VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

GENEROVÁNÍ KRYPTOGRAFICKY-BEZPEČNÝCH PRVOČÍSEL POMOCÍ HARDWARE

HARDWARE GENERATOR FOR CRYPTOGRAPHICALLY SECURE PRIMES

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Filip Wagner

AUTHOR

VEDOUCÍ PRÁCE — Ing. Peter Cíbik

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Filip Wagner ***ID:*** 230351

Ročník: 3 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Generování kryptograficky-bezpečných prvočísel pomocí hardware

POKYNY PRO VYPRACOVÁNÍ:

Téma práce je zaměřené na návrh a implementaci generátoru prvočísel s dostatečnou kryptografickou bezpečností na platformě FPGA respektive testu prvočíselnosti a splnění požadavků na kryptografickou bezpečnost prvočísla. Student teoreticky zpracuje problematiku, porovná možnosti a vytvoří návrh zvoleného implementačního řešení komponenty testu prvočíselnosti pro platformu FPGA a jazyk VHDL. Následně student implementuje tento návrh v jazyku VHDL, otestuje jeho funkcionalitu na hardware a zhodnotí dosažené výsledky.

Výstupem bakalářské práce bude rešerše, návrh a funkční realizace zvoleného testu prvočíselnosti pro platformu FPGA v jazyce VHDL splňující požadovanou kryptografickou bezpečnost generovaných prvočísel.

DOPORUČENÁ LITERATURA:

[1] PINKER, Jiří a Martin POUPA. Číslicové systémy a jazyk VHDL. Praha: BEN - technická literatura, 2006. ISBN 80-7300-198-5

[2] BURDA, Karel. Aplikovaná kryptografie. Brno: VUTIUM, 2013. ISBN 978-80- 214-4612-0

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 26.5.2023

Vedoucí práce: Ing. Peter Cíbik

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

UPOZORNĚNÍ:

Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRAKT

Tato práce se zaobírá problematikou prvočísel, jejich užití v kryptografii, možnostech generování prvočísel, kryptografií realizovanou s pomocí hardware, užitím platformy FPGA, popisem jazyka VHDL a porovnáním způsobů generování prvočísel.

KLÍČOVÁ SLOVA

Prvočísla, Asymetrická kryptografie, Kryptografie na integrovaných obvodech, FPGA, VHDL, Generátor prvočísel

ABSTRACT

This work tackles the topics of prime numbers, their use in cryptography, options for generating prime numbers, cryptography done on hardware, usage of FPGA, description of VHDL and comparison of approaches in generating prime numbers.

KEYWORDS

Prime numbers, Asymmetric cryptography, Cryptography on integrated circuits, FPGA, VHDL, Prime number generator

Vysázeno pomocí balíčku `thesis` verze 4.07; `http://latex.feec.vutbr.cz`

WAGNER, Filip. *Generování kryptograficky-bezpečných prvočísel pomocí hardware*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023, 77 s. Bakalářská práce. Vedoucí práce: Ing. Peter Cíbik

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Filip Wagner
VUT ID autora:	230351
Typ práce:	Bakalářská práce
Akademický rok:	2022/23
Téma závěrečné práce:	Generování kryptograficky-bezpečných prvočísel pomocí hardware

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Tímto bych rád poděkoval vedoucímu bakalářské práce panu Ing. Peteru Cíbikovi za odborné vedení práce, poskytnuté konzultace a předané znalosti z oblasti VHDL. Cením si také jeho trpělivosti a přívětivého přístupu při vedení mé bakalářské práce.

Dále také děkuji panu Ing. Davidu Smékalovi za poskytnutí přístupu ke školnímu serveru pro vývojové prostředí Vivado a jeho správu.

Obsah

Úvod **21**

1 Matematické vlastnosti prvočísel **23**
1.1 Definice prvočísla . . . 23
1.2 Prvočísla, jakožto nekonečná řada . . . 23
1.2.1 Mersennova prvočísla . . . 24
1.2.2 Aproximace počtu prvočísel v daném úseku . . . 25
1.2.3 Goldbachova hypotéza . . . 28
1.3 Metody hledání prvočísel . . . 29
1.3.1 Pravděpodobnostní testy prvočíselnosti . . . 30
1.3.2 Deterministické testy prvočíselnosti . . . 32

2 Užití prvočísel v kryptografii **33**
2.1 Kryptograficky-bezpečná prvočísla . . . 33
2.2 Kryptografické algoritmy založené na faktorizaci velkého čísla . . . 34
2.3 Kryptografické algoritmy založené na problému diskrétního logaritmu 34

3 Kryptosystémy na integrovaných obvodech **37**
3.1 SIM karty . . . 37
3.2 EMV platební karty . . . 38
3.3 FPGA v kryptografii . . . 38

4 Jazyky pro popis hardware **41**
4.1 Porovnání VHDL a Verilogu . . . 41
4.2 Jazyk VHDL . . . 41

5 Praktické řešení problematiky **43**
5.1 Určení vhodného testu prvočíselnosti . . . 43
5.2 Model generování prvočísel v Pythonu . . . 45
5.3 Definice kryptograficky-bezpečných prvočísel pro potřeby práce . . . 46
5.3.1 Generovaná prvočísla dle jejich užití . . . 46
5.3.2 Určení parametrů generátoru prvočísel v praktické části . . . 50
5.4 Implementované algoritmy . . . 50
5.4.1 Inicializace vstupních hodnot testu . . . 50
5.4.2 Problém modulárního umocňování na velká čísla . . . 51
5.4.3 Montgomeryho forma, násobení a umocňování . . . 52
5.4.4 Problém násobení velkých čísel . . . 57
5.4.5 Problém přetečení a nenaplnění hodnot . . . 58

6 Výsledky praktické části **61**
6.1 Vyhodnocení jednotlivých modelů 61
6.1.1 Model č.1 61
6.1.2 Model č.2 62
6.1.3 Model č.3 62
6.1.4 Modely č.4 a č.5 62
6.2 Chybovost modelů 63
6.3 Porovnání výsledků 63

Závěr **65**

Literatura **67**

Seznam symbolů a zkratek **71**

Seznam příloh **73**

A Zdrojové kódy použitých Python modelů **75**
A.1 Generování prvočísel s pomocí Fermatova testu 75
A.2 Generování prvočísel s pomocí Miller-Rabinova testu 76

B Struktura elektronické přílohy **77**

Seznam obrázků

1.1 Graf funkce $\pi(x)$ a její aproximace 26
1.2 Graf odchylky aproximace od funkce $\pi(x)$ 28
1.3 Goldbachova kometa 29
2.1 Sčítání bodu na eliptické křivce 35
3.1 Schéma autentizace v kryptosystému v sítích GMS 38
5.1 Doby průběhů testů na Python modelu 45
B.1 Struktura souborů odevzdaných v rámci elektronické přílohy 77

Seznam tabulek

1.1 Porovnání počtu prvočísel s jeho aproximací 27
5.1 Požadavky na M-R testování pro užití problému faktorizace na prvočísla 48
5.2 Požadavky na M-R testování pro užití problému diskrétního logaritmu 49
5.3 Požadavky na kofaktor h pro prvočísla n 49
6.1 Porovnání jednotlivých modelů 64

Seznam výpisů

5.1 Modulární umocňování ve vzestupném režimu 52
5.2 Modulární umocňování v sestupném režimu 52
5.3 Převod malých svědků do Montgomeryho formy 54
5.4 Nalezení inverzního prvku s pomocí Euklidova teorému 55
5.5 Montgomeryho součin 55
5.6 Montgomeryho umocňování 56
5.7 Násobička pro 160 bitové součinitele 58
A.1 Model generování prvočísla pomocí Fermatova testu v jazyce Python 75
A.2 Model generování prvočísla pomocí Miller-Rabinova testu v jazyce Python 76

Úvod

Moderním problémem dnešní doby jsou nepochybně kybernetické hrozby, které vnímáme jakožto stále častější společenské téma. Potřeba zajistit efektivní bezpečnost s pomocí šifrování tak vyvstává i na drobných zařízeních, které nemohou disponovat značným výkonem. Tento problém se stává ještě aktuálnější s každodenním užíváním technických prostředků jako je třeba platební karta, či mobilní telefon. Právě v případě těchto zařízení nesoucích cenná aktiva se setkáváme s vyšším zájmem potenciálních útočníků a tedy i s vyšší mírou jejich ohrožení. Z tohoto důvodu je požadováno dosáhnout míry zabezpečení, kterou nelze prolomit ani na strojích na vrcholu v současné době dostupného výpočetního výkonu.

Nejčastěji používaný kryptografický mechanizmus, co zajišťuje takto výrazný nepoměr mezi výpočetními požadavky na implementaci vůči požadavkům na jeho prolomení je asymetrická kryptografie založená na problematice faktorizace na prvočísla, či problému diskrétního logaritmu. Pro správnou implementaci takového zabezpečení je klíčové zajistit na šifrovacím zařízení generátor kryptograficky-bezpečných prvočísel. Jakožto kryptograficky-bezpečné prvočíslo rozumíme prvočíslo odpovídající současným bezpečnostním požadavkům na jeho implementaci pro daný algoritmus. Jedním z ideálních prostorově méně náročných řešení se tak stává užití nadesignovaného hardware dedikovaného právě pro tyto výpočty. Dvěma dominantními ve světě nejrozšířenějšími hardware popisujícími jazyky, jimiž lze dosáhnout zmíněného cíle, jsou VHDL a Verilog.[1]

Tato práce se bude věnovat vlastnostem prvočísel, jejich užití v kryptografii a prozkoumá možnosti algoritmů generování prvočísel na základě příslušných testů prvočíselnosti. Dále budou představeny příklady užití kryptografie na integrovaných obvodech v praxi, hardware popisující jazyk VHDL s jehož pomocí bude ve vývojovém prostředí Vivado navrhnuto, testováno a zrealizováno vícero verzí generátoru kryptograficky-bezpečných prvočísel. Práce se také zaměřuje na popis matematických algoritmů, které implementace efektivního generátor kryptograficky-bezpečných prvočísel obnáší.

1 Matematické vlastnosti prvočísel

Prvočísla jsou matematickým konceptem přinejmenším známým již od dob starověkého Řecka. Některé zdroje poukazují také na možnost, že by prvočísla objevili již staří Egypťané [2]. Ovšem nejstarším dochovaným dokumentem zabývajícím se touto problematikou jsou Eukleidovy Základy, respektive jejich kniha IX, datovaná do doby kolem roku tři sta před naším letopočtem [3]. Od jejich objevení daly prvočísla vzniknout mnoha matematickým problémům z nichž některé trápí a udivují matematiky dodnes a dávají tak najevo, že kapitola prvočísel není ve světě matematiky ani po více než dvou tisíci letech plně probádána [4].

1.1 Definice prvočísla

V první řadě je podstatné podotknout, že při práci s prvočísly se ve většině případech omezujeme pouze na množinu přirozených čísel. Výjimku v této problematice tvoří různé spojité funkce pracující s odchylkou od diskrétního charakteru povahy prvočísel zapřičiněné jejich často pravděpodobnostní povahou. [1] Tedy inkrementovanou řadu začínající jedničkou značenou symbolem $\mathbb{N}$.

$$\mathbb{N} = \{1; 2; 3; \ldots \infty\} \tag{1.1}$$

Prvočíslo je pak takové přirozené číslo, které je bezezbytku dělitelné pouze samo sebou a jedničkou, přičemž číslo jedna se z této definice explicitně vyřazuje. Ostatní přirozená čísla, vyjma jedničky, která má svůj zvláštní status, co nespadají do této definice se nazývají čísla složená. Princip tohoto pojmenování spočívá v tom, že každé složené číslo lze jednoznačně zapsat jakožto součin dvou či více prvočísel, ze kterých se "skládá". Tento způsob zápisu čísla se nazývá rozklad nebo také faktorizace na prvočinitele, popřípadě prvočísla. [2]

$$4 = 2 \cdot 2 \qquad 18 = 2 \cdot 3 \cdot 3 \qquad 51 = 3 \cdot 17 \tag{1.2}$$

1.2 Prvočísla, jakožto nekonečná řada

To, že je prvočísel nekonečně mnoho dokázal již ve své publikaci Eukleides v jeho větě o prvočíslech [3]. Její důkaz lze provést sporem [3] a spočívá v tom, že pokud

[1] Příkladem může být později zmiňovaná aproximace počtu prvočísel menších než zadané číslo n

[2] Jednoznačnost zápisu přirozeného čísla většího než jedna v podobě rozkladu na prvočísla formuloval Eukleides jakožto Základní větu aritmetiky [3]

[3] Eukleidův originální důkaz je proveden, ale rozborem případů, kde dochází k závěru, že nové číslo je buď prvočíslo a nebo neznáme prvočísla, ze kterých se dané číslo skládá [5].

existuje konečný počet všech prvočísel, tak musí existovat číslo složené vynásobením všech těchto prvočísel, ale pak také musí existovat číslo o jedna větší, které bude mít po dělení každým ze známých prvočísel zbytek jedna. Z toho vyplývá, že takové číslo musí být nesoudělné se všemi objevenými prvočísly a tím pádem se jedná o další prvočíslo.

$$\sum_{n=1}^{\infty} p_1 \cdot p_2 \cdot p_3 \cdot \cdots \cdot p_n + 1 = p_{n+k} \qquad [4], \qquad k \in \mathbb{N} \qquad (1.3)$$

Mnoho matematiků usilovalo o nalezení jakékoliv závislosti v této číselné řadě a v následujících podkapitolách se na některé podíváme.

1.2.1 Mersennova prvočísla

Pozoruhodný matematický objev na poli prvočísel přinesl francouzský mnich a matematik Marin Mersenne, když ve své publikaci *Cogitata Physico-Mathematica* definoval jev, který dnes na jeho počest známe jakožto Mersennova prvočísla. Obecně Mersennovým číslem je označováno číslo, které lze zapsat jako: $2^n - 1$, *kde* $n \in \mathbb{N}$. Objevená závislost spočívá v tom, že pokud je číslo n prvočíslo, tak v některých případech je Mersennovo číslo prvočíslem. Zprvu se tato závislost jeví, jakožto jistý způsob hledání nových prvočísel, ale můžeme si spolehlivost takovéhoto postupu ověřit už na prvních několika prvočíslech. Čísla $2^2-1 = \mathbf{3}$; $2^3-1 = \mathbf{7}$; $2^5-1 = \mathbf{31}$; $2^7-1 = \mathbf{127}$ sice prvočísly jsou, ale už u $2^{11} - 1 = \mathbf{2047}$ dojdeme k závěru, že se jedná o číslo složené a lze jej takto rozložit: $2047 = 23 \cdot 89$

Pro tyto výjimky byla nejprve Mersennova prvočísela dokazována z opačného konce. Tedy pokud nějaké z Mersennových čísel je prvočíslem, tak i exponent n musí být prvočíslem. Onu závislost potřebnou k hledání nových vyšších prvočísel s pomocí Mersennových čísel později objevil É. Lucas a následně vylepšil D.H.Lehmer [6], což dalo vzniknout ve třicátých letech minulého století tzv. Lucas-Lehmerovu testu prvočíselnosti.

Dle pravidel Lucas-Lehmerova testu je každé číslo n Mersennovým prvočíslem pokud splňuje tyto podmínky[7]:

- $n = 2^p - 1$, kde p je prvočíslo
- a v posloupnosti definované jakožto $u_0 = 4$ *a* $u_{k+1} = (u_k^2 - 2)\ mod\ (n)$ je prvek $u_{p-2} = 0$ pro $p \geq 3$

Pokud budeme test považovat za potenciální generátor prvočísel, je zde vidět, že doba běhu takového generátoru exponenciálně roste s velikosti vstupního prvočísla p, podle kterého zkoušíme vygenerovat nové prvočíslo n.

[4] Nikoliv však $\cdots = p_{n+1}$, jelikož se nemusí nutně jednat o další prvočíslo v pořadí.

1.2.2 Aproximace počtu prvočísel v daném úseku

Jak již ve starověku Eukleides dokázal, prvočísla jsou nekonečnou řadou čísel. Je jasné, že vždy tedy musí existovat větší prvočíslo, ale také musí být konečný počet těch menších. Úplnou metodou pro určení tohoto počtu by bylo postupně prověřit každé číslo v rozsahu a spočíst kolik se zde nachází prvočísel. Obecným značením funkce, co definuje počet prvočísel do daného x, je pak: $\pi(x)$.

Jednodušší řešení v podobě aproximace představil nejspíše [5] již Carl Friedrich Gausse v podobě **Prvočíselné věty**. Její důkaz však podali až o téměř sto let později v roce 1896 nezávisle na sobě matematici Jacques Hadamard [6] a Charles Jean de la Vallée Poussin [7].

Prvočíselná věta udává spojitou funkci [8], k jejímuž průběhu by se funkce $\pi(x)$ měla blížit v nekonečnu. Matematické vyjádření Prvočíselné věty vypadá pak takto:

$$\lim_{x \to \infty} \frac{\pi(x)}{\frac{x}{\log x}} = 1 \tag{1.4}$$

Pro obecné x tedy platí zápis:

$$\pi(x) \sim \frac{x}{\log x} \tag{1.5}$$

Čímž byla takto definována první [9] prvočíselná funkce $f(x) : \frac{x}{\log x}$ pro aproximaci počtu prvočísel do daného x [8]. Průběh této funkce v porovnání se skutečným počtem prvočísel v rozsahu až po $x = 1000$ je zobrazen v následujícím grafu.

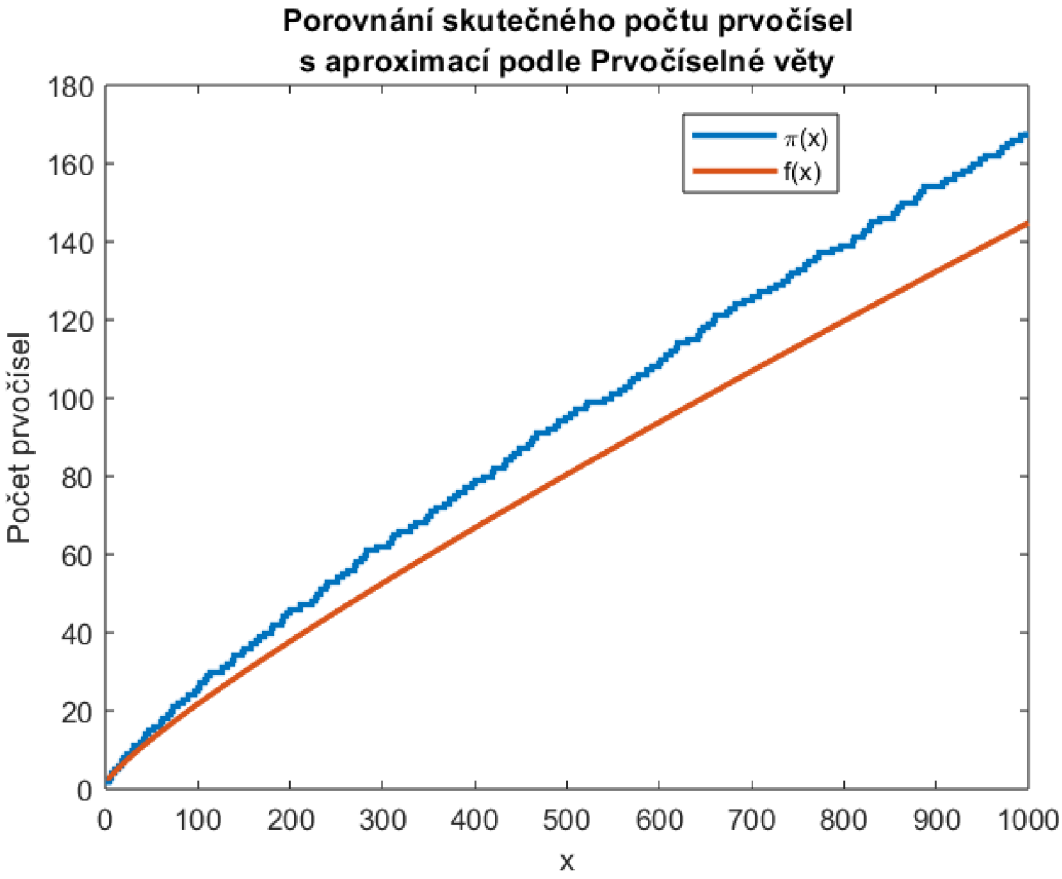
[5] Vícero matematiků se problematikou zabývalo a nelze tento objev jednoznačně přiřadit

[6] HADAMARD, J. Sur la distribution des zéros de la fonction $\zeta(s)$ et ses conséquences arithmétiques. [s.l.]: Bulletin Société Mathématique de France, 1896

[7] DE LA VALLÉE-POUSSIN, Ch. J. Recherches analytiques la théorie des nombres premiers. Brusel: Ann. Soc. scient., 1896.

[8] Z diskrétní povahy prvočísel nemůže tedy přesně opisovat funkci $\pi(x)$

[9] Existují i novější a složitější alternativy s vyšší mírou přesnosti, jako je třeba Rimannova funkce $R(x)$[8]



Obr. 1.1: Porovnání skutečného počtu prvočísel s jeho aproximací

Na první pohled se od sebe průběhy zobrazených funkcí v grafu vzdalují, což vede k pochybnostem ohledně přesnosti funkce $f(x) : \frac{x}{\log x}$ pro popis $\pi(x)$. Pro $x = 1000$ je dokonce odchylka $\delta_{1000} = 13,83\ \%$.

V případě naší limity blížící se k nekonečnu je potřeba prozkoumat průběh těchto funkcí i pro větší čísla. Ten si můžeme zobrazit v následující tabulce z dat z *On-Line Encyclopedia of Integer Sequences* [10]

[10]Tabulka byla vytvořena s pomocí dat dostupných na: https://oeis.org/A006880 , repektive jejich rozšířené verze od Davida Baugha dostupné na https://oeis.org/A006880/b006880.txt a https://oeis.org/A057834

x	$\pi(x)$	$\frac{x}{\log x}$ *	δ_x **
10^1	4	4	-8,57
10^2	25	22	13,14
10^3	168	145	13,83
10^4	1 229	1 086	11,66
10^5	9 592	8 686	9,45
10^6	78 498	72 382	7,79
10^7	664 579	620 421	6,64
10^8	5 761 455	5 428 681	5,78
10^9	50 847 534	48 254 942	5,10
10^{10}	455 052 511	434 294 482	4,56
10^{29}	1520698109714272166094258063	1497567178976730440176306617	1,52

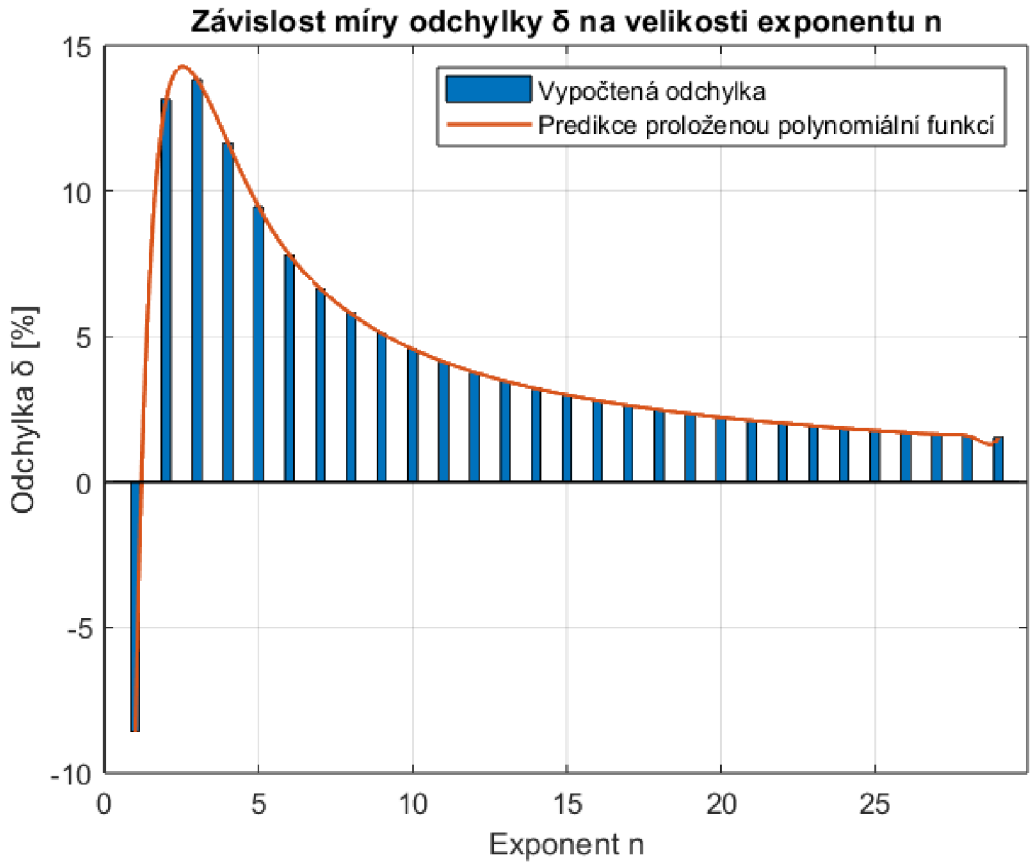
Tab. 1.1: Porovnání počtu prvočísel s jeho aproximací

* *Hodnoty jsou zaokrouhleny na celá čísla*

** *Odchylka je vypočtena z nezaokrouhlených hodnot a vyjádřena v procentech*

Právě 10^{29} je největší mocninou deseti dosazenou za x, ke které se doposud podařilo vyjádřit hodnotu $\pi(x)$ a lze zde tedy vyčíslit odchylku od předpisu $\frac{x}{\log x}$.
Z tabulky lze vyčíst podle klesající odchylky stoupající efektivitu předpisu s rostoucím číslem x, což odpovídá původnímu $\lim_{x\to\infty} \frac{\pi(x)}{\frac{x}{\log x}} = 1$.

Následující graf zobrazuje změnu této odchylky δ v závislosti na exponentu n, kde vstupní hodnota pro porovnávané funkce $x = 10^n$. Graf je tedy ve své podstatě logaritmický na ose x. Pro znázornění postupné změny v odchylce jsou hodnoty grafu proloženy polynomem dvacátého stupně.



Obr. 1.2: Porovnání skutečného počtu prvočísel s jeho aproximací

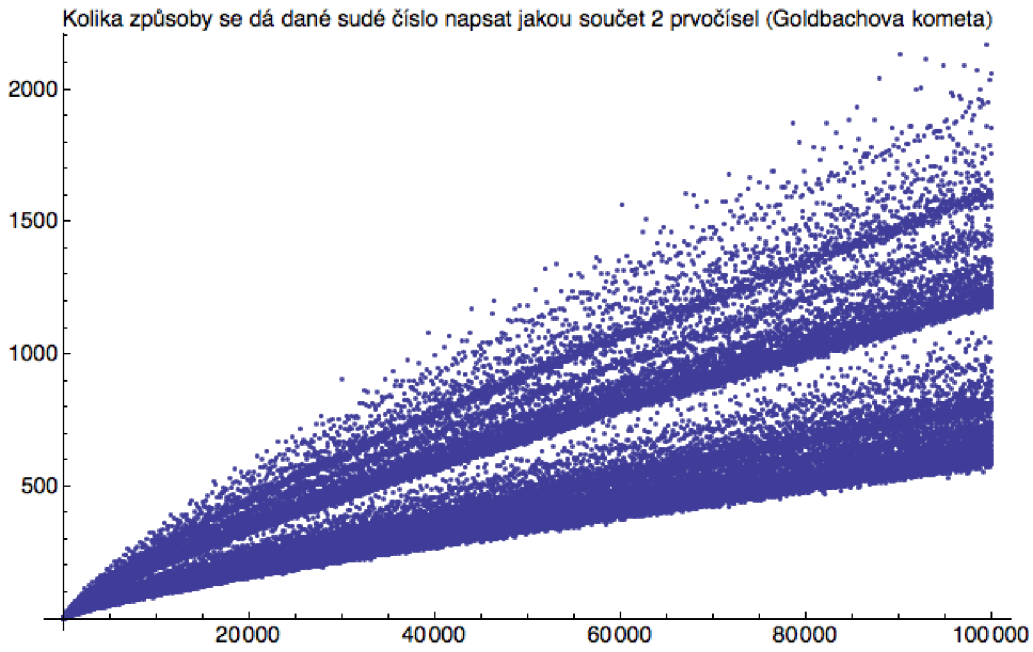
1.2.3 Goldbachova hypotéza

Mimo problematiku aproximace, kde dosažení cíle je měřeno odchylkou od skutečnosti, jak tomu bylo v předešlé kapitole, je matematika prvočísel protkána také mnoha binárními spory o tvrzení, u kterých se hledá, zda jsou, či nejsou pravda. Příkladem pravdivě dokázaného tvrzení pak může být na začátku zmíněná Eukleidova věta o prvočíslech [3], která tvrdí, že je jich nekonečně mnoho.

Je tu ovšem i natolik sporné tvrzení, jež se nedaří již po více než 270 let obecně dokázat, že bývá pochybováno o jeho rozhodnutelnosti. Tím je Goldbachova hypotéza pojmenována po pruském matematikovi Christianu Goldbachu. Problém se poprvé vyskytl v korespondenci se slavným švýcarským matematikem Leonhardem Eulerovem [4]. Respektive původní znění Goldbachova problému bylo, že každé liché číslo větší než 5 lze rozložit na součet tří prvočísel, což bylo také dokázáno důkazem

úplným [11]. Ovšem odvozený Eulerův problém [12], dle kterého lze každé sudé číslo rozložit na součet dvou prvočísel se doposud dokázat ani vyvrátit nepodařilo.

Vizualizaci počtu způsobů rozložená sudých čísel na součet dvou prvočísel se říká Golcbachova kometa [9]. Z jejího omezení zdola můžeme jen předpokládat, ale nikoliv dokázat platnost Goldbachovy hypotézy.



Obr. 1.3: Goldbachova kometa [9]

1.3 Metody hledání prvočísel

Způsob hledání prvočísel v podstatě spočívá v určení metody výběru vstupních kandidátů a použitých testů prvočíselnosti. Nejprimitivnější přístup, který je nám blízký u malých prvočísel, je zkoušet každé nové číslo podělit doposud nalezenými prvočísly. Tomuto postupu se říká *Trial division* [13] [7].

Z hlediska zpracování metody *Trial division* člověkem běžně optimalizujeme algoritmus, jak ze strany výběru prověřovaných kandidátů, tak postupu při testu.
Po zaznačení prvočísla 2 již nikdo neuvažuje nad sudými čísly. Podobně se dá vyřadit čísla dělitelná 3 po spočtení ciferného součtu pro určení dělitelnosti 3 a dále také automaticky vyřazujeme čísla končící 5, jelikož musí být také číslem 5 dělitelná.

[11] Došlo předpisem k omezení oblasti nutné k prověření shora a všechna čísla v ní byla následně prozkoušena

[12] Přeneseně nazýván Goldbachovou hypotézou

[13] V překladu doslova "zkušební dělení"

V části postupu pak využíváme skutečnosti, že libovolné složené číslo n musí být dělitelné prvočíslem p, pro které platí, že $p \leq \sqrt{n}$. To je zapříčiněno tím, že nejmenší z x součinitelů může být nanejvýš x-tou odmocninou ze součinu. Pokud bychom jej navýšili přes tuto hranici, tak tato operace proběhne na úkor ostatních a naše číslo již nemůže být nejmenším, jelikož ve stavu x-té odmocniny jsou si všichni součinitelé rovni. Minimální počet součinitelů složeného čísla jsou 2 a proto je hranicí pro smysluplné zkoušení u *Trial division* $\sqrt{n}$.

Dalším zlepšením ze strany prováděného testu pro *Trial division* je jeho verze *Wheel factorization* [7]. Tento algoritmus snižuje nároky na znalost prvočísel menších než $\sqrt{n}$ tím, že jsou zkoušena pouze čísla nesoudělná s již testovanými prvočísly.

1.3.1 Pravděpodobnostní testy prvočíselnosti

Pravděpodobnostní testy prvočíselnosti udávají výsledek jen s určitou mírou jistoty, kterou lze dále navýšit jejich opakováním za použití jiných vstupních koeficientů. Pokud chceme tyto testy využít k hledání prvočísel, celý postup spočívá nejprve ve výběru náhodného zkoušeného čísla, které pokud testy projde, bude s danou pravděpodobností prvočíslem. Praxe spočívá v iterování těchto testů do takové míry, až je pro problematiku jejich možná chybovost při aplikaci nepodstatná.

Fermatův test

Pro svou povahu a chybovost bývá Fermatův test spíše označován testem složenosti čísla, jelikož pokud je číslo tímto testem označeno za složené je tomu s jistotou tak. Test si zakládá na Malé Fermatově větě vycházející z vlastností mocnění v kongruenci. Exponent jedné strany v kongruenci lze vždy modulovat číslem $\phi(n)$, což vyplývá z vlastností algebraických grup $mod(n)$. Tedy:

$$x^a \; mod(n) = x^{a \; mod(\phi(n))} \; mod(n) \tag{1.6}$$

Funkce Euler phi $\phi(n)$ vyjadřuje počet přirozených menších čísel nesoudělných se zadaným přirozeným číslem n. Jelikož prvočísla nelze dále dělit, jejich hodnota $\phi(p) = p - 1$. Malá Fermatova věta pak vychází z odvození, že:

$$a^p \; mod(p) = a \; mod(p) \quad | : a \qquad \Rightarrow \qquad a^{p-1} \; mod(p) = 1 \tag{1.7}$$

Přepíšeme-li výraz z podoby $\ldots mod(p) = \ldots mod(p)$ na modulární kongruenci $\cdots \equiv \ldots mod(p)$ dostaneme vyjádření Malé Fermatovy věty:

$$a^{p-1} \equiv 1 \; mod(p) \tag{1.8}$$

Samotný Fermatův test pak spočívá ve zkoušení platnosti této věty v případě potencionálního prvočísla n pro různá čísla a, kde $1 < a < n - 1$ [14]. Každé číslo a, pro které test neplatí se pak nazývá Fermatův svědek složenosti čísla [7].

Značnou komplikací pro praktické užití tohoto testu představují tzv. **Carmichaelova čísla**. Jedná se nekonečnou řadu složených čísel, která vždy projdou všemi iteracemi testu a budou tak stále chybně považována za prvočísla. Prvních 7 Carmichaelových čísel objevil ovšem bez povšimnutí širší matematické veřejnosti český mnich a matematik Václav Šimerka již v roce 1885 [10]. Tedy čísla: 561, 1105, 1729, 2465, 2821, 6601 a 8911 [15]. Jméno pak nesou podle amerického matematika Roberta Daniela Carmichaela, jenž učinil podobný objev a ve svém článku z roku 1910 nalezl čísla vyjádřená jako: $3 \cdot 11 \cdot 17,\ 5 \cdot 13 \cdot 17,\ 7 \cdot 13 \cdot 31,\ 7 \cdot 31 \cdot 73$, tedy první, druhé, páté a deváté Carmichaelovo číslo [11].

Pro tyto od testovaného n odvozené nedostatky je přesnost Fermatova testu sporná. Zanedbáme-li ovšem možnost, že by testované číslo mohlo být Carmichaelovým, pak vždy alespoň polovina možných jsou Fermatovi svědci složenosti čísla a proto v nejhorším případě pravděpodobnost, že test se mýlí v označení čísla za prvočíslo, je $(\frac{1}{2})^k$, kde k je počet provedených iterací testu.

Miller-Rabinův test

Jedná se o v dnešní době nejpoužívanější test prvočíselnosti. Jeho základy položil v roce 1976 ve své publikaci Gary Lee Miller, kde ovšem cílil na test deterministický založený na nedokázané zobecněné Rimannově hypotéze [12]. Úpravy tohoto algoritmu na důkazně podložený pravděpodobnostní test dosáhl o čtyři roky později Michael O. Rabin [13].

Test vychází a lze jej odvodit z vyjádření Fermatova testu, respektive Malé Fermatovy věty v podobě $a^{p-1} \equiv 1\ mod(p)$ vyjádřené v 1.8
Pokud rovnici odmocníme, dostaneme nepochybně platné vyjádření, že:

$$\sqrt{a^{p-1}} \equiv \pm\sqrt{1}\ mod(p) \tag{1.9}$$

A jelikož $p - 1$ musí být v případě lichých prvočísel [16] sudé číslo, lze jej zapsat s-tou mocninou dvojky vynásobenou číslem d. Pak je nový tvar naší odmocněné Malé Fermatovy věty:

$$a^{2^{s-1}d} \equiv \pm 1\ mod(p) \tag{1.10}$$

Miller-Rabinův test je následně proveden postupným odmocňováním výrazu v podobě dosazování za $s - 1$ stále menší a menší r až po nulu.

[14] Při zvolení čísel 1 a p-1 výraz vždy vychází a nenese tak žádnou výpovědní hodnotu.
[15] Řadu prvních 33 Carmichaelových čísel lze nalézt na: https://oeis.org/A002997
[16] Všechna krom 2

Průběh testu prošel, pokud pro zvolené číslo a z rozsah $2 \leq a \leq n-2$ platí rovnice $a^{2^r d} \equiv \pm 1 \; mod(n)$ pro všechna r z rozsahu $0 \leq r \leq s-1$, kde s je určeno z rovnice $2^s d = n-1$ a n je testované číslo

Chybovost tohoto testu pak Rabin [13] vyčíslil na $(\frac{1}{4})^k$, kde k je počet provedených iterací testu, jelikož pro každé složené číslo dosazené za p jsou $(\frac{3}{4})$ všech možných a svědky jeho složenosti. Míra chyby je tedy 2^k-krát menší než u Fermatova testu a zároveň jej nesužují nedostatky v podobě Carmichaelových čísel.

1.3.2 Deterministické testy prvočíselnosti

Deterministické testy prvočíselnosti jsou testy, které si jsou nepochybně jisté, že zkoušené číslo je prvočíslem, pokud ho tak označili. Mezi ně by se dal zařadit i v úvodu zmíněný *Trial division*.

Lucas-Lehmerův test

Lucas-Lehmerův test je deterministickým testem zaměřeným na hledání Mersennových prvočísel. V podstatě se jedná o algoritmus vysvětlený v části věnované Mersennovým prvočíslům, kde je zvoleno Mersennovo číslo n odvozené od známého prvočísla p vztahem: $n = 2^p - 1$
následně v posloupnosti definované jakožto $u_0 = 4 \; a \; u_{k+1} = (u_k^2 - 2) \; mod \; (n)$ je prvek $u_{p-2} = 0$ pro $p \geq 3$

Pro povahu tohoto testu, kdy z řádově menšího prvočísla s absolutní jistotou určujeme vyšší prvočíslo, je tento postup využíván k "lámání rekordů"při hledání nejvyšších prvočísel. Na tomto principu funguje projekt GIMPS (Great Internet Mersenne Prime Search), který spoluprací zainteresovaných uživatelů hledá další Mersennova prvočísla. Současně největším nalezeným prvočíslem je 51. Mersennovo prvočíslo odpovídající $2^{82589933} - 1$ a bylo nalezeno 7. prosince 2018 [17].

[17] Výsledky projektu jsou dostupné na: https://www.mersenne.org/report_milestones/

2 Užití prvočísel v kryptografii

Základním algebraickým prvkem využívající prvočísla jsou pro kryptografii konečná pole $GF(p^k)$, což je v podstatě množina $\mathbb{Z}_{p^k} = \{0, 1, \dots p^k - 1\}$ sdružená s operací $mod(p^k)$ [14]. Příkladem nekonečného pole mohou být reálná čísla $\mathbb{R}$ jdoucí k nekonečnu. V našem případě konečných polí se pomocí operace modulo cyklicky vracíme na začátek. Pro kryptografii toto představuje dvě klíčové výhody. Zjednoduší se výpočty na menší čísla a samotná operace modulo je ireversibilní a skrývá tak své konkrétní vstupy.

Důležité je také zmínit, že prvočísla se výhradně užívají v asymetrické kryptografii, kde jsou díky jejich vlastnostem generovány klíčové páry.

2.1 Kryptograficky-bezpečná prvočísla

V kontextu kryptografie můžeme za bezpečná prvočísla považovat taková prvočísla, která jsou v současných podmínkách natolik velká, či splňují příslušné požadavky, že z nich složené kompozity není v dnešní době možné efektivně faktorizovat v polynomiálním čase [14]. Otázku dostatečné velikosti řeší ve svých doporučení NIST (National Institute of Standards and Technology) a v současné době například pro šifru RSA doporučuje generovat prvočísla o minimální délce 1024 bitů [15].

Přísnějším bezpečnostním kritériem by byla volba silného prvočísla. To je pak prvočíslo, co svými vlastnostmi přímo cílí na zobtížnění faktorizace kompozitu n pomocí Pollardova n-1 algoritmu a Williamsova n+1 algoritmu [16]. Pro toto prvočíslo p pak platí, že v rozkladu $p - 1 = a * q_1$ a $p + 1 = b * q_2$ jsou q_1 *a* q_2 velká prvočísla s danou minimální délkou[1]. Silná prvočísla také požadují i rozklad $q_1 - 1 = c * q_3$, kde q_3 je opět silným prvočíslem pro zobtížnění cyklického útoku na RSA [2].

V teorii se také pojmem bezpečné prvočíslo označuje prvočíslo q se vztahem k jinému prvočíslu p v podobě $q = 2p + 1$. Jedná se tak o čísla s obdobou části požadavků na silná prvočísla. V praxi ovšem silná prvočísla zohledňuji více kritérií a jsou snazší na nalezení, jelikož jsou častější [3][4]

[1] Délkové požadavky dle příslušných délek klíče jsou popsány NISTem v Digital Signature Standard dostupném na: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf#page=60

[2] Neustálé šifrování veřejným klíčem by eventuálně mělo vést k získání původní zprávy, ale vzhledem k délce klíčů, která se váže na nutný počet iterací pro tento útok, je jeho úspěch v praxi nepravděpodobný [16]

[3] Seznam prvních několika bezpečných prvočísel: http://oeis.org/A005385/list

[4] Seznam několika prvních silných prvočísel: https://oeis.org/A051634/list

2.2 Kryptografické algoritmy založené na faktorizaci velkého čísla

Řešení problému faktorizace neboli rozkladu na prvočinitele celých čísel má v současné době s využitím nejpokročilejších algoritmů přinejmenším subexponenciální složitost [5] [17]. Pro značný nepoměr mezi obtížností řešení tohoto problému a obtížností vytvoření takového složeného čísla se jedná o optimální matematický problém pro aplikaci v kryptografii. Na jeho principech pak stojí nejpoužívanější asymetrická šifra **RSA**.

2.3 Kryptografické algoritmy založené na problému diskrétního logaritmu

Dalším problémem, co doposud nemá řešení proveditelné v polynomiálním čase, je hledání diskrétního logaritmu. Problém spočívá v hledání čísla x v multiplikativní cyklické grupě $\mathbb{Z}_n^*$ z předpisu $g^x = X$, pokud známe čísla g i X. Obráceně lze $g^x mod(n) = X$ snadno spočítat, ovšem pro ireversibilitu operace mod(n) si nemůžeme být jisti kterému $kn + X$, kde $k \in \mathbb{Z}$ je g^x rovno [18]. Právě za číslo n bývá dosazováno velké prvočíslo pro požadavek na jeho nesoudělnost s číslem g. Tohoto jevu pak využívají kryptografické algoritmy jako je **Diffie-Hellmanův protokol**, **DSA** (Digital Signature Algorithm), či šifrovací algoritmus **ElGamal**.

Kryptografie na eliptických křivkách

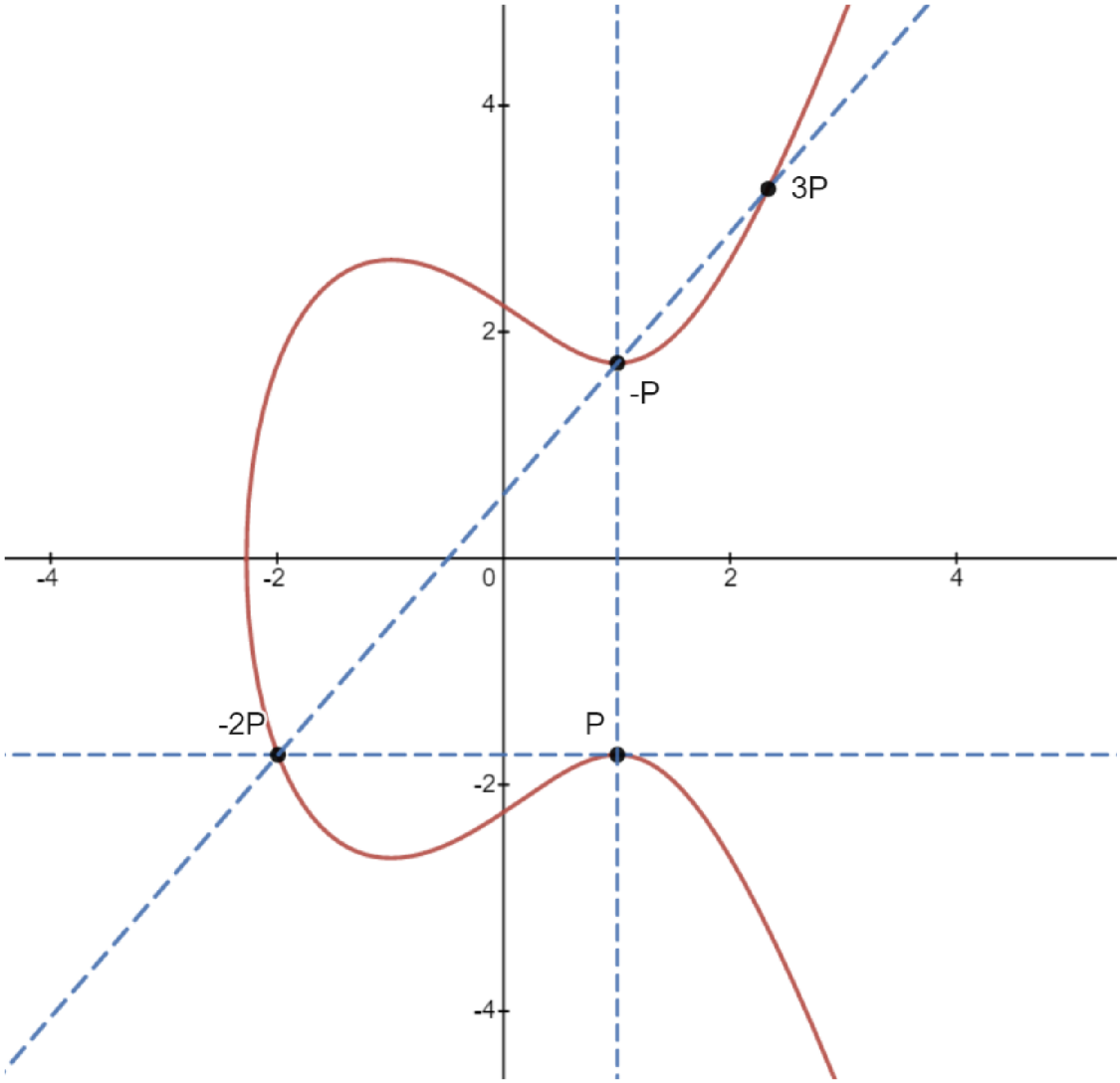
Tento princip se také dočkal zlepšení v roce 1985, kdy Neal Koblitz a Viktor Saul Miller nezávisle na sobě představili ECC (Elliptic curve cryptography), tedy kryptografii na eliptických křivkách [14]. Užití eliptických křivek také značně snížilo požadavky na šifrovací zařízení, jelikož z bezpečnostního hlediska je podle NISTu [15] síla šifry na bázi eliptických křivek s klíčem o délce 256 bitů ekvivalentní s šifrou RSA s klíčem o délce 2048 bitů [6].

Při užití je definována eliptická křivka ve tvaru $y^2 = x^3 + ax + b$ se systémem sčítání jejich bodů. Ke sčítání dochází pomocí sečny grafu. Příslušná přímka pak určuje, že součet všech jí protnutých bodů je roven nule. Součet bodu sama se sebou lze provést pomocí tečny v daném bodě. Z těchto vlastností je odvozeno skalární násobení $Q = nP$, kde je zjištění velkého n problém diskrétního logaritmu na eliptických křivkách [14]. Nalezení bodu Q za neznámého n nakonec vyžaduje

[5] Jedná se o složitost $L_n[\frac{1}{2}, 1 + o(1)]$, kterou lze vyjádřit jako: $e^{(1+o(1))\sqrt{(log(n))(log(log(n)))}}$

[6] Stále se jedná o rozdílné algoritmy, které nelze volně zaměnit

náhodné hledání všech možných n, jelikož pozice bodů jsou dosti neintuitivní, tak jak si můžeme prohlédnout pro určený bod 3P.



Obr. 2.1: Určení bodu 3P na křivce $y^2 = x^3 - 3x + 5$

3 Kryptosystémy na integrovaných obvodech

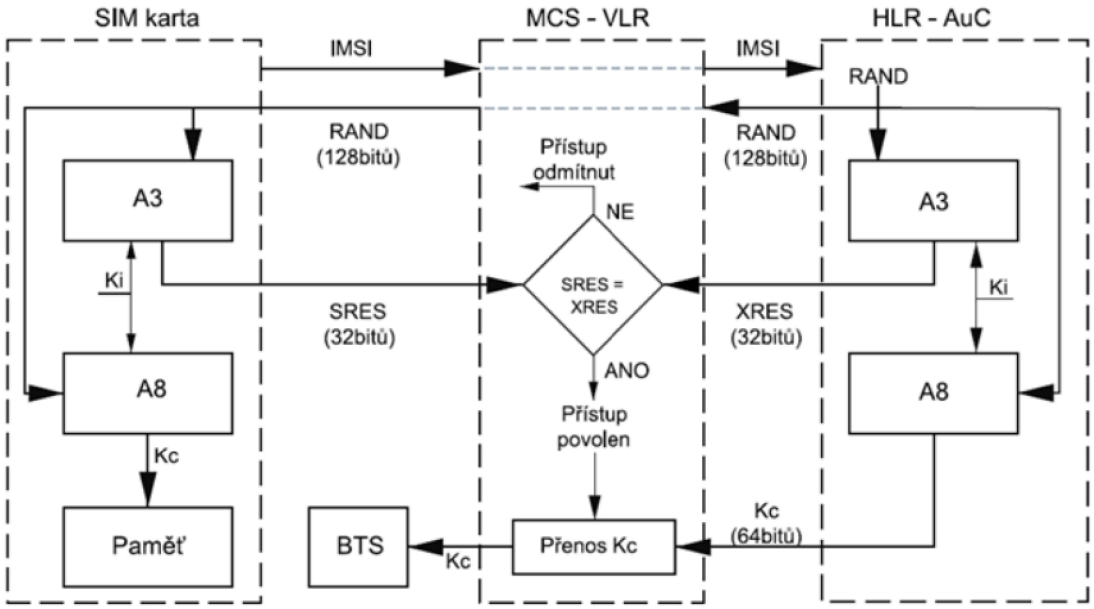
V této kapitole se zaměřím na příklady praktického užití kryptografie realizované pomocí dedikovaného hardware. Takový způsob aplikace umožňuje zajistit dostatečnou míru zabezpečení i s pomocí relativně drobných zařízení. Zdrojem pro tuto kapitolu jsou přednášky docenta V. Zemana v rámci předmětu Aplikovaná kryptografie.

3.1 SIM karty

První obecné normy pro SIM (Subscriber Identity Module) [1] karty uvedl ETSI (European Telecommunications Standards Institute) [2] ve své specifikaci TS 11.11 v roce 1996. Cílem zmíněné normy bylo definovat GSM (Global System for Mobile Communications) [3], telekomunikační síť později známou pod obchodní značnou 2G.

Pro užití v 2G sítích je na každé SIM kartě trvale uloženo číslo IMSI (International Mobile Subscriber Identity) [4] a důvěrný předsdílený 128 bitový klíč Ki. S pomocí tohoto klíče se na kartě provádí algoritmy A3 a A8 pro autentizaci generuje klíč Kc pro šifrování algoritmem A5. V praxi pro autentizaci zařízení přes VLR (Visitor Locator Register) [5] zašle své IMSI svému HLR (Home Local Register) [6], kterému VLR důvěřuje. HLR zná klíč Ki daného uživatele a vytvoří pro něj výzvu v podobě náhodného vstupu 128 bitů pro algoritmy A3 a A8. Jak HLR, tak samotná SIM karta spočtou dočasný ověřovací klíč pomocí A3 stanici VLR, kde dojde k autentizaci a v případě povolení je následná komunikace šifrována pomocí A5 s klíčem Kc, který byl vygenerován algoritmem A8 jak na straně uživatele, tak na straně HLR a poslán na VLR. Celý systém je popsán v následujícím schématu[7].

[1] Přeneseně uživatelský identifikační modul
[2] Evropský ústav pro telekomunikační normy
[3] V překladu Globální systém pro mobilní komunikace
[4] Mezinárodní identifikační kód uživatele
[5] V podstatě místní ověřovací stanici
[6] Domácí ověřovací stanice
[7] Schéma pochází z přednášky docenta V. Zemana



Obr. 3.1: Schéma autentizace v kryptosystému v sítích GMS

3.2 EMV platební karty

Častým praktickým užitím kryptografie na integrovaných obvodech je také systém elektronických plateb pomocí debetních, kreditních, či charge karet. Celý kryptosystém zaměřený na bezpečné ověřování plateb se nazývá 3D Secure a spočívá v ověřování na doménách plátce, prostředníka a nabyvatele. Samotné karty pak nesou EMV čipy standardizované jejich výrobci Europay, Mastercard a Visa podle nichž jsou pojmenovány a jsou odvozeny z standardu ISO/IEC 7816 pro elektronické identifikační karty.

3.3 FPGA v kryptografii

FPGA (Field Programmable Gate Array) neboli programovatelné hradlové pole je integrovaný obvod, pro který je typická jeho snadná rekonfigurace. Při každém jeho vypnutí dojde ke ztrátě dosavadní konfigurace a při příštím startu je potřeba jej se nechat nakonfigurovat z běžně připojené konfigurační paměti [19]. Díky tomu, že se tento výkoný integrovaný obvod snadno rekofiguruje, našel si uplatnění v kryptografii, kde je potřebo provádět časté bezpečnostní aktualizace pro plnění současných standardů [20]. Praktickým příkladem takového šifrování je užití na síťových prvcích, kde jsou značné nároky na rychlost zpracování dat při šifrování důvěrných informací,

kterými můžou být kupříkladu logy o provozu. Takovou implementaci pro zpracování šifrování pomocí FPGA užívá na některých svých zařízeních i nejrozšířenější výrobce síťových prvků CISCO. [8]

[8]Návod na konfiguraci profilu FPGA na CISCO zařízeních dostupný z archivu na: https://webcache.googleusercontent.com/search?q=cache:sd_wNnoLl7sJ:https://www.cisco.com/c/en/us/td/docs/switches/lan/cisco_ie3X00/software/17_4/b_system-management-ie3x00/m_configuring-fpga-profile.pdf&cd=3&hl=cs&ct=clnk&gl=cz

4 Jazyky pro popis hardware

S technologickým pokrokem postupně dochází ke zmenšování výpočetních zařízení a to se zachováním stejného výkonu. Důležitým krokem v tomto vývojovém procesu se stal vývin tzv. HDL (Hardware Description Language) [1]. Jejich vývin začal již v šedesátých letech minulého století. Nelze s jistotou říci, který HDL byl první, ale jakožto významné se uvádí CANCER (Computer Analysis of Non-linear Circuits, Excluding Radiation) představený v roce 1971 [21] a především z něho odvozený SPICE (Simulation Program for Integrated Circuit Emphasis) [22], vydaný o dva roky později. V dnešní době dominantními HDL jsou jazyky **Verilog** a **VHDL**

4.1 Porovnání VHDL a Verilogu

Vývin obou jazyků počal již v roce 1983. Verilog byl vyvinut společností Automated Integrated Design Systems a VHDL vznikl v rámci projektu VHSIC (Very High-Speed Integrated Circuits Program) pod záštitou Ministerstva obrany Spojených států amerických [19]. Z hlediska četnosti užití v praxi došlo ke geografickému rozdělení, kdy Verilog se stal populárnějším v Severní Americe, zatímco VHDL se těšil popularity spíše v Evropě [1]. V dnešní době se ovšem toto rozdělení smývá a záleží spíše na zkušenostech zaměstnanců a obecné preferenci v příslušné korporaci [19].

Z hlediska složitosti bývá Verilog označován za lehčí k ovládnutí a je přirovnáván ke konvenčnímu programovacímu jazyku C. Oproti tomu VHDL disponuje třídním programováním a ve stejné projekci si ho tedy můžeme představit jako jazyk Java [19].

Jelikož oba jazyky usilují o řešení téhož problému, tedy popisu integrovaného obvodu za účelem jeho výroby, jsou v dnešní době požadavky na trhu práce omezovány na zkušenosti s některým z HDL nehledě na preferenci společnosti.

4.2 Jazyk VHDL

Jazyk VHDL se dočkal své IEEE standardizace v roce 1987 a byl dále revidován o deset let později [23]. Jazyk umožňuje popsat návrh integrovaného obvodu do jednotlivých bloků a ty pak navázat s pomocí signálů. Skutečný vzniklý obvod pak svými kvalitami dosti závisí na použitém syntetizátoru, který se snaží vytvořený návrh co nejlépe zkompilovat pro implementaci na samotný hardware. VHDL také poskytuje nesyntetizovatelné příkazy, které jsou určeny pro simulaci ve virtuálním prostředí, kde lze takto například simulovat vady součástek či prostředí [1].

[1] Jazyk pro popis hardware

Co se týče pravidel syntaxe, je VHDL case sensitive, komentáře se značí dvěma pomlčkami a řádky jednotlivých příkazů se ukončují středníkem. Pojmenovávání identifikátorů má také svá pravidla, kdy jsou povolena pouze malá a velká písmena, číslice, a podtržítka. Každý identifikátor musí začínat písmenem, nesmí končit podtržítkem a nesmí se v něm nacházet dvě podtržítka za sebou [1].

Při psaní popisu jakéhokoliv obvodu za pomoci VHDL se nejprve vypíší použité knihovny, jako může být například některá ze standardních knihoven IEEE [23]. Každý vytvořený popis v jazyce VHDL vyžaduje definovat dva typy jednotek:

Entitu, kde jsou popsány její generické konstanty a její vstupní a výstupní porty spolu s jejich datovými typy. Generické konstanty lze pak měnit pomocí generického mapu a zvenku tak parametrizovat entitu [1]

Architekturu, která je přiřazena příslušné entitě a popisuje své operace na předložené entitě. Každá entita může mít vícero architektur, tedy vícero možných užití dané entity, ale architektura může být přiřazen pouze jedné entitě. Pro každou architekturu je po klíčovém slovu **of** uvedena jí přiřazená entita. Následně je uvedena dodatečná deklarace pro tuto architekturu a mezi klíčovými slovy **begin** a **end** jsou vypsány samotné operace, které probíhají paralelně. Z tohoto důvodu dvojité přiřazení signálu do jednoho výstupu vyvolá chybu, jelikož při paralelním zpracovávání dojde ke kolizi [23].

Základními rozlišovanými objekty jazyka jsou **konstanta**, **signál**, **soubor** a speciálně **alias**, který pouze zastupuje jiný objekt pro zpřehlednění kódu [1].

Co se používaných datových typů týče, jazyk VHDL umožňuje typ **výčtový**, **celočíselný**, **fyzický**, **typ s plovoucí řádovou čárkou**, **pole**, **záznam** a zvláštní typ **soubor** [1].

5 Praktické řešení problematiky

V této části se budu věnovat praktickému využití sesbíraných poznatků za účelem vytvoření generátoru kryptograficky bezpečných prvočísel na platformě FPGA. Práce je realizována s pomocí softwaru Vivado, určeným k návrhu, syntéze a implementaci hardwarových návrhů na desky od společnosti Xilinx.

5.1 Určení vhodného testu prvočíselnosti

Důležitým prvním krokem je určení tíženého testu prvočíselnosti, který splňuje kladené požadavky. Použité testované vstupní číslo pak bude výsledkem náhodného generátoru čísel.

Trial division

Nejjednodušší test z pohledu laika *Trial division* by měl ve světě kryptografie, kde dle NISTu (National Institute of Standards and Technology) jsou pro RSA požadovány klíče o velikosti 2048 bitů [15], což vyžaduje generování dvou prvočísel o minimálně 1024 bitech, značné potíže. Pro test bychom museli vyzkoušet číslo dělit prvočísly menšími než $\sqrt{2^{1024}}$, tedy 2^{512}. Množství takových prvočísel můžeme aproximovat pomocí $f(x) : \frac{x}{\log x}$ a dostaneme se k počtu $3,778 \cdot 10^{151}$ prvočísel. První objevené prvočíslo tak velké, že bychom ho do tohoto seznamu nepotřebovali, by bylo 13. Mersennovo prvočíslo, které v roce 1952 objevil americký matematik Raphael M. Robinson na platformě SWAC (Standards Western Automatic Computer) [1]. Nepochybně jsou tu zastoupena i prvočísla, která nikdo nikdy nezapsal a nejsou nám tedy známa. Nadále pokud nepoužijeme úsporné kódování a zapíšeme si oněch $3,778 \cdot 10^{151}$ prvočísel s bitovou délkou 512 bitů na své úložiště, zabere nám to $2,2 \cdot 10^{141}$ TB dat. V roce 2020 byl odhadovaný počet uložených dat na světě $59 \cdot 10^9$ TB [2]. Z toho lze vypočítat, že by nám pro naši implementaci algoritmu *Trial division* stačil úložný prostor jen $3,727 \cdot 10^{130}$-krát větší než objem všech uložených dat v roce 2020. Krom zmíněných komplikací je celý algoritmus značně výpočetně neefektivní pro ověřování větších čísel a tedy v závěru můžeme jeho implementaci zamítnout.

[1] Seznam objevitelů Mersennových prvočísel dostupný na: https://www.mersenne.org/primes/

[2] Dle článku dostupného na: https://theconversation.com/the-worlds-data-explained-how-much-were-producing-and-where-its-all-stored-159964

Lucas-Lehmerův test

Jelikož v našem případě hodláme generovat kryptograficky bezpečná prvočísla, nepřipadá použití Lucas-Lehmerova testu v úvahu, protože by značně omezil množinu prvočísel, které jsme schopni vygenerovat a oslabil tak tvořený kryptosystém. Generování prvočísel tímto testem má však stále využití [3] a to při hledání velkých prvočísel pomocí dříve zmíněného nástroje GIMPS.

Fermatův test

Fermatův test je pravděpodobnostním testem, který s každým úspěšným průběhem snižuje pravděpodobnost chyby na polovinu. V jeho případě tedy po úspěšném průchodu například čtyř cyklů si je test jistý na 93,75% , že se jedná o prvočíslo. Achilovou patou tohoto testu jsou ovšem Carmichaelova čísla, která i po libovolném počtu cyklů chybně označí číslo za prvočíslo. V kryptografické praxi by se tak tento nedostatek mohl stát kritickou zranitelností, pokud by útočník nejprve hledal použití některého z těchto složených čísel na místo prvočísla. Samotné použití složeného čísla příslušný kryptografický algoritmus přinejmenším kompromituje a pokud je také omezena jejich množina na Carmichaelova čísla, lze se na ně přesněji zacílit a zefektivnit tak potenciální útok. Z těchto důvodů není Fermatův test ideálním řešením, ovšem je již řešením prakticky aplikovatelným.

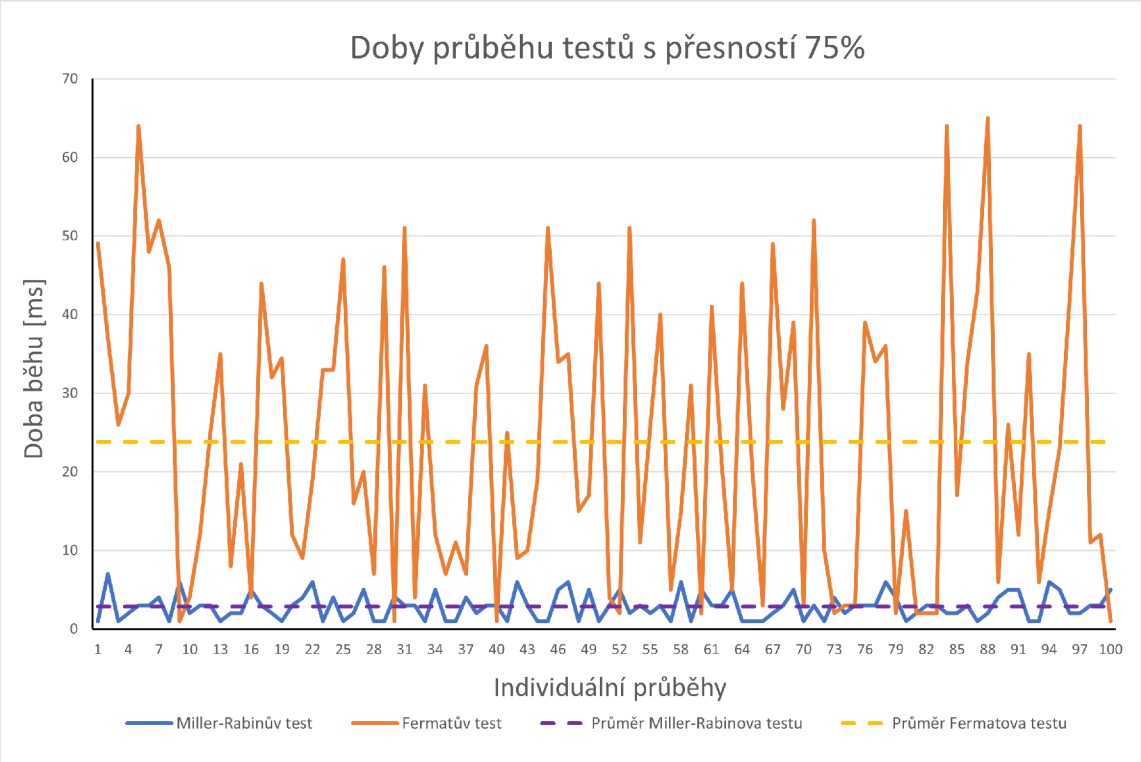
Miller-Rabinův test

V praxi zdaleka nejpoužívatelnějším je Miller-Rabinův test [7]. Tato jeho samotná širší aplikace nepřímo svědčí o jeho příhodnosti pro užití v kryptografii. Je podobně jak Fermatův test testem pravděpodobnostním a jeho přesnost závisí na počtu proběhlých cyklů. Pravděpodobnost chyby se s každým průběhem snižuje na čtvrtinu a tak po čtyřech úspěšných cyklech si je test prvočíselností čísla jist na 99,61%. Oproti Fermatovu testu také Miller-Rabinův test při správné implementaci, kdy jsou pokaždé volena nová náhodná a, nenese potenciálně zneužitelnou kritickou zranitelnost. Pokud by se totiž nastavila statická a např. 2, 3, 4 a 5, tak test trpí podobným jevem ke Carmichaelovým číslům u Fermatova testu, jelikož zůstane konstantní skupina 0,39% složených čísel, která vždy takovou konfigurací testu projdou. Miller-Rabinův test je sice nejobtížnějším na implementaci, ale přináší nejlepší výsledky.

[3] Jedná se spíše o "senzaci"

5.2 Model generování prvočísel v Pythonu

V programovacím jazyce Python jsem vytvořil modely aplikace na generování prvočísel podle Frematova a Miller-Rabinova testu. Jelikož tento přístup není výpočetně optimální, generuji pouze dvacetibitová prvočísla, testuji je jenom na míře pravděpodobnosti 75% [4] a je zde i zavedena zranitelnost s pevně určenými čísly a [5] pro urychlení výpočtu. Doby průběhu pro porovnání efektivity využití jednotlivých algoritmů ve 100 průbězích jsou zaneseny do následujícího grafu.



Obr. 5.1: Průběhy testů

Každý průběh určil s pravděpodobností 75% dvacetibitové prvočíslo, což s použitím Fermatova testu zabralo v průměru 23,86 ms, zatímco v případě implementace Miller-Rabinova testu pouhých 2,86 ms. Tedy v průměru bylo na modelu hledání prvočísel s pomocí Miller-Rabinova testu více než osmkrát rychlejší. Součástí přílohy jsou zdrojové kódy použitých testů.

[4] Dva průběhy pro Frematův test a jeden pro Miller-Rabinův.
[5] Nejsou generována náhodně

Nedokonalost modelu

Tento test nebere v potaz efekt zlepšení s pomocí paralelizace výpočtů, při které může běžet více cyklů daných testů zároveň. V našem případě by probíhaly oba cykly Fermatova testu současně, což by čas na prozkoušení jednoho čísla mohlo snížit při této pevné konfiguraci až na polovinu. Pro Miller-Rabinův test je to ještě výraznějším zlepšením, jelikož může probíhat pro všechna možná r současně a tedy bude mít stále lepší výsledky. Pro oba testy při paralelizaci také platí, že jimi může být zkoušeno více prvočísel současně.

5.3 Definice kryptograficky-bezpečných prvočísel pro potřeby práce

Samotná problematika definice spojení kryptograficky-bezpečné prvočíslo byla popsána v kapitole 2.1. My se nadále zaměříme na podobu dnes v kryptografické praxi užívaných prvočísel a jejich požadované specifikace s přihlédnutím k příslušným odůvodněním pro tyto požadavky.

Platné standardy

Pro tuto kapitolu jsem se podíval na současná doporučení Evropské agentury pro bezpečnost sítí a informací ENISA [24] a stále platná doporučení FIPS 186-4 Digital Signature Standard od amerického NISTu (National Institute of Standards and Technology) [25]. Jelikož FIPS 186-4 má již od 3. února 2023 [6] vydaného nástupce FIPS 186-5 [26], vzal jsem jej také v potaz pro otázku budoucnosti praktičnosti prezentovaného řešení. Doporuření z publikace FIPS 186-4 jsou ovšem stále platná a to až do 3. února 2024.

5.3.1 Generovaná prvočísla dle jejich užití

Jak již vyplynulo z mnoha popsaných kryptografických algoritmů, užití prvočísel je velice častým a nedílným prvkem asymetrické kryptografie. Nejčastější užité matematické problémy jsou:

- **Faktorizace na prvočísla** spočívající v obtížnosti rozkladu složeného čísla **n** na součin prvočísel **p**, **q**, či případně dalších.

$$n = p * q \dots \tag{5.1}$$

[6] Rok psaní této práce

- **Problém diskrétního logaritmu**, který se opírá o náročnost nalezení exponentu **x** v cyklické grupě modulo **n**, kde za **n** bývá vybíráno **prvočíslo**.

$$X = g^x mod(n) \tag{5.2}$$

Vlastnosti těchto matematických problémů byly popsány v kapitole **2 Užití prvočísel v kryptografii**.

Faktorizace na prvočísla

Pro **generování prvočísel užitých v šifrách typu RSA**, které závisí na problému faktorizace, nabízí NIST dva přístupy generování [25][7]. Pro prvočísla o velikosti 1024 a více bitů [8] postačí pouze výstup náhodného generátoru čísel příslušné délky, co prošel testem prvočíselnosti s požadovanou mírou přesnosti. Druhý přístup a jediný přípustný pro generování prvočísel pro toto užití s bitovou délkou 512 [25][9] je prvotní tvorba malých pomocných prvočísel p_1 a p_2 pro nalezení prvočísla **p**, kde $p-1$ a $p+1$ jsou násobky pomocných prvočísel p_1 a p_2 [10]

Důvodem pro tyto dva přístupy není nic jiného než ochrana proti Pollardovu n-1, případně Williamsova n+1 , algoritmu. Ovšem již v roce 1999 v publikaci "Are 'Strong' Primes Needed for RSA?"[16] Ronald L. Rivest společně s Robertem D. Silvermanem budoucnost praktické užitečnosti onoho druhého složitějšího způsobu generování prvočísel pro RSA zpochybnili. V případě větších čísel, kde jsou povoleny oba způsoby generování, se značnou mírou pravděpodobnosti budou mít čísla $p+1$ i $p-1$ natolik velké své největší faktory, že čísla generovaná prvním způsobem budou splňovat stejné požadavky, co čísla generovaná druhým způsobem. Hraničním stanovením těchto požadavků je bitová délka případných pomocných prvočísel $p-1$ a $p+1$. Ta je určena tak, aby pro případného útočníka se značným, ale nikoliv příliš předimenzovaným, výpočetním výkonem byla faktorizace složeného čísla n s pomocí Lenstrovy metody s využitím eliptických křivek snažší než s využitím Pollardova, popřípadě Williamsova algoritmu [15].

Vedle Lenstrovy metody pro faktorizaci jsou také popsány dvě výpočetně efektivnější metody a to Pomeranceho metoda kvadratického síta a Pollardova obecná metoda síta číselného pole [27]. Tento technologický pokrok dělá bezpečnostní požadavek na velké faktory čísel $p-1$ a $p+1$ natolik redundantní, že se dá připodobnit k instalaci petlicového zámku na prosklené zahradní dveře v době, kdy každý zloděj disponuje kamenem. Cílené plnění tohoto požadavku u velkých čísel tedy pouze komplikuje jejich hledání a nepřináší žádné další bezpečnostní výhody[16].

[7] FIPS 186-4, kapitola B.3.1 Criteria for IFC Key Pairs, strana 51

[8] FIPS 186-5 zvažuje standardizované délky 1024 bitů, 1536 bitů a 2048 bitů[26]

[9] FIPS 186-5 již tak malou bitovou délku nezvažuje

[10] Příslušný algoritmus hledání popisuje FIPS 186-4 v apendixu C.9 [25]

Požadavky na počty kladných průběhů[11] Miller-Rabinova testu při generování těchto prvočísel podle FIPS 186-4[25] a FIPS 186-5[26] jsou popsány v následující tabulce spolu s přípustnou mírou chybovosti, na kterou daný počet průběhů cílí.

Požadavky na M-R testování prvočísel pro šifry typu RSA podle FIPS 186-4 a FIPS 186-5				
Dosažená chybovost	Velikost pomocných prvočísel $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{q}_1$ a $\mathbf{q}_2$	Počet kol M-R testů pro pomocná prvočísla	Velikost prvočísel **p** a **q**	Počet kol M-R testů pro prvočísla **p** a **q**
2^{-80}	$>$ 100 bitů	28 kol	512 bitů	5 kol
2^{-112}	$>$ 140 bitů	38 kol	1024 bitů	5 kol
2^{-128}	$>$ 170 bitů	41 kol	1536 bitů	4 kola
2^{-144}	$>$ 200 bitů	44 kol	2048 bitů	4 kola

Tab. 5.1: Požadavky na M-R testování pro užití problému faktorizace na prvočísla

K tabulce je vhodné zmínit, že její první řádek již není standardem pro FIPS 186-5 a od 3.února 2024 nebudou tato prvočísla doporučována za bezpečná k užití pro tyto šifry. První řádek je také jediný, kde je požadováno přistupovat pouze s využitím pomocných prvočísel a nelze tak **p** a **q** generovat napřímo.

Diskrétní logaritmus

Pokud se zaměříme na generování prvočísel pro potřeby algoritmů založených na principu diskrétního logaritmu, musíme rozlišit dva možné typy jeho implementace. Především je známá jeho klasická podobu ve tvaru $g^x mod(p) = X$ užitá v algoritmech jako je Diffie-Hellman protokol, či šifra ElGamal v běžném režimu. Pro tuto aplikaci je potřeba generovat prvočíslo **q**, řád užívaného generátoru **g**. S pomocí **q** je následně hledáno prvočíslo **p**, tedy modulus z užívané rovnice, které je jen o 1 větší než násobek čísla **q**[25]. Je důležité zmínit, že FIPS 186-5 již tento způsob užití problému diskrétního logaritmu nedoporučuje [26].

Následující tabulka znázorňuje doporučované velikosti pro prvočísla **p** a **q** s příslušným počtem požadovaných proběhlých rund Miller-Rabinova testu při jejich hledání podle doporučení FIPS 186-4 [25].

[11] Myšleno průběhy, co číslo nevyřadí za složené, vzhledem k tomu, že M-R test je spíše testem složenosti čísla

Požadavky na M-R testování prvočísel pro algoritmy založené na problému diskrétního logaritmu podle FIPS 186-4		
Velikost prvočísla **p**	Velikost prvočísla **q**	Počet kol M-R testů pro prvočísla **p** a **q**
1024 bitů	160 bitů	40 kol
2048 bitů	224 bitů	56 kol
2048 bitů	256 bitů	56 kol
3072 bitů	256 bitů	64 kol

Tab. 5.2: Požadavky na M-R testování pro užití problému diskrétního logaritmu

Alternativou k tomuto způsobu implementace je užití eliptických křivek a problém představit jakožto násobení bodu na eliptické křivce v podobě $P = a * G$. V této rovnici jsou **P** a **G** body na eliptické křivce. Řád křivky je pak určen jakožto $q = n * h$, kde **n** je prvočíslo požadovaných rozměrů a **h** je kofaktor, libovolně zvolené číslo z požadovaného rozsahu. Následně zvolený bod **G** je na této křivce řádu **n**. Některé křivky užívají za kofaktor **h** číslo 1 a jsou proto nazývány křivky na prvočíselných polích. Pro takové křivky jsou vyšší velikostní požadavky na bitovou délku prvočísla **n** [25].

Jelikož se v zásadě jedná o stále stejný matematický problém o stejné míře bezpečnosti a tedy i přijatelné chybovosti při implementaci, jsou počty doporučovaných kol testů při generování čísla **n** stejná, co při generování čísla **q** u klasického užití diskrétního logaritmu [25].

Můžeme se spíše v následující tabulce zaměřit na doporučené velikosti kofaktoru **h** pro příslušné užívané skupiny délek prvočísla **n** [12].

Požadavky na velikost kofaktoru h pro příslušná n podle FIPS 186-4	
Velikost prvočísla **n**	Velikost kofaktoru **h**
160-223 bitů	10 bitů
224-255 bitů	14 bitů
256-383 bitů	16 bitů
384-511 bitů	24 bitů
$\geq$ 512 bitů	32 bitů

Tab. 5.3: Požadavky na kofaktor h pro prvočísla n

[12]Skupinu 160-223 bitů již do budoucna norma FIPS 186-5 [26] nedoporučuje.

5.3.2 Určení parametrů generátoru prvočísel v praktické části

Jak bylo v této kapitole představeno, prvočísla se těší v kryptografii širšímu užití a pro potřebu tvorby generátoru kryptograficky-bezpečných prvočísel si musíme vybrat příslušné doporučení, které hodláme splnit.

Pro potřebu této bakalářské práce jsem se rozhodl pro generování 160 bitových prvočísel s užitím 40 kol Miller-Rabinova testu. Pokud opomeneme doplňková prvočísla **p1**, **p2**, **q1** a **q2** užívaná při tvorbě prvočísel pro šifry založené na problému **faktorizace velkých čísel** pro zaměření jen proti některým faktorizačním algoritmům, jelikož samy potřebují doplnit o generátor/test pro větší prvočísla **p** a **q**, je 160 bitů nejmenší stále užívaná délka prvočísel pro kryptografii. Může se jednat buď o řád generátoru **g** v klasickém pojetí problému diskrétního logaritmu, či o řád bodu **G** v jeho verzi založené na eliptických křivkách.

Takto malá velikost nebude již za rok doporučována k užití, ale současné standardy splňuje. Vybral jsem si ji pro snazší a rychlejší zpracovávání při měření a testování navrhovaného obvodu. Už takto velké číslo nese při práci ve VHDL komplikace spjaté právě s jeho velikostí. Případná úprava algoritmu pro větší čísla s jiným počtem testů spočívá už pouze ve škálování vytvořeného designu [13]

5.4 Implementované algoritmy

Tato sekce bude individuálně rozebírat jednotlivé matematické prvky a principy obsažené ve finálním návrhu a představovat jejich reprezentaci ve VHDL.

Kostrou celého designu, jak již bylo výše popsáno, je Miller-Rabinův test prvočíselnosti pro 160 bitová čísla ověřovaná jeho čtyřiceti průběhy.

5.4.1 Inicializace vstupních hodnot testu

Vstupními parametry užitého testu jsou vyjma přímo testovaného čísla také použití svědci pro jednotlivé průběhy. Jelikož hledáme prvočíslo o nezkrátitelné délce 160 bitů, musí být oba krajní bity nastaveny na jedna, protože automaticky odmítáme menší čísla a testovat sudá čísla na prvočíselnost pro nás nemá smysl. Co se výběru svědků týče, tak požadujeme čtyřicet různých čísel větších než 1, která se budou v různých průbězích obměňovat, aby test netrpěl vadou v podobě obdoby Carmichaelových čísel u Fermatova testu, která testem chybně prochází [14].

[13] Krom navýšení všech příslušných konstant to obnáší i přepsání užívané násobičky, jež je popsána v kapitole 5.4.4

[14] S konstantními svědky by bylo možné snáze sestavit tabulku chybně vyhodnocovaných čísel.

V návrhu je užito vstupního portu nesoucího 160 bitů, ze kterého jsou první dva bity užity na náhodné posunutí jinak konstantní užívané skupiny svědků a zbylých 158 bitů je vsazeno mezi dvě jedničky pro vygenerování testovaného čísla.

5.4.2 Problém modulárního umocňování na velká čísla

Podstatou Miller-Rabinova testu je ověřování platnosti rovnic $a^{p-1} \equiv 1 \; mod(p)$ a $a^{2^{s-1}d} \equiv \pm 1 \; mod(p)$ blíže popsaných dříve v kapitole 1.3.1. Číslo **p**, ze kterého odvozujeme pro tyto rovnice exponent, je v našem případě značně velké číslo. Klasický přístup k umocňování v podobě rozložení na součin $a^x = a_1 \cdot a_2 \cdot \cdots \cdot a_x$, kde $a = a_1 = a_2 = \cdots = a_x$ není výpočetně efektivní a značně by zpomalil až svými nároky znemožnil reálný průběh algoritmu i za předpokladu, že by byl každý individuální mezivýsledek pro zjednodušení příštích výpočtů modulován.

Celou operaci lze zefektivnit algoritmem "Square and Multiply", česky nazývaným "Algoritmus binárního umocňování". Spočívá v možnosti pokrácení sudého exponentu dvojkou při umocnění základu na druhou. Lichý exponent lze případně snížit o jedna na sudý pomocí vytknutí násobení jeho základem. Výpočet 3^{14} lze následně upravit takto:

$$3^{14} = 9^7 = 9 \cdot 9^6 = 9 \cdot 81^3 = 9 \cdot 81 \cdot 81^2 = 9 \cdot 81 \cdot 6561 \tag{5.3}$$

Pokud zároveň i pracujeme v modulární algebře např mod(7), lze jednotlivé základy a součiny individuálně modulovat:

$$3^{14} \equiv 2^7 \equiv 2 \cdot 2^6 \equiv 2 \cdot 4^3 \equiv 2 \cdot 4 \cdot 4^2 \equiv 2 \cdot 4 \cdot 1 \equiv 2 \; mod(7) \tag{5.4}$$

V případě individuálního násobení třemi je potřeba provézt jednoznačně více multiplikativních operací:

$$\begin{aligned} 3^{14} \equiv 3 \cdot 3^{13} \equiv 2 \cdot 3^{12} \equiv 6 \cdot 3^{11} \equiv 4 \cdot 3^{10} \equiv 5 \cdot 3^9 \equiv 1 \cdot 3^8 \equiv 3 \cdot 3^7 \\ \equiv 2 \cdot 3^6 \equiv 6 \cdot 3^5 \equiv 4 \cdot 3^4 \equiv 5 \cdot 3^3 \equiv 1 \cdot 3^2 \equiv 3 \cdot 3 \equiv 2 \; mod(7) \end{aligned} \tag{5.5}$$

Při binárním zápisu exponentu jej lze podělit dvěma pomocí pouhého bitového posunu a dělitelnost také posoudit podle posledního bitu. Díky této vlastnosti lze algoritmus snadno zapsat pomocí následujícího kódu [15], kde číslo **c** je modulárně umocněno na 1024 bitový exponent **d** s modulem **m**.

[15] Jedná se o pro větší m nesyntetizovatelný kód pro potřeby simulátoru.

Výpis 5.1: Modulární umocňování ve vzestupném režimu

```
1    ModExp(c,d):
2        resu := to_unsigned(1,1024);
3        for i in 0 to 1023 loop
4            if d(i) = '1' then
5                resu := resu*c mod m;
6            end if;
7            c:=c*c mod m;
8        end loop;
9        return resu;
```

Algoritmus je možné upravit pro inverzní průchod čísla **d** tak, aby předčasně vzaté výsledky představovaly hodnoty $a^{2^{s-1}d}$ užité v Miller-Rabinově testu. Tato pro nás příznivější implementace modulárního binárního umocňování by v obdobném zápisu vypadala takto:

Výpis 5.2: Modulární umocňování v sestupném režimu

```
1    InvModExp(c,d):
2        resu := to_unsigned(1,1024);
3        for i in 1023 downto 0 loop
4            resu := resu*resu mod m;
5            if d(i) = '1' then
6                resu := resu*c mod m;
7            end if;
8        end loop;
9        return resu;
```

5.4.3 Montgomeryho forma, násobení a umocňování

Jak je z testu patrné, exponent není jedinou pozicí, co zabírá číslo velikostně odvozené od testovaného. V našem případě tak délky 160 bitů dosahuje v rovnicích $a^{p-1} \equiv 1\ mod(p)$ a $a^{2^{s-1}d} \equiv \pm 1\ mod(p)$ také užitý modulus **p**, který je roven našemu testovanému číslu. Doporučená literatura udává, že operátory mod a dělení jsou ve VHDL definovány pouze pro celočíselné typy a největším v naší implementaci je integer, který má pouze 32 bitů [1]. V našem případě tedy nelze takto modulovat 160 bitovým číslem. Vivado je ovšem také schopno syntetizovat modulování a dělení libovolnou mocninou čísla dva, jelikož v bitovém zápisu lze tyto operace provádět

pouhým bitovým posuvem [16]. Tento způsob dělení a modulování si můžeme znázornit následujícím příkladem:

$$\begin{aligned} 10011101101001 \; mod \; 10000000 &= \mathbf{1101001} \\ \mathbf{1001110}1101001 \; \; div \; \; 10000000 &= \mathbf{1001110} \end{aligned} \tag{5.6}$$

Tuto vlastnost můžeme využít při implementaci **Montgomeryho násobení**, které v dubnu 1985 představil Peter L. Montgomery ve svém článku "Modular Multiplication Without Trial Division"[28]. Tato metoda umožňuje modulární násobení s modulem **n** převést do upravené algebry s modulem a dělením pouze zvoleným číslem **r**. Pro volbu čísla **r** musí být splněny následující dvě podmínky.

- $\mathbf{r} > \mathbf{n}$
- $GCD(\mathbf{r}, \mathbf{n}) = 1$, neboli jsou čísla nesoudělná

A právě libovolné mocniny dvou jsou vždy nesoudělné s testovanými lichými čísly. Proto je u tohoto generátoru 160 bitových prvočísel zvolena za **r** nejmenší vhodná mocnina 2 a to 2^{160} [17]

Určení parametrů pro Montgomeryho formu

Každé číslo **x**, lze převézt na jeho Montgomeryho formu $\bar{\mathbf{x}}$ jako [29]:

$$\bar{x} = x \cdot r \; mod(n) \tag{5.7}$$

Tento hned úvodní krok nás opět odkazuje na modulování velkým **n**. To však s tím rozdílem, že je jej nutno užít pouze pro **x** volená za svědky na začátku procesu umocňování. Pro užité svědky není požadováno velikostní omezení, jedná se pouze o různá celá čísla větší než 1, která jsou volena náhodně. Skupina svědků může být tedy velikostně omezena a tento převod na montgomeryho formu uskutečněn v rámci loopu, kde od součinu $x \cdot (r - n)$ stačí až maximálně $(x_{max} - 1)$-krát odečíst číslo n. Reprezentace tohoto modulování, kde **wit** je náš svědek v základní formě s $wit_{max} = 44$ pak může vypadat takto:

[16] Jedná se o celočíselné dělení, případně dělení se zbytkem.

[17] V bitovém zápisu se jedná o jedničku a 160 nul.

Výpis 5.3: Převod malých svědků do Montgomeryho formy

```
1    SmallMod(wit, n):
2        x := wit*(r-n);
3        for i in 0 to 42 loop
4            if x < n then
5                return x;
6            end if;
7            x := x-n;
8        end loop;
9        return x;
```

Pro práci s čísly v Montgomeryho formě je potřeba také určit inverzní prvek k **n** v algebře modulo **r**. Lze tak docílit nalezením Bézoutovy rovnosti v podobě [28]:

$$r \cdot R' - n \cdot N' = 1 \tag{5.8}$$

N' je inverzním prvkem pro **n** a **R'** inverzním pro **r**. K číslu **N'** se můžeme dopočítat dvěma způsoby.

- Užitím rozšířeného Euklidova algoritmu
- Užitím Euklidova teorému, který lze provézt, jelikož čísla **r** a **n** jsou nesoudělná

Ve své práci jsem se rozhodl pro užití Euklidova teorému v podobě:

$$n^{-1} \equiv n^{\phi(r)-1}\ mod(r) \tag{5.9}$$

A to pro jeho snazší implementaci také díky tomu, že je snadné určit $\phi(r) - 1$ pro naše $r = 2^{160}$. Výsledek $\phi(r) - 1$ se následně binárně zapíše 159 jedničkami. Tímto číslem se modulárně umocňuje s pomocí algoritmu popsaného v kapitole 5.4.2. Výstupem Euklidova teorému je v tomto případě ovšem inverzní číslo k n při obrácené Bézoutově rovnosti [18] a je nutné výsledek ještě odečíst od čísla **r** pro nalezení skutečného **N'**.

Jelikož je exponent tvořen pouze jedničkami [19] a modulujeme vybraným **r**[20], lze hledání čísla **N'** popsat takto:

[18] Tedy v podobě: $n \cdot N' - r \cdot R' = 1$

[19] Odpadá potřeba rozhodovací konstrukce if u modulárního umocňování

[20] Modulaci lze zjednodušit na práci pouze se spodními 160 bity

Výpis 5.4: Nalezení inverzního prvku s pomocí Euklidova teorému

```
1    InverseN(n):
2        c := n;
3        d := to_unsigned(1,160);
4        for i in 0 to 158 loop
5            var1 := d*c;
6            d := var1(159 downto 0);
7            var2 := c*c;
8            c := var2(159 downto 0);
9        end loop;
10       return r-d;
```

Montgomeryho součin a umocňování

Klíčový algoritmus Montgomeryho násobení je popsaný REDC(T) [28] sloužící k výpočtu $\bar{c} = \bar{a} \cdot \bar{b} \cdot r^{-1}\ mod(n)$, kde $T = \bar{a} \cdot \bar{b}$. Pokud za c označíme součin $a \cdot b$ platí následující vyjádření [29]:

$$\begin{aligned} \bar{c} &= \bar{a} \cdot \bar{b} \cdot r^{-1}\ mod(n) \\ &= a \cdot r \cdot b \cdot r \cdot r^{-1}\ mod(n) \\ &= c \cdot r\ mod(n) \end{aligned} \tag{5.10}$$

Z vnějšího pohledu pak můžeme říct, že algoritmus dělí vstup číslem **r** v **mod(n)** a jeho vyvolání na číslo v Montgomeryho formě vrací toto číslo v původní algebře modulo **n**. Funkci REDC(T) jsem v práci užil v podobě Montgomeryho součinu, kde místo čísla **T** jsou vstupem jeho součinitelé **a** a **b**. Ten lze vyjádřit v následující podobě:

Výpis 5.5: Montgomeryho součin

```
1    MontProd(a,b,n,N'):
2        T := a*b;
3        u := (t+((T*N')mod r)*n)/r;
4        if u < (n) then
5            return u;
6        end if;
7        return u-n;
```

Po převedení čísla do Montgomeryho formy jej můžeme takto libovolně modulárně násobit podle původní algebry s jinými čísly v Montgomeryho formě bez nutnosti modulování, či dělení modulem původní algebry.

S těmito poznatky lze popsané funkce skloubit do Montgomeryho umocňování pro výpočet $x = a^e mod(n)$

Výpis 5.6: Montgomeryho umocňování

```
1   MontExp(a,e,n):
2       N' := InverseN(n);
3       a' := SmallMod(a,n);
4       b' := r-n;       -- číslo 1 v Montgomeryho formě
5                        -- ekvivalentem je SmallMod(1,n);
6       for i in 159 downto 0 loop  -- pro 160 bitové e
7           b' := MontProd(b',b',n,N')
8           if e(i) = '1' then
9               b' := MontProd(b',a',n,N');
10          end if;
11      end loop;
12      return MontProd(b',1,n,N');
```

V rámci Miller-Rabinova testu jsou kontrolovány platnosti kongruencí $a^{p-1} \equiv 1\ mod(p)$ a $a^{2^{s-1}d} \equiv \pm 1\ mod(p)$. Pokud některé $a^{2^{s-1}d} \equiv 1\ mod(p)$, pak se skrze umocňování jedničky s jedničkou tento stav pouze vleče do finálního $a^{p-1} \equiv 1\ mod(p)$ a nemá smysl na něj testovat. V případě $a^{2^{s-1}d} \equiv 1\ mod(p)$ lze mezivýsledky porovnávat s Montgomeryho formou čísla -1. Ta lze spočíst jako:

$$\begin{aligned} M_{-1} &= -1 \cdot r\ mod(n) \\ M_{-1} &= -r\ mod(n) \end{aligned} \tag{5.11}$$

A v našem případě, kde je r voleno tak, že se do něj n vleze právě jednou můžeme M_{-1} vyjádřit takto:

$$M_{-1} = 2 \cdot n - r \tag{5.12}$$

Mezivýsledky jsou porovnávány s M_{-1} a neustále je aktualizovaná boolean hodnota **bless**, která se při e(i) = '1' resetuje na neprůchozí, jelikož dosavadní část nemohla být do této míry dělitelná 2, aby ji možné hodnoty s [21] zahrnuly do rovnice. V případě, že na konci smyčky je mezivýsledek roven M_{-1} je boolean hodnota nastavena na průchozí. Protože rovnice a jejich vyhodnocování se v posledním cyklu liší, volím za číslo **e** svrchních 159 bitů čísla **n** a poslední cyklus[22] probíhá bokem s upravenou kontrolou Miller-Rabinovým testem.

[21] s určuje kolikrát lze dělit testované číslo 2 po odečtení 1. s-1 je tedy počet nul v řadě za sebou před poslední jedničkou v bitovém zápisu čísla n.

[22] Vždy jde také o další kolo umocňování s bitem e(i) = '0', jelikož usilujeme o umocnění na sudé číslo

Po skončení celého cyklu je kontrolována $a^{n-1} \equiv 1\ mod(n)$ a stav **bless**. Pokud je některý z výrazů pravdivý, tak svědek neodhalil složenost čísla a s pravděpodobností 75 % je **n** prvočíslo.

5.4.4 Problém násobení velkých čísel

Obdobné problémy, s kterými se potýká modulování velkými čísly, obnáší také problematika jejich násobení. Tedy, že operace s velkými čísly prochází simulátorem, ale při syntéze způsobují nerozpoznanou chybu, jelikož jsou vývojovým prostředím obecně považovány za syntetizovatelné. V doporučené publikaci je operátor '*' omezen na práci pouze s celočíselnými typy, typy s plovoucí čárkou a fyzické typy [1]. Toto omezení již také není pro schopnosti Vivada zcela aktuální. Dle dostupných prostředím nabízených IP bloků můžeme vyvodit, že program dokáže vytvořit až 64 bitovou násobičku s výstupem 128 bitů.

V dříve popsaných algoritmech běžně probíhá násobení dvou 160 bitových čísel, které ale není v jeho prosté podobě syntetizovatelné. Pro řešení tohoto problému jsem byl nucen implementovat v podobě funkce vlastní 160 bitovou násobičku [23]. Princip násobičky spočívá v násobení rozkladem s obecnou podobou:

$$\begin{aligned} a \cdot b &= (a_1 + a_2 + a_3) \cdot (b_1 + b_2 + b_3) \\ &= a_1 \cdot b_1 + a_1 \cdot b_2 + a_1 \cdot b_3 + a_2 \cdot b_1 + a_2 \cdot b_2 + a_2 \cdot b_3 + a_3 \cdot b_1 + a_3 \cdot b_2 + a_3 \cdot b_3 \end{aligned} \tag{5.13}$$

Pokud číslo rozdělíme podle cifer s jejich předznamenáním, lze třeba v dekadické soustavě násobit trojciferná čísla s užitím malé násobilky:

$$\begin{aligned} 256 \cdot 382 &= (2 \cdot 100 + 5 \cdot 10 + 6) \cdot (3 \cdot 100 + 8 \cdot 10 + 2) \\ &= 2 \cdot 100 \cdot 3 \cdot 100 + 2 \cdot 100 \cdot 8 \cdot 10 + 2 \cdot 100 \cdot 2 + 5 \cdot 10 \cdot 3 \cdot 100 \\ &\quad + 5 \cdot 10 \cdot 8 \cdot 10 + 5 \cdot 10 \cdot 2 + 6 \cdot 3 \cdot 100 + 6 \cdot 8 \cdot 10 + 6 \cdot 2 \\ &= 6 \cdot 10000 + 16 \cdot 1000 + 4 \cdot 100 + 15 \cdot 1000 + 40 \cdot 100 \\ &\quad + 10 \cdot 10 + 18 \cdot 100 + 48 \cdot 10 + 12 \\ &= 97792 \end{aligned} \tag{5.14}$$

Tímto způsobem můžeme požadovanou 160 bitovou násobičku složit z 64 bitových při rozložení každého vstupu na dvě 64 bitová a jedno 32 bitové číslo. Funkci zastupující 160 bitovou násobičku pak můžeme popsat následovně:

[23] Případně její alternace, jako je 160 bitová modulární násobička s modulem **r**, kde jsou šetřeny výpočetní prostředky

Výpis 5.7: Násobička pro 160 bitové součinitele

```
1   Mult160 (a,b):
2       a1:=a (63 downto 0);
3       a2:=a (127 downto 64);
4       a3:=a (159 downto 128);
5       b1:=b (63 downto 0);
6       b2:=b (127 downto 64);
7       b3:=b (159 downto 128);
8       c11:= a1*b1;
9       c12:= ((a1*b2)&to_unsigned(0,64));
10      c13:= ((a1*b3)&to_unsigned(0,128));
11      c22:= ((a2*b2)&to_unsigned(0,128));
12      c21:= ((a2*b1)&to_unsigned(0,64));
13      c23:= ((a2*b3)&to_unsigned(0,192));
14      c31:= ((a3*b1)&to_unsigned(0,128));
15      c32:= ((a3*b2)&to_unsigned(0,192));
16      c33:= ((a3*b3)&to_unsigned(0,256));
17      return c11+c12+c13+c22+c21+c31+c32+c33+c23;
```

Tato násobička je specificky vytvořena pro práci se 160 bitovými čísly a v případě potřeby upravit generátor na čísla o jiných délkách by ji bylo nutné nahradit a nejedná se tak o prosté škálování konstant, jak by tomu bylo u jiných v práci popsaných algoritmů[24].

5.4.5 Problém přetečení a nenaplnění hodnot

V rámci práce ve Vivadu jsem se setkal s problematikou pro program nejasného naplnění proměnných, co se délky výstupu týče. V případě součtu dvou čísel zapsaných x bity může být výsledek zapsán až x+1 bity. Vivado tento stav předpokládá a simulaci při nedostatečný velikostech proměnných nepovolí. Ovšem pokud součet do x+1 bitů nepřeteče, je možné, že poslední bit z výsledku zůstane nepřiřazený, což komplikuje budoucí výpočty.

Pro tento nedostatek jsem se uchýlil k přičítání x+1 bitové nuly na konci každého součtu a proto konec posledního výpisu je v práci reprezentován tímto způsobem:

```
1       return c11+c12+ . . . +c23+to_unsigned(0,321);
```

Tento nedostatek se projevuje i u násobení a před přiřazením hodnoty je přičtena nula o bitové délce rovné součtu bitových délek součinitelů.

[24]Bylo by nutné připsat další proměnné.

V mém návrhu jsou také místy užity 161 bitové zápisy čísel i v případě nevyužití posledního bitu. Důvodem je Vivadem chybně určený požadavek na modulování číslem r v podobě 2^{160}. Mělo by se jednat o předání spodních 160 bitů, ale jelikož Vivado pracuje s modulem o 161 bitech, požaduje výstup zapsat 161 bity i přesto, že poslední bit bude vždy '0'.

6 Výsledky praktické části

Finálním výstupem praktické části je celkem pět výkonnostně rozdílných verzí generátoru 160 bitových prvočísel splňujících veškeré bezpečnostní požadavky pro zvolené užití v kryptografii.

Výsledné návrhy přijímají v příslušných intervalech náhodné vstupy **RNG** o 160 bitech a v případě objevení vstupu vedoucího k nalezení prvočísla aktualizují výstupní port **result** na nalezené prvočíslo.

Dle pokynů vedoucího byla provedena syntéza v režimu "out of context"se zaměřením na implementaci na desce xcvu7p-flvb21042i ze série Virtex UltraScale+ od společnosti Xilinx. Finální měření a syntézy byly provedeny ve verzi Vivado 2022.2.2 běžící na 64 bitovém operačním systému Windows 10 Pro s operační pamětí o velikosti 64 GB.

6.1 Vyhodnocení jednotlivých modelů

Každý z představovaných návrhů je strukturálně tvořen dvěma moduly. Hlavním **Main** a komponentem **Testwit**, který je vždy generován čtyřicetkrát pro ověření čísla s pomocí čtyřiceti různých svědků. V Mainu vždy probíhá příprava potřebných parametrů k Montgomeryho umocňování a finální vyhodnocení vrácených výsledků.

6.1.1 Model č.1

Model č.1 je základní podobou generátoru, která vždy v každé komponentě zpracovává po dobu 160 hodinových cyklů jedno číslo a tedy dává jeden výsledek každých 160 hodinových cyklů a to se zpožděním 320 cyklů po k výsledku vázanému vstupu. Co se hardwarové naročnosti týče, tak syntéza toto řešení zrealizovala s pomocí 89 LUT (LookUp Table) a 664 FF (Flip Flop). Minimální hodinové periody se podařilo dosáhnout o velikosti 1,401 ns, což odpovídá maximální frekvenci 713 MHz.

S pomocí dříve popsané funkce pro aproximaci počtu prvočísel můžeme odvodit pravděpodobnost, že zkoušený náhodný vstup vygeneruje prvočíslo. Nejprve je vhodné odečíst počet prvočísel do čísla 2^{159} od čísel do 2^{160}. Protože automaticky vyřazujeme lichá čísla a výsledek je odvozen z pouze 158 bitů, lze pravděpodobnost spočíst následovně:

$$\frac{\frac{2^{160}}{ln(2^{160})} - \frac{2^{159}}{ln(2^{159})}}{2^{158}} = 0,0179 \tag{6.1}$$

Lepším znázorněním tohoto výsledku je, že s pravděpodobností 1,79 % nám náhodný vstup vygeneruje prvočíslo a jedná se tedy o každý 55,8 vstup.

Pokud tedy model **Model č.1** otestuje vstup každých $160 \cdot 1,401 = 224,16\ ns$, tak prvočíslo vygeneruje v průměru každých $224,16\ ns \cdot 55,8 = 12,5\ \mu s$. Za maximální frekvence tedy **Model č.1** generuje **80 tisíc prvočísel za sekundu**.

6.1.2 Model č.2

Model č.2 využívá vnitřní skladby modulů, kde je v **Main** potřeba sekvenčně provést 159 kroků inverze čísla s pomocí Euklidova teorému a v **Testwit** je to 159 kroků Montgomeryho umocňování. Číslo 159 lze faktorizovat na $3 \cdot 53$ a proto **Model č.2** zkracuje oba procesy do 53 hodinových cyklů, kde v každém z nich jsou provedeny 3 kroky příslušného algoritmu. Takto jsou vždy zpracovávány dvě čísla s výsledkem dostupným každých 54 hodinových cyklů se zpožděním od evokujícího vstupu 108 hodinových cyklů. Výsledná syntéza úspěšně zjednodušila logiku a dosáhla výsledku s užitím 77 LUT (LookUp Table) a 664 FF (Flip Flop). Minimální délka periody byla vyměřena na 1,669 ns a maximální frekvence modelu je tedy 599 MHz.

Dle naměřených hodnot **Model č.2** ověří vstup každých $54 \cdot 1,669 = 90,126\ ns$, což odpovídá nalezení prvočísla každých $90,126\ ns \cdot 55,8 = 5\ \mu s$. Při maximální frekvenci je model schopen vygenerovat až **200 tisíc prvočísel za sekundu**.

6.1.3 Model č.3

Model č.3 využívá stejného principu, co **Model č.2**, ale pokouší se o ambiciózněji krok a to rozdělení vnitřních algoritmů do 3 hodinových cyklů s 53 průběhy v jednom.

Syntéza prvního modelu trvala kolem půl hodiny. V případě druhého bylo výsledku dosaženo po dvou hodinách. Pro **Model č.3** jsem se výsledku nedočkal ani po osmi hodinách syntézy, kdy mi Vivado nahlásilo blíže nespecifikovanou chybu "EXCEPTION_ACCESS_VIOLATION", pro kterou syntéza modelu skončila neúspěchem.

Model č.3 tedy zůstává jen pouze teoreticky proveditelným řešením, jehož fungování si můžeme ověřit na simulátoru.

6.1.4 Modely č.4 a č.5

Modely č.4 a č.5 usilují s pomocí registrů neustále plnit struktury **modelů č.1 a č.2** a předávat si své mezivýsledky tak, aby v každém hodinovém cyklu došlo k přijmutí vstupu a vydání případného výsledku. Pro odvázání současně prověřovaného čísla od předávané náhodnosti posunutí skupiny svědků jsem ji rozšířil na 4 bity.

Registry jsou zpracovány do podoby matice [1], kde se předávají příslušné hodnoty s pomocí indexace.

Bohužel v případě **modelů č.4 a č.5** došlo při syntéze k maximálnímu vytížení výpočetních prostředků [2] a proces tak nedoběhl. Modely **č.4** a **č.5** jsou tedy s prací přiloženy pro návrh teoreticky nejvýkonnější implementace s využitím registrů a náhodnějšího volení svědku, jejíž funkčnost lze ověřit pouze v rámci simulátoru.

6.2 Chybovost modelů

Jak bylo popsáno v sekci 1.3.1 v její části o Miller-Rabinově testu, pro každé složené číslo je $\frac{1}{4}$ jeho možných svědků lháři a chybně jej označují za prvočíslo. V případě užití 40 svědků je tedy pravděpodobnost $(\frac{1}{4})^{40}$, že složené číslo bude označeno za prvočíslo. V našem testovaném vzorku je každé 55,8 číslo prvočíslem a proto na jedno prvočíslo připadá 54,8 čísel složených. Z tohoto důvodu na každé nalezené skutečné prvočíslo připadne $\frac{54{,}8}{4^{40}} = 4,533 \cdot 10^{-23}$ složených čísel chybně označených za prvočísla.

V případě našich modelů se tak **Model č.1** dopustí chyby v průměru jednou za **8,74 miliard let** a rychlejší **Model č.2** chybuje jednou každých **3,5 miliardy let**. Tak dlouho by na nich také trvalo najít číslo, které vždy projde konkrétní skupinou 40 svědků, co by představovalo pro takové nastavení obdobu Carmichaelova čísla pro Fermatův test. Zranitelnost vůči tomuto číslu je pak jen 25 %, jelikož jsou v **modelech č.1 a č.2** užívány 4 sety svědků.

6.3 Porovnání výsledků

V následující tabulce lze porovnat dosažené výsledky jednotlivých modelů, které byly popsány výše. Pro modely **č.3**, **č.4** a **č.5** chybí některé údaje, jelikož jsou pouze názornými možnostmi řešení v simulátoru a neproběhla jejich syntéza.

[1] Je vytvořen vlastní datový typ

[2] Především paměti RAM

Porovnání výsledků jednotlivých modelů					
Model	**č.1**	**č.2**	**č.3**	**č.4**	**č.5**
Zpoždění [clk]	320	108	8	320	108
Výkonnost [clk]	160	54	4	1	1
Minimální hodinová perioda [ns]	1,401	1,669	-	-	-
Maximální hodinová frekvence [MHz]	713	599	-	-	-
Počet testovaných čísel za sekundu [mil.]	4,461	11,096	-	-	-
Odhadovaný počet nalezených prvočísel za sekundu [tis.]	79,95	198,85	-	-	-
Počet požadovaných součástek LUT	89	77	-	-	-
Počet požadovaných součástek FF	664	664	-	-	-

Tab. 6.1: Porovnání jednotlivých modelů

Zpoždění představuje vstupní latenci se kterou se dostavuje příslušná odpověď na poskytnutý vstup a Výkonnost určuje periodu s jakou je model schopen podávat výsledky.

Značný nárůst výkonu mezi modely č.1 a č.2 přikládám skutečnosti, že se v případě obou modelů pohybujeme v příliš vysokých frekvencích. S takto nízkou hodinovou periodou může být výraznějším zdrojem zpoždění i samotná propagace signálu a nepochybně nás omezuje fyzická náročnost dosažení takto vysokých frekvencí.

Závěr

Tato práce se zaměřuje na problematiku generování kryptograficky-bezpečných prvočísel s pomocí hardware. Popisuje vlastnosti prvočísel a jejich užití v kryptografii. Představuje příslušné normy stanovující bezpečnostní požadavky na generování prvočísel a zaznamenává jejich vývin. Dále se práce věnuje samotné implementaci generátoru kryptograficky-bezpečných prvočísel na platformě FPGA spolu s matematickými principy jako je Montgomeryho umocňování, které zpracování takto velikých čísel umožňují.

Výstupem práce jsou dva plně funkční syntetizované generátory 160 bitových prvočísel, které splňují současné požadavky pro implementaci do příslušných kryptografických algoritmů a bylo tak tedy dosaženo stanoveného cíle. Spolu s nimi jsou poskytnuty další tři simulovatelné modely popisující potenciální možnosti navýšení výkonu s pomocí paralelizace. Dle naměřených hodnot zrealizované generátory poskytují až 80 a 200 tisíc prvočísel za sekundu.

Pro užití generátoru je potřeba mu poskytnou náhodný 160 bitový vstup, kde první dva bity ovlivňují náhodnost svědků Miller-Rabinova testu, na kterém jsou všechny modely postaveny a zbylých 158 bitů definuje testované číslo. Výstupem každého z návrhů je pak vždy vygenerované prvočíslo a tedy v případě, že náhodný vstup nevede k získání prvočísla, není výstupní hodnota aktualizována. Pro pravděpodobnostní charakteristiku Miller-Rabinova testu byla vyčíslena chybovost modelů, která je pro splnění požadavků na kryptograficky-bezpečná prvočísla v praktickém užití zanedbatelná.

Literatura

[1] Jiří Pinkr and Martin Poupa. *Číslicové systémy a jazyk VHDL*. 2006.

[2] R. J. Gillings. The recto of the rhind mathematical papyrus how did the ancient egyptian scribe prepare it? . *Archive for History of Exact Sciences*, 1974. URL: https://sci-hub.se/https://link.springer.com/article/10.1007/BF01307175#citeas.

[3] Euclid. *Euclid's Elements*. Green Lion Press, Santa Fe, NM, January 2002.

[4] Mario Livio. *Je Bůh matematik?* 2010.

[5] Torkel Franzen. Logic and theories. In *Inexhaustibility*, pages 119–142. Cambridge University Press, Cambridge, March 2017.

[6] D. H. Lehmer. Tests for primality by the converse of Fermat's theorem. *Bulletin of the American Mathematical Society*, 33(3):327 – 340, 1927. URL: https://doi.org/, doi:bams/1183492108.

[7] Eliška Ochodková. Matematické základy kryptografických algoritmů. *Vysoká škola báňská – Technická univerzita Ostrava & Západočeská univerzita v Plzni*, 2011. URL: https://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/matem_zaklady_kryptograf_algoritmu_obr.pdf.

[8] Chris K. Caldwell. How many primes are there? *Prime Pages*, 2008. URL: https://web.archive.org/web/20121015002415/http://primes.utm.edu/howmany.shtml.

[9] Jan Řeháček. Matykání v: Tajemný svět prvočísel. *Idnes.cz*, 2017. URL: https://janrehacek.blog.idnes.cz/blog.aspx?c=454057.

[10] Václav Šimerka. Zbytky arithmetické posloupnosti. *Časopis Pro Pěstování Matematiky a Fysiky*, 1885. URL: https://dml.cz/handle/10338.dmlcz/122245.

[11] R. D. Carmichael. Note on a new number theory function. *Bulletin of the American Mathematical Society*, 1910. URL: https://www.ams.org/journals/bull/1910-16-05/home.html.

[12] Gary L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 1976. URL: https://dl.acm.org/doi/10.1145/800116.803773.

[13] Michael O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 1980. URL: https://www.sciencedirect.com/science/article/pii/0022314X80900840?via%3Dihub.

[14] Milan Oulehla and Roman Jašek. *Moderní kryptografie*. 2017.

[15] Elaine Barker and Quynh Dang. Recommendation for key management part 3: Application-specific key management guidance. *NIST Special Publication 800-57 Part 3 Revision 1*, 2015. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf.

[16] Ronald L. Rivest and Robert D. Silverman. Are 'Strong' Primes Needed for RSA? 1999. URL: http://people.csail.mit.edu/rivest/RivestSilverman-AreStrongPrimesNeededForRSA.pdf.

[17] J.P. Buhler, H.W. Lenstra Jr., and Carl Pomerance. Factoring integers with the number field sieve. *The Development of the Number Field Sieve*, 1993. URL: https://sci-hub.se/https://link.springer.com/chapter/10.1007/BFb0091539.

[18] Mike Rosulek. The joy of cryptography. *School of Electrical Engineering & Computer Science, Oregon State University, Corvallis, Oregon, USA*, 2021. URL: https://joyofcryptography.com/pdf/book.pdf.

[19] Peter Wilson. High speed video application. In *Design Recipes for FPGAs*, pages 67–77. Elsevier, 2016.

[20] Tim E. Güneysu. Fpgas in cryptography. *Department of Electrical Engineering and Information Technology, Ruhr-University Bochum, Bochum, Germany*, 2011. URL: https://sci-hub.se/https://link.springer.com/referenceworkentry/10.1007/978-1-4419-5906-5_31.

[21] L. Nagel and R. Rohrer. Computer analysis of nonlinear circuits, excluding radiation (cancer). *IEEE Journal of Solid-State Circuits*, 6(4):166–182, 1971. doi:10.1109/JSSC.1971.1050166.

[22] Andrei Vladirmirescu. Shaping the history of spice. *IEEE Solid-State Circuits Magazine*, 3(2):36–39, 2011. doi:10.1109/MSSC.2011.942105.

[23] Lukáš Sekanina. Úvod do jazyka vhdl. *Návrh počítačových systémů INP, Brno: Vysoké učení technické v Brně, Fakulta informačních technologií*, 2006.

[24] Nigel P Smart. *Algorithms, key size and parameters: Report - 2014*. 2013.

[25] Cameron F. Kerry and Patrick D. Gallagher. Fips 186-4: Digital signature standard. *Information Technology Laboratory, National Institute of Standards and Technology*, 2013. URL: https://doi.org/10.6028/NIST.FIPS.186-4.

[26] Gina M. Raimondo and Laurie E. Locascio. Fips 186-5: Digital signature standard. *Information Technology Laboratory, National Institute of Standards and Technology*, 2023. URL: https://doi.org/10.6028/NIST.FIPS.186-5.

[27] Carl Pomerance. A tale of two sieves. *NOTICES OF THE AMS*, 1996. URL: https://www.ams.org/notices/199612/pomerance.pdf.

[28] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985. URL: https://www.ams.org/journals/mcom/1985-44-170/S0025-5718-1985-0777282-X/S0025-5718-1985-0777282-X.pdf, doi:10.1090/s0025-5718-1985-0777282-x.

[29] Çetin Kaya Koç, Tolga Acar, and Burton S. Kalisky Jr. Analyzing and comparing montgomery multiplication algorithms. *Department of Electrical & Computer Engineering & RSA Laboratories*, 1996. URL: https://www.microsoft.com/en-us/research/wp-content/uploads/1996/01/j37acmon.pdf.

Seznam symbolů a zkratek

CANCER Computer Analysis of Non-linear Circuits, Excluding Radiation

DSA Algoritmus digitálního podpisu – Digital Signature Algorithm

ECC Kryptografie na eliptických křivkách – Elliptic Curve Cryptography

EMV Čipy kreditních karet od společností Europay, Mastercard a Visa

ENISA Evropská agentura pro bezpečnost sítí a informací – European Network and Information Security Agency

ETSI Evropský ústav pro telekomunikační normy – European Telecommunications Standards Institute

FF Flip Flop

FIPS Federal Information Processing Standards

FPGA Programovatelné hradlové pole – Field Programmable Gate Array

GIMPS Great Internet Mersenne Prime Search

GSM Groupe Spécial Mobile

HDL Jazyk pro popis hardware – Hardware Description Language

HLR Domovský registr – Home Local Register

IEC Mezinárodní elektrotechnická komise – International Electrotechnical Commission

IEEE Institut pro elektrotechnické a elektronické inženýrství – Institute of Electrical and Electronics Engineers

IMSI International Mobile Subscriber Identity

ISO Mezinárodní organizace pro normalizaci – International Organization for Standardization

LUT LookUp Table

M-R test Miller-Rabinův test prvočíselnosti

NIST Národní institut standardů a technologie – National Institute of Standards and Technology

RSA	Šifra, jejíž autory jsou Ronald L. Rivest, Adi Shamir a Leonard M. Adleman
SPICE	Simulation Program for Integrated Circuit Emphasis
SWAC	Standards Western Automatic Computer
TS	Technical Specification
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLR	Návštěvnický registr – Visitor Locator Register
$REDC(T)$	Montgomeryho redukce – Montgomery reduction
$GF(p^k)$	Konečné pole / Konečné těleso / Galoisovo těleso – Finite field / Galois field
$\pi(x)$	Prvočíselná funkce – Prime-counting function
$\phi(n)$	Eulerova funkce – Euler's totient function

Seznam příloh

A Zdrojové kódy použitých Python modelů **75**
A.1 Generování prvočísel s pomocí Fermatova testu 75
A.2 Generování prvočísel s pomocí Miller-Rabinova testu 76

B Struktura elektronické přílohy **77**

A Zdrojové kódy použitých Python modelů

A.1 Generování prvočísel s pomocí Fermatova testu

Výpis A.1: Model generování prvočísla pomocí Fermatova testu v jazyce Python

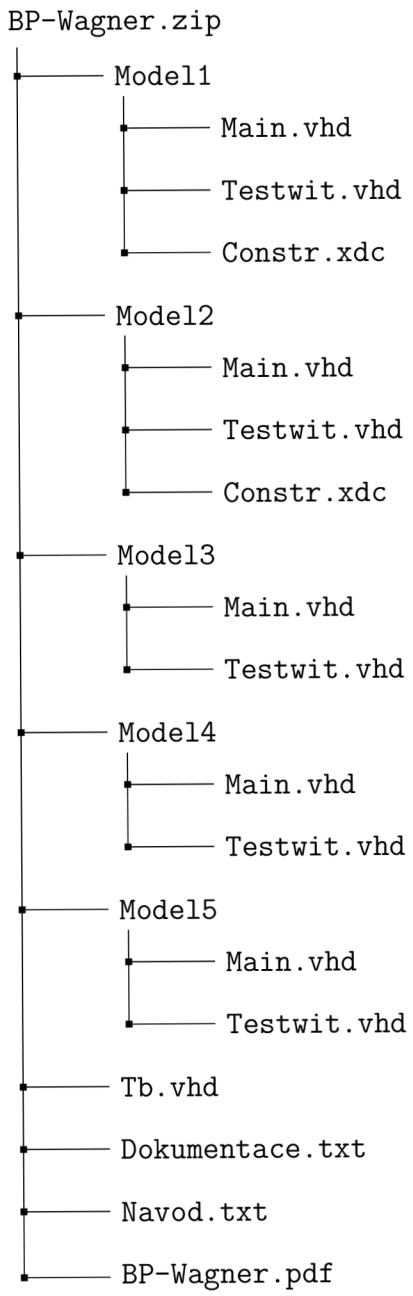
```
1  import random
2  run = True
3  while(run):
4      run = False
5      limit = 0
6      while(limit == 5 or limit%2 == 0):
7      #Jsou vyřazována čísla dělitelná 2 a 5
8          n = random.getrandbits(20)
9          limit = n%10
10     for a in range(2):
11         if (a+2)**(n-1)%n !=1:
12             continue
13 print(n)
```

A.2 Generování prvočísel s pomocí Miller-Rabinova testu

Výpis A.2: Model generování prvočísla pomocí Miller-Rabinova testu v jazyce Python

```
1  import random
2  run = True
3  while(run):
4      limit = 0
5      while(limit == 5 or limit%2 == 0 or n<5):
6          n = random.getrandbits(20)
7          limit = n%10
8      d = n-1
9      i = 0
10     while(d % 2 == 0):
11         d = d//2
12         #Dělení se zaokrouhlením pro
13         #chybu Pythonu u velkých čísel
14         i = i + 1
15     for r in range(i+1):
16         x = 2**(d*2**r)%n
17         #Za svědka je na pevno dosazeno číslo 2
18         if r == i:
19                 if x == 1:
20                     break
21                 else:
22                     continue
23         if x == 1 or x == n-1:
24              break
25 print(n)
```

B Struktura elektronické přílohy



Obr. B.1: Struktura souborů odevzdaných v rámci elektronické přílohy