no. 2, pp. 407–717, 1994.

[14] I. Z. Emiris, V. Y. Pan, Y. Yu, "Modular Arithmetic for Linear Algebra Computations in the Real Field", *J. Symbolic Computation*, vol. 26, pp. 71–87, 1998.

[15] R. T. Gregory, E. V. Krishnamurthy, "Methods and Applications of Error-Free Computation", *Springer-Verlag, New York, Berlin, Heidelberg, Tokyo*, 1984.

[16] A. Gutub.: "New Hardware Algorithms and Designs for Montgomery Modular Inverse Computation in Galois Fields GF(p) and GF(2^n)", *PhD Thesis, Department of Electrical & Computer Engineering, Oregon State University*, 2002.

[17] A. A. Gutub, A. F. Tenca, Ç. K. Koç, "Scalable VLSI Architecture for GF(p) Montgomery Modular Inverse Computation", *Proceeding of the IEEE Computer Society Annual Symposium on VLSI*, 2002.

[18] A. Gutub, A. Tenca, E. Savas, and Ç. Koç, "Scalable and Unified Hardware to Compute Montgomery Inverse in GF(p) and GF(2^n)", *Workshop on Cryptographic Hardware and Embedded Systems – CHES 2002, Springer-Verlag LNCS*, vol. 2523, pp. 484–499, 2002.

[19] J. Hlaváč, R. Lórencz, "Hardware Architecture for Computing the Modular Inverse", *6th International Workshop on Electronics, Control, Measurement and Signals – ECMS 2003, Liberec: Technical University*, pp. 289–293, 2003

[20] J. Hlaváč, R. Lórencz, "Ordinary Modular Inverse Using AMI – Hardware Implementation", *Proc. IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems– DDECS'03, Poznań, Poland*, pp. 309–310, 2003.

[21] J. Hlaváč, R. Lórencz, "Improved Hardware Architecture for Computing the Modular Inverse Using AMI", *International Conference on Computer, Communication and Control Technologies CCCT 2003 and ISAS 2003, Orlando, USA, International Institute of Informatics and Systemics 3*, pp. 255–260, 2003.

[22] J. Hlaváč, R. Lórencz, "FPGA Implementation of Ordinary Modular Inverse Calculation Using AMI ", *10th International Workshop on Systems, Signals and Image Processing – IWSSIP 2003, Recent Trends in Multimedia Information Processing, Praha: Sdělovací technika, 2003*, pp. 284–287, 2003.

[23] J. Hlaváč, R. Lórencz, "Modular Arithmetic Unit", *Proceedings of the 10th International Conference on Information Systems Analysis and Synthesis. Orlando: IIIS – International Institute of Informatics and Systemics, 2004*, vol. 2, pp. 190 – 195, 2004.

[24] T. Itoh, O. Teechai, S. Tsujii, "A Fast Algorithm for Computing Multiplicative Inverses, in GF(2^t) using normal bases". J. Society for electronic Communications (Japan) 44, pp. 31-36, 1986.

[25] B. S. Kaliski Jr., "The Montgomery Inverse and Its Application", *IEEE Transaction on Computers*, vol. 44, no. 8, pp. 1064–1065, 1995.

[26] D. E. Knuth, "The Art of Computer Programming **2** / Seminumerical Algorithms", *Addison-Wesley, Reading, Mass. Third edition*, 1998.

[27] N. Koblitz, "Elliptic Curve Cryptosystem", *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, January 1987.

[28] Ç. K. Koç, "High-Radix and Bit Recoding Techniques For Modular Exponentiation", *Int'l J. Computer Mathematics*, vol. 40, pp. 139–156, 1991.

[29] Ç. K. Koç, T. Acar, B. S. Kaliski Jr., "Analyzing and Comparing Montgomery Multiplication Algorithms", *IEEE Micro*, vol. 16, Issue 3, pp. 26–33, 1996.

[30] G. Lamé, "Note sur la limite du nombre des divisions dans la recherche du plus grand commun diviseur entre deux nombres entiers", *Comptes rendus de l'academie des sciences*, tome 19, pp. 867–870, 1984.

[31] R. Lórencz, "Algorithms for the Computation of Ordinary Modular Inverse", *Proc. of the Fifth International Scientific Conference: Electronic Computers and Informa 2002, Košice–Herl'any, Slovakia*, pp. 154–158, 2002.

[32] R. Lórencz, "New Algorithm for the Classical Modular Inverse", *Proc. of Workshop on Cryptographic Hardware and Embedded Systems – CHES 2002, August 13-15, 2002, Redwood Shores, California, USA, Springer-Verlag 'LNCS, vol. 2523'*, pp. 57–70, 2002.

[33] R. Lórencz, "New Approaches to Computing the Modular Inverse in Hardware", Habilitation Thesis, CTU, Prague, 2004,

[34] R. Lórencz, Hlaváč, "FPGA Implementation of Arithmetic Unit for Modular Arithmetic", *DDECS – Proceedings of 7th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop. Stará Lesná: Institute of Informatics, Slovak Academy of Sciences, Bratislava, 2004*, pp. 187–190, 2004.

[35] R. Lórencz, J. Hlaváč, "Subtraction-Free Almost Montgomery Inverse Algorithm", Information Processing Letters. 2005, vol. 94, no. 1, p. 11-14.

[36] R. Lórencz, "Způsob generování multiplikativní inverze nad konečným tělesem GF(p)". Patent Úřad průmyslového vlastnictví, 294898. 2005-02-07.

[37] R. Lórencz R., J. Buček, "Comparison of FPGA and ASIC Implementations of Subtraction-free Almost Montgomery Inverse", Nebublikovaná přednášska, 4th International Workshop on Cryptographic Architectures Embedded in Reconfigurable Devices - CryptArchi 2006, Košice, Slovensko.

[38] R. Lórencz, "METHOD FOR GENERATING THE MULTIPLICATIVE INVERSE IN A FINITE FIELD GF(p)". Patent United States Patent and Trademark Office, 7574469. 2009-08-11.

[39] C. McIvor, M. McLoone, J. McCanny, A. Daly, W. Marnane, "Fast Montgomery Modular Multiplication and RSA Cryptographic Processor Architectures", *Proc. 37th IEEE Computer Society Asilomar Conference on Signals, Systems and Computers, Monterey, USA*, pp. 379–384, 2003.

[40] A. J. Menezes, "Elliptic curve Public Key Cryptosystem", *Kluwer Academic Publishers, Boston, MA*, 1993.

[41] P. L. Montgomery, "Modular Multiplication Without Trial Division", *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.

[42] "Nat'l Inst. of Standards and Technology (NIST)", *FIPS Publication 186: Digital Signature Standard*, May 19, 1994.

[43] G. B. Purdy: "A carry-free algorithm for finding the greatest common divisor of two integers", *Computer Math. Appl.* vol. 9, pp. 311-316, 1983.

[44] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[45] K. H. Rosen, "Elementary Number Theory and Its Applications", *Addison-Wesley Publishing Company*, 1993.

[46] J.-J. Quisquarter, C. Couvreur, "Fast Decipherment Algorithm for RSA Public-key Cryptosystem", *Electronics Letters*, vol. 18, no. 21, pp. 905–907, 1982.

[47] E. Savaş, Ç. K. Koç, "The Montgomery Modular Inverse - Revisited", *IEEE Transaction on Computers*, vol. 49, no. 7, July 2000.

[48] E. Savaş, A. F. Tenca, Ç. K. Koç: "A Scalable and Unified Multiplier Architecture for Finite Fields GF(p) and GF(2^m)", *Workshop on Cryptographic Hardware and Embedded Systems – CHES 2000, Springer-Verlag LNCS*, vol. 1965, pp. 277–292, 2000.

[49] J. Stein, "Computational problems associated with Racah algebra", *Journal of Computational Physics*, vol. 1, pp. 397–405, 1967.

[50] A. F. Tenca, Ç. K. Koç, "A Scalable Architecture for Modular Multiplication Based on Montgomery's Algorithm", *IEEE Transaction on Computers*, vol. 52, no. 9, pp. 1215–1221, 2003.

[51] Z. Yan, D. Sarwate, "New Systolic Architectures for Inversion and Division in GF(2^m)", *IEEE Transaction on Computers*, vol. 5, no. 11, pp. 1514–1519, 2003.

Abstract

This work presents an overview of research results in the area of efficient binary algorithms for computing the multiplicative modular inverse. The algorithms are designed with respect to their optimal implementation in hardware in various technologies. The work mainly focuses on the minimum time complexity of the algorithms and the smallest time-area product. Significant improvements were achieved for two types of binary algorithms. The algorithms for computing the multiplicative modular inverse and their implementations are among the most complex portions of many cryptographical primitives, and thus their efficient implementation is important as far as performance, low power consumption, low cost, and also security is concerned.