FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Faktorizace pomocí eliptických křivek
Student:	Bc. Jakub Dvořák
Vedoucí:	doc. Ing. Ivan Šimeček, Ph.D.
Studijní program:	Informatika
Studijní obor:	Počítačová bezpečnost
Katedra:	Katedra informační bezpečnosti
Platnost zadání:	Do konce letního semestru 2021/22

Pokyny pro vypracování

1. Nastudujte a popište problém faktorizace a příklad jeho využití v kryptografii.
2. Popište princip faktorizace pomocí Lenstrovi metody (ECM). [1]
3. Implementujte ve vhodně zvoleném programovacím jazyku ECM za použití eliptických křivek ve Weierstrassově formě a Edwardsových křivek. [2, 3]
4. Diskutujte možnou paralelizaci implementovaných metod a paralelizaci implementujte.
5. Diskutujte naměřené výsledky.

Seznam odborné literatury

[1] Parker, D.: Elliptic Curves and Lenstra's Factorization Algorithm. University of Chicago, 2004
[2] Daniel J. Bernstein and Tanja Lange: Faster addition and doubling on elliptic curves, University of Illinois at Chicago 2007
[3] DANIEL J. BERNSTEIN, PETER BIRKNER, TANJA LANGE, AND CHRISTIANE PETERS: ECM USING EDWARDS CURVES

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 18. listopadu 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Porovnání algoritmů pro faktorizaci velkých celých čísel

Bc. Jakub Dvořák

Katedra informační bezpečnosti
Vedoucí práce: doc. Ing. Ivan Šimeček, Ph.D.

7. ledna 2021

Poděkování

Na tomto místě bych rád poděkoval doc. Ing. Ivanovi Šimečekovi, Ph.D. za vedení této práce a věcné rady při zpracovávání. Také bych rád poděkoval své rodině za velkou podporu během celého mého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisu, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programu, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli zpusobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelum). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným zpusobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 7. ledna 2021 ……

Odkaz na tuto práci

Dvořák, Jakub. *Porovnání algoritmů pro faktorizaci velkých celých čísel.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

V této práci se zabývám faktorizačním algoritmem, který využívá eliptických křivek, jinak známý také jako Lenstrův algoritmus. Analyzuji a porovnávám tento algoritmus nad dvěma modely eliptických křivek, kterými jsou Weierstrassův a Edwardsův model, za použití projektivního souřadnicového systému. V první části zavedu matematické pojmy, které jsou nutné pro pochopení popisovaného algoritmu společně s popisem modelů eliptických křivek. Následně popíši kryptosystém RSA, který je založen na problému faktorizace. V další části se zaměřuji na Lenstrův algoritmus, který detailněji popíši. Na závěr jsou uvedeny možnosti implementace s některými vylepšeními a paralelizací Lenstrova algoritmu společně s výsledky měření.

Klíčová slova Edwardsova křivka, Weierstrassova křivka, faktorizace, Lenstrův algoritmus, paralelizace

Abstract

In this thesis I deal with factorization algorithm that uses elliptic curves, also known as Lenstra's algorithm. I analyze this algorithm using two elliptic curve models in Weierstrass and Edwards form using projective coordination system. In the first part I define mathematical concepts that are necessary for understanding described algorithm together with description of some known ellptic curve forms. In the next part I describe RSA cryptosystem that is based on factorization problem. Then I focus on Lenstra's algorithm, which I describe in more detail. At the end I desribe implementation and paralelization with some improvements of algorithm and show results of measurements.

Keywords Edwards elliptic curve, Weierstrass elliptic curve, factorization, Lenstra's algorithm, parallelization

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Šifrovaná komunikace na internetu je dnes téměř běžnou činností. Některé šifrovací kryptosystémy bývají založeny na matematických problémech, které jsou pro nás dnes obtížně řešitelné. Mezi takové problémy patří například problém diskrétního logaritmu nebo problém faktorizace celých čísel. Pokud by se nám tedy podařilo některý z těchto problémů efektivně řešit, tak bychom tím mohli narušit kryptosystém, který je na daném problému založen, a tím dešifrovat tajná data.

Faktorizace celých čísel je dnes stále diskutované téma. Tento problém není pouze definován v dnešní době. Tímto problémem se zabývali matematici ještě v dobách před naším letopočtem jako například Eukleidés, avšak pevné základy tohoto problému se začaly definovat až v období kolem 17. století. V této době se daným problémem začali zabývat známí matematici jako byli Leonhard Euler nebo Pierre de Fermat. Počítače výpočet sice urychlily, avšak doposud známé algoritmy jsou stále pomalé. Od 70. let 20. století se začínají objevovat algoritmy, jejichž časová složitost je subexponenciální.

Problém faktorizace celých čísel řadíme do třídy takzvaných NP–problémů. To znamená, že nedokážeme faktorizovat čísla v polynomiálním čase, tedy výpočet může trvat i několik let od nějaké velikosti faktorizovaného čísla. Díky tomu se na tomto problému zakládají některé šifrovací algoritmy jako je například RSA. Pokud bychom byli schopni faktorizovat číslo v reálném čase, potom by byla narušena bezpečnost kryptosystémů založených na problému faktorizace a tím bychom mohli dešifrovat naše utajené zprávy, které byly zašifrovány právě pomocí šifry založené na problému faktorizace.

I když se dnes zdá být tento problém neřešitelný, tak algoritmus pro kvantové počítače již vyvinut byl a ten ho dokáže řešit v polynomiálním čase. Tento algoritmus je pojmenovám podle svého objevitele, amerického profesora na MIT, Petera Shora, Shorův algoritmus. Společnosti IBM se sice již povedlo implementovat tento algoritmus na jejich kvantovém systému (více viz `https://www.ibm.com/quantum-computing/`), avšak jak je uvedeno v [1], tak

stále tento systém není v takové fázi vývoje, aby ohrozil dnešní kryptosystémy.

Kapitola 1

Teoretický základ pro metody faktorizace

V této kapitole se seznámíme se základní teorií. Následující definice jsou důležité pro pochopení fungování použitých faktorizačních algoritmů. Dané pojmy budou použity v další kapitole s popisem algoritmu. Budeme zde vycházet zejména ze studijního textu k předmětu MI-MKY [2] a skript [3]. Jiné použité zdroje budou specifikovány u příslušné části textu.

1.1 Základní algebraické struktury

Definice 1. (Grupa) **Grupou** nazýváme uspořádanou dvojici $(M, \circ)$, kde M je neprázdná množina (*nosič*) a $\circ$ je binární operace, pokud splňuje následující podmínky:

- asociativity, tedy $a \circ (b \circ c) = (a \circ b) \circ c$, platí pro každé $a, b, c \in M$,
- existence neutrálního prvku e vůči binární operaci $\circ$, tedy $a \circ e = e \circ a$ pro každé $a \in M$,
- existence inverzních prvků – ke každému $a \in M$ existuje prvek $a^{-1} \in M$ takový, že $a \circ a^{-1} = e = a^{-1} \circ a$.

Pokud navíc splňuje podmínku komutativity, tj. $a \circ b = b \circ a$, pro každé $a, b \in M$, nazýváme grupu **abelovskou** (komutativní). Pokud se bavíme o prvku grupy $g \in G$, tak myslíme prvek *nosiče*, tedy $g \in M$. Pro abelovské grupy značíme $\circ$ symbolem $+$, pokud se bavíme o aditivní notaci. Pro multiplikativní notaci budeme $\circ$ značit symbolem $\cdot$.

Definice 2. (Konečná grupa) Nechť $G = (M, \circ)$ je grupa. Grupu nazýváme konečnou, má-li množina M konečný počet prvků. **Řádem** konečné grupy

nazýváme počet prvků množiny M a značíme jej $\#G$. Pokud má množina M nekonečný počet prvků, pak říkáme, že řád grupy je nekonečný.

Definice 3. (Řád prvku) Nechť $G = (M, \circ)$ je grupa a $a \in M$ je prvkem této grupy. Nejmenší $x \in \mathbb{N}$, takové že $a^x = e$ nazýváme **řádem prvku**. Pokud takové x neexistuje, říkáme, že prvek má nekonečný řád.

Nyní si připomeňme pár notačních zvyklostí. Pokud se budeme bavit o multiplikativní notaci, tak místo $a \cdot b$ budeme psát pouze ab. Pro aditivní notaci budeme inverzní prvek k $a \in G$ (viz definici 1) místo a^{-1} značit $-a$. Neutrální prvek v aditivní notaci budeme značit symbolem 0. V multiplikativní notaci jej budeme značit symbolem 1.

V grupě si zavedeme **násobení** pro aditivní notaci takto:

$$\forall a \in G, \forall k \in \mathbb{N} : a^k = ka = \underbrace{a + a + \cdots + a}_{\text{k–krát}}$$

Pro umocňování v multiplikativní notaci pak takto:

$$\forall a \in G, \forall k \in \mathbb{N} : a^k = \underbrace{a \cdot a \cdot \cdots \cdot a}_{\text{k–krát}}$$

Uvedeme si pro příklad pár grup:

1. Množina celých čísel $\mathbb{Z}$ vybavená operací sčítání je prvků z této množiny tvoří grupu. Neutrálním prvkem je 0 a inverzním k $n \in \mathbb{Z}$ je $-n \in \mathbb{Z}$
2. Množina $\{0, 1, \ldots, n-1\}$ vybavená operací sčítání modulo n tvoří modulární aditivní grupu, kterou značíme $\mathbb{Z}_n^+$.
3. Množina $\{k \in \mathbb{Z} | 1 \leq k \leq n-1, \gcd(k, n) = 1\}$ vybavená operací násobení modulo n tvoří modulární multiplikativní grupu, kterou značíme $\mathbb{Z}_n^\times$.

Definice 4. (Podgrupa) Nechť je dána grupa $G = (M, \circ)$ a podmnožina N množiny M. O dvojici $H = (N, \circ)$ říkáme, že je **podgrupou** grupy G, pokud H tvoří grupu.

Definice 5. (Normální podgrupa) Podgrupu H grupy G nazýváme **normální podgrupou**, pokud pro každé $x \in G$ platí:

$$xHx^{-1} = H$$

1.2 Okruhy a tělesa

Definice 6. (Okruh) **Okruhem** R nazýváme množinu R se dvěma binárními operacemi $+ : R \times R \to R$ a $\cdot : R \times R \to R$ splňujícími:

- R je abelovská grupa vůči $+$,

- operace $\cdot$ je asociativní,
- platí levý i pravý distributivní zákon.

Pokud navíc:

- existuje neutrální prvek vůči násobení $\cdot$, nazýváme okruh **okruhem s jedničkou**,
- násobení $\cdot$ je komutativní, nazýváme okruh **komutativní**,
- je okruh komutativním a s jedničkou, a z rovnosti $ab = 0$ plyne $a = 0 \vee b = 0$, nazýváme okruh **oborem integrity**.
- $(R \setminus \{0\}, \cdot)$ tvoří grupu, nazýváme okruh **okruhem s dělením**.

Definice 7. (Těleso) Komutativní okruh s dělením nazýváme **tělesem**.

Definice 8. (Galoisovo těleso) Mějme prvočíslo p. **Galoisovo(konečné) těleso** definujeme jako trojici

$$(\{0, 1, \ldots, p-1\}, + \bmod p, \cdot \bmod p)$$

a toto těleso značíme $GF(p)$.

Definice 9. (Charakteristika tělesa) Bud T těleso. Pokud existuje $n \in \mathbb{N}$ takové, že $nr = 0$ pro každé $r \in T$, pak nejmenší takové n nazýváme **charakteristikou** tělesa T. O takovém tělesu T pak říkáme, že má kladnou charakteristiku. Pokud žádné takové n neexistuje, pak o T mluvíme jako o tělesu s charakteristikou 0.

Definice 10. (Podokruh, podtěleso) Podmnožinu S okruhu R nazýváme **podokruhem** okruhu R, pokud trojice $(S, +, \cdot)$ tvoří okruh. Analogicky definujeme i pro **podtěleso**.

1.3 Eliptické křivky

V této části si popíšeme teorii eliptických křivek a definujeme si operace nad eliptickými křivkami.

Definice 11. (Eliptická křivka) **Eliptickou křivkou** nad tělesem $\mathbb{R}$ nazýváme množinu $E \subset \mathbb{R}^2$ všech řešení rovnice

$$y^2 = x^3 + ax + b, (x, y) \in \mathbb{R}^2, a, b \in \mathbb{R}, 4a^3 + 27b^2 \neq 0$$

Tuto rovnici nazýváme také **Weierstrassovou** rovnicí. Takto definované podmínky nám zaručí korektní definici potřebných operací nad touto množinou. Do takovéto množiny se navíc přidává ještě bod O, který označuje neutrální prvek. Nyní si zavedeme operaci sčítání nad touto množinou.

Definice 12. (Operace $\oplus$) Mějme eliptickou křivku $E \cup \{O\}$ nad $\mathbb{R}$ danou rovnicí

$$y^2 = x^3 + ax + b, 4a^3 + 27b^2 \neq 0,$$

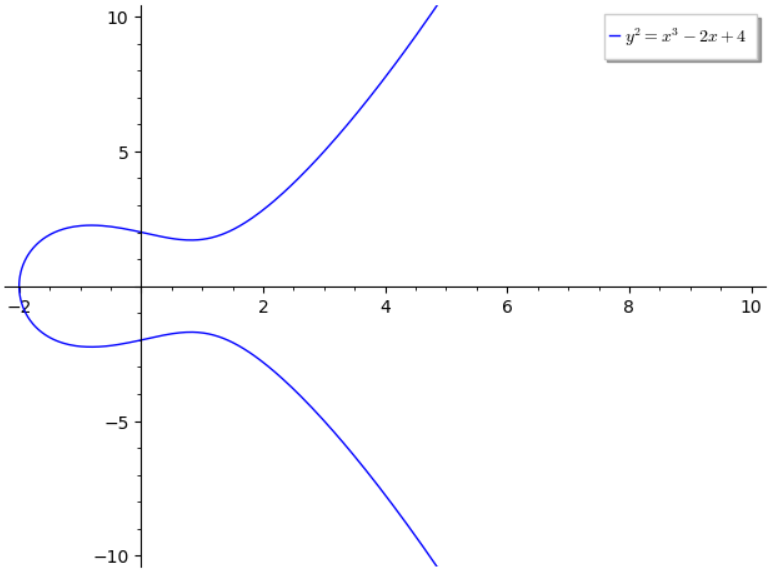
potom operaci $\oplus$ bodů $P, Q \in E$ definujeme takto:

1. $P = O \Rightarrow P + Q = Q$,
2. $P = (p_1, p_2), Q = (q_1, q_2), p_1 = q_1, p_2 = -q_2$, potom $P \oplus Q = O$, tedy bod Q můžeme označit také jako $-P$,
3. pokud nenastal ani jeden z bodů předtím, pak pro $P = (p_1, p_2), Q = (q_1, q_2)$

$$\lambda = \begin{cases} \frac{q_2 - p_2}{q_1 - p_1} & , P \neq Q \\ \frac{3p_1^2 + a}{2p_2} & , P = Q \end{cases}$$

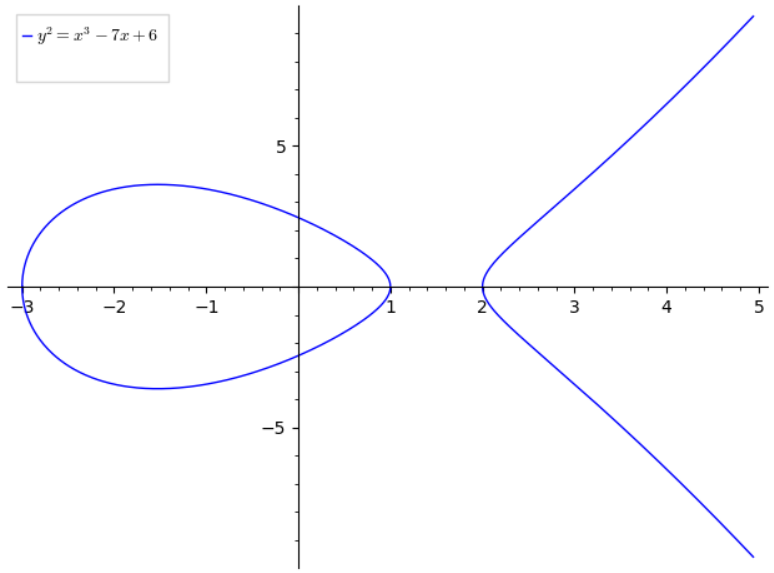
$P \oplus Q = (r_1, r_2)$, kde $r_1 = \lambda^2 - p_1 - q_1$ a $r_2 = \lambda(p_1 - r_1) - p_2$.

Takto definovaná operace a k ní daná množina potom splňuje podmínky pro abelovskou grupu. Příklady některých eliptických křivek můžeme vidět na obrázcích 1.1 a 1.2.



Obrázek 1.1: Eliptická křivka $y^2 = x^3 - 2x + 4$

Nyní máme definovanou eliptickou křivku nad $\mathbb{R}$, avšak v našem případě bude potřeba pracovat s eliptickými křivkami v modulární aritmetice, nad Galoisovým tělesem $GF(p)$, kde p je prvočíslo. Doplňme si tedy definici nad tímto tělesem.



Obrázek 1.2: Eliptická křivka $y^2 = x^3 - 7x + 6$

Definice 13. (Eliptická křivka nad $GF(p), p \geq 3$) Eliptickou křivkou nad Galoisovým tělesem $GF(p)$, kde p je prvočíslo, nazýváme množinu

$$E(GF(p)) = \{(x, y)|x, y \in GF(p), y^2 = x^3 + ax + b\} \cup \{O\},$$

kde $a, b \in GF(p)$ splňují $4a^3 + 27b^2 \neq 0$.

1.4 Vybrané modely eliptických křivek

Eliptické křivky mají více podob, takže nemusí vypadat tak, jak jsme si je uvedli v předešlé sekci. Rozdílem v těchto modelech bývá například v počtu operací při sčítání, resp. násobení bodů eliptické křivky. Právě tato vlastnost nás bude velmi zajímat, jak si ukážeme později v popisu faktorizace pomocí eliptických křivek.

1.4.1 Weierstrassův model

Základním modelem souřadnicového systému eliptických křivek je Weierstrassův model, který definuje eliptickou křivku stejně jako v definici 11:

$$y^2 = x^3 + ax + b, 4a^3 + 27b^2 \neq 0.$$

Bod na křivce máme definován jako dvojici souřadnic $P = (p_1, p_2)$. Takovéto souřadnice označíme jako **afinní**. V předešlé části naznačili, že budeme

pracovat hlavně s konečnými tělesy, tedy budeme se snažit hledat inverzi k některým hodnotám. Tato operace je jednou z nejnáročnějších operací při násobení bodů. Abychom se mohli vyhnout této operaci, tak využijeme **projektivních** souřadnic. To znamená, že *homogenizujeme* rovnici ve Weiestrassově tvaru a dostáváme tvar:

$$zy^2 = x^3 + axy^2 + bz^3.$$

Jak získat přesně tento tvar můžeme najít v [2]. Projektivní bod je pak trojice $(X : Y : Z)$ a odpovídá původnímu afinnímu bodu $(X/Z, Y/Z)$. Při převodu z projektivního bodu na afinní je tedy vyžadován výpočet inverze souřadnice Z.

Pokud hledáme řešení s nenulovým z (např. $z = 1$), pak dostáváme již známé body eliptické křivky. Pokud ale hledáme řešení se $z = 0$, pak dostáváme rovnici $0 = x^3 \Rightarrow 0 = x$, což je jediným řešením rovnice a tím získáme bod $(0 : 1 : 0)$, což je náš bod O v nekonečnu. Bod $-P$ opět získáme invertováním souřadnice Y bodu $P = (X : Y : Z)$, tedy $-P = (X : -Y : Z)$.

Pro projektivní souřadnice se nám i změní výpočet součtu dvou bodů eliptické křivky $P + Q = (X_P, Y_P, Z_P) + (X_Q, Y_Q, Z_Q) = (X_R, Y_R, Z_R)$. Kroky výpočtu nyní budou podle [4] vypadat takto:

1. $A = Y_Q Z_P$
2. $B = Y_P Z_Q$
3. $C = X_Q Z_P$
4. $D = X_P Z_Q$
5. $E = A - B$,
6. $F = C - D$,
7. $G = F^2$
8. $H = GF$
9. $I = Z_P Z_Q$
10. $J = E^2 I - H - 2GD$,
11. $X_R = FJ$,
12. $Y_R = E(GD - J) - HB$,
13. $Z_R = HI$.

Pro zdvojení bodu $2P = P + P$ pak vypočítáme jako:

1. $A = aZ_P^2 + 3X_P^2$

2. $B = Y_P Z_P$
3. $C = X_P Y_P B$
4. $D = A^2 - 8C$
5. $X_R = 2BD$
6. $Y_R = A(4C - D) - 8Y_P^2 B^2$
7. $Z_R = 8B^3$

Můžeme si všimnout, že nyní máme sice více kroků při výpočtu, ale vyhnuli jsme se počítání jakékoliv inverze. Její výpočet bude jedině v případě, kdy budeme chtít převést projektivní bod na afinní, jak už jsme si naznačili výše.

1.4.2 Montgomeryho model

Další modelem je Montgomeryho eliptická křivka, která je dle [5] definována rovnicí:

$$By^2 = x^3 + Ax^2 + x, A, B \in T, B(A^2 - 4) \neq 0,$$

kde T je těleso. Definice sčítání bodů, podle [6], $P = (x_P, y_P)$ a $Q = (x_Q, y_Q)$, $P + Q = (x_R, y_R)$ vypadá následovně:

$$x_R = B\lambda^2 - (x_P + x_Q) - A$$

$$y_R = (2x_P + x_Q + A)\lambda - B\lambda^3 - y_P = \lambda(x_P - x_R) - y_P,$$

kde hodnotu λ získáme:

$$\lambda = \begin{cases} \frac{y_Q - y_P}{x_Q - x_P} & , P \neq \pm Q \\ \frac{3x_P^2 + 2Ax_P + 1}{2By_P} & , P = Q \end{cases}$$

Montgomeryho model se dá využít k efektivnímu způsobu výpočtu bodu $mP + nP = (m + n)P$, kde $P = (X : Y : Z), nP = (X_n : Y_n : Z_n), mP = (X_m : Y_m : Z_m)$ jsou body eliptické křivky v projektovním souřadnicovém systému. Nejdříve je potřeba zmínit, že k výpočtu nepotřebujeme souřadnici Y. Výpočet definovaný v [6] je pak následující pro $m \neq n$:

1. $X_{m+n} = Z_{m-n}((X_m - Z_m)(X_n + Z_n) + (X_m + Z_m)(X_n - Z_n))^2$
2. $Z_{m+n} = X_{m-n}((X_m - Z_m)(X_n + Z_n) - (X_m + Z_m)(X_n - Z_n))^2$

Pokud $m = n$, potom výpočet pro $2P_n = P_n + P_n = (X_{2n} : Z_{2n})$ je:

1. $4X_n Z_n = (X_n + Z_n)^2 - (X_n - Z_n)^2$
2. $X_{2n} = (X_n + Z_n)^2 (X_n - Z_n)^2$

3. $Z_{2n} = (4X_nZ_n)((X_n - Z_n)^2 + (\frac{A+2}{4})(4X_nZ_n))$

Převod z projektivních souřadnic do afinních je podobný, jako tomu bylo u Weierstrassového modelu $(X : Y : Z) \to (X/Z, Y)$. Bod $-P$ získáme invertováním souřadnice Y bodu $P = (x, y)$, tedy $-P = (x, -y)$.

1.4.3 Edwardsův model

Posledním modelem, který si zde podrobněji popíšeme je Edwardsova křivka. V této části budeme vycházet z [7]. Edwardsovu křivku nad tělesem T, jehož charakteristika je různá od 2, definujeme jako rovnici:

Definice 14. (Edwardsova křivka)

$$x^2 + y^2 = c^2(1 + dx^2y^2), c, d \in T, cd(1 - c^4d) \neq 0.$$

Neutrální prvek tohoto modelu tentokrát označíme jako $O = (0, 1)$. Bod $-P$ získáme invertováním souřadnice x bodu $P = (x, y)$, tedy $-P = (-x, y)$. Definujme si nyní operaci sčítání dvou bodů $P = (x_P, y_P), Q = (x_Q, y_Q)$ za použití afinních souřadnic:

$$P + Q = (\frac{x_Py_Q + x_Qy_P}{c(1 + dx_Px_Qy_Py_Q)}, \frac{y_Py_Q - x_Py_Q}{c(1 + dx_Px_Qy_Py_Q)})$$

Pro použití projektivních souřadnic homogenizujeme rovnici popisující Edwardsovu křivku:

$$(X^2 + Y^2)Z^2 = c^2(Z^4 + dX^2Y^2).$$

Bod P označíme jako trojici $(X : Y : Z)$. Afinní bod získáme z projektivního bodu jako $(X/Z, Y/Z)$. Neutrální prvek bude mít tentokrát tvar $(0 : c : 1)$ a inverzi bodu získáme opět invertováním souřadnice X. I na tomto souřadnicovém systému se nám bude operace sčítání lišit oproti výpočtu afinních bodů. Výpočet součtu dvou různých bodů

$$P + Q = (X_P : Y_P : Z_P) + (X_Q : Y_Q : Z_Q) = (X_R : Y_R : Z_R)$$

v projektivním souřadnicovém systému podle [7] vypadá takto:

1. $A = Z_PZ_Q$,
2. $B = A^2$,
3. $C = X_PX_Q$,
4. $D = Y_PY_Q$,
5. $E = CD$,

6. $F = B - E$,
7. $G = B + E$,
8. $X_R = AF((X_P + Y_P)(X_Q + Y_Q) - C - D)$,
9. $Y_R = AG(D - C)$,
10. $Z_R = FG$,

Pro zdvojení bodu eliptické křivky poté použijeme:

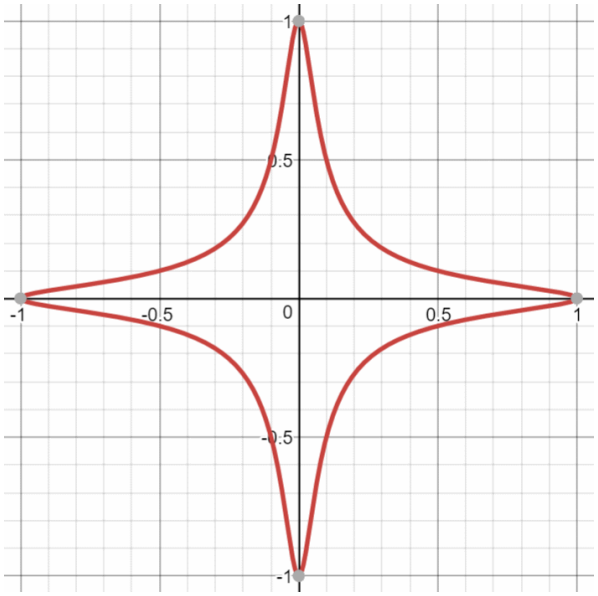
1. $B = (X_P + Y_P)^2$
2. $C = X_P^2$
3. $D = Y_P^2$
4. $F = C + D$
5. $H = Z_P^2$
6. $J = F - 2H$
7. $X_R = (B - C - D)J$
8. $Y_R = F(C - D)$
9. $Z_R = FJ$

Jak vypadají Edwardsovy eliptické křivky můžeme vidět na obrázku 1.3.

1.5 Porovnání modelů eliptických křivek

V předešlé sekci jsme si popsali a ukázali pár modelů eliptických křivek. Společně s tím jsme si taky ukázali, jak počítat s jejich body v afinním souřadnicovém systému a také v projektivním. V projektivním souřadnicovém systému si můžeme všimnout, že nám vymizela náročná část, kterou byla výpočet inverze. Nyní si popsané modely mezi sebou porovnáme. Měřítko porovnání modelů mezi sebou je počet provedených operací při sčítání, či zdvojení bodu. V tabulce 1.1 níže máme uvedeny počty násobení (označíme jako **M**), mocnění (označíme jako **S**), sčítání (označme jako **A**) a inverze (označíme **I**). Počet těchto operací spočítáme z definovaných výpočtů, které jsme si uvedli u každého souřadnicového systému eliptických křivek.

V tabulce 1.1 můžeme vidět počet operací pro afinní a projektivní souřadnicový systém a stejné body (jedná se tedy o zdvojení bodu) nebo různé body. Nejvýhodnějšími systémy tedy můžeme vidět, že pro projektivní souřadnicový systém jsou Edwardsovy a Weierstrassovy křivky. Pro Edwardsovy křivky potřebujeme sice 13 operací pro násobení, což se může zdát jako velký



Obrázek 1.3: Edwardsova křivka $x^2 + y^2 = 1 + 300x^2y^2$. (Zdroj `https://cryptobook.nakov.com/asymmetric-key-ciphers/elliptic-curve-cryptography-ecc`)

Model\Souřadnicový systém	Afinní Různé body Stejné body	Projektivní Součet bodů Zdvojení bodu
Weierstrass	6A + 2M + S + I 5A + 4M + 2S + I	6A + 12M + 2S 4A + 7M + 5S
Montgomery	7A + 3M + 2S + I 7A + 8M + 3S + I	10A + 6M + 2S 8A + 4M + 4S + I
Edwards	4A + 14M + 2I	8A + 13M + S 6A + 3M + 4S

Tabulka 1.1: Počet potřebných operací pro součet dvou bodů různých souřadnicových systémů

počet oproti Montgomeryho systému, avšak není potřeba počítat inverzi pro každé zdvojení bodu. Tomu se chceme co nejvíce vyhnout. Při počítání těchto operací se vždy zaměřujeme pouze na násobení a sčítání souřadnic mezi sebou a pokud jsme již nějaký výpočet provedli, znovu ho již nezapočítáváme.

Co je efektivnější, zda použití Weierstrassovy nebo Edwardsovy formy za použití projektivního souřadnicového systému, si ukážeme pomocí naměřených výsledků v poslední kapitole. Zde nám bude vždy záležet na tom, zda proběhne více operací zdvojení bodu ve prospěch Edwardsovy křivky nebo součet různých bodů ve prospěch Weierstrassovy křivky. Obecně se uvádí jako efek-

tivnější použití Edwardsových křivek (viz [8]), avšak zde používají takzvaný rozšířený souřadnicový systém, pomocí kterého pracují nad podkřivkách a tím výpočet urychlí a zefektivní. Lepší výsledky za použití Edwardsových křivek ukazuje dostupná implementace EECM-MPFQ[1]. Ta však využívá již zmíněného rozšířeného souřadnicového systému.

Modelů pro eliptické křivky je více (například Jacobiho nebo Hessianovský model). Další modely společně s porovnáním náročnosti výpočtů můžeme nalézt v [7]. V této práci se budeme ale hlavně zabývat Weierstrassovými a Edwardsovými křivkami. Naše výpočty budou probíhat zejména v projektivních souřadnicích.

1.6 Přibližný počet operací Weierstrassova a Edwardsova modelu

Na závěr této kapitoly si ukážeme, jaký je přibližný počet operací za použití Weierstrassova a Edwardsova modelu. Tento přibližný počet získáme tak, že vybereme pár násobků bodu P a vypočítáme kolikrát budeme násobit a sčítat souřadnice mezi sebou. Počet operací pak využijeme pro naši analýzu výsledků v poslední kapitole.

Hodnota násobku	Weierstrass	Edwards
2000 (6, 11)	80A + 138M + 67S	114A + 94M + 50S
10000 (5, 14)	86A + 144M + 80S	124A + 88M + 61S
100000 (6, 17)	104A + 174M + 97S	150A + 106M + 74S

Tabulka 1.2: Přibližný počet násobení, sčítání a mocnění pro Weierstrassův a Edwardsův model při násobení hodnotou k v projektivním souřadnicovém systému. V závorce u hodnoty násobku je určen celkový počet součtů a zdvojení.

V tabulce 1.2 máme přibližný počet operací potřebný pro vynásobení bodu eliptické křivky hodnotou násobku. Můžeme si všimnout, že u Edwardsovy křivky máme velký počet součtů, avšak oproti Weierstrassova modelu máme méně násobení a mocnění, což by měly být náročnější operace. Toto však není faktor, který nám může zpomalit výpočet, bude nám také záležet zvolení eliptické křivky. Tuto závislost si popíšeme při popisu a volby parametrů Lenstrova algoritmu.

[1] `https://eecm.cr.yp.to/`

Kapitola 2

Problém faktorizace a RSA

V této kapitole si nejdříve popíšeme problém faktorizace a ukážeme příklad jedné šifry, která je založena na problému faktorizace, která je používána dodnes. Ukážeme si zde pouze základní formu této šifry. V praxi je pak šifra více optimalizována, ale pro náš účel bude stačit právě pouze její základní forma, abychom si na ní ukázali problém faktorizace.

Tato šifra se jmenuje RSA. V 70. letech ji navrhli Ron Rivest, Adi Shamir a Leonard Adelman a šifra nese jméno právě podle nich, respektive název se skládá z iniciálů autorů. Je řazena do kategorie asymetrických šifer, tedy používá dva navzájem různé klíče. Jeden klíč se používá pro šifrování (veřejný) a druhý klíč (soukromý) pro dešifrování. Tato šifra je dnes stále využívána a při dostatečně dlouhém klíči, jak se můžeme dočíst v [9], je stále bezpečná.

2.1 Problém faktorizace

Základní věta aritmetiky, jak je uvedena v [10], nám říká, že každé přirozené číslo větší než 1 je buď prvočíslem, nebo je možné zapsat jako součin dvou či více prvočísel. Takový rozklad je přitom jednoznačně dán až na pořadí činitelů. **Faktorizací** nazýváme právě rozklad přirozeného čísla na součin prvočísel. Pro faktorizaci čísel neexistuje efektivní algoritmus, který by to dokázal v přijatelném čase.

Velikost prvočísla může být až $\sqrt{n}$, kde n je faktorizované číslo. Tím by se mohlo zdát, že faktorizace může proběhnout naivním způsobem v $O(\sqrt{n})$ čase, avšak je nutné si uvědomit, že hodnota n bude přibližně stejně velká jako 2^b, kde b je počet bitů, které jsou potřeba pro zápis čísla n. Budeme tedy zkoušet všechny různé kombinace bitů, což je $O(\sqrt{2^b}) = O(2^{\frac{b}{2}})$, tedy exponenciální složitost.

2.2 Symetrické a asymetrické šifry

Dříve než začneme s popisem konkrétního algoritmu, tak si uvedeme základní rozdělení šifer. Zde budeme vycházet ze studijních materiálů[2] k předmětu *Bezpečnost* (BI-BEZ), který se vyučuje na FIT ČVUT.

2.2.1 Symetrické šifry

Symetrické šifry pracují pouze s jedním klíčem, který je určen pro šifrování a dešifrování. Pokud vyžadujeme šifrovanou komunikaci například mezi dvěma zařízeními, tak je nutné si nejdříve domluvit společný klíč, který se bude právě používat pro šifrování a dešifrování. K tomu můžeme využít například algoritmus Diffieho–Hellmana pro výměnu klíčů, který zajistí, že po nešifrovaném kanálu si můžeme domluvit tajný klíč.

Výhoda těchto šifer spočívá v tom, že nejsou příliš náročné na různé výpočty. Nevýhoda naopak je právě domluva tajného klíče, kdy může být tajný klíč odposlechnut a zneužit k dešifrování tajných zpráv.

Mezi známé symetrické šifry například patří:

- **AES** – dnes jeden z nejvíce využívaných a nejbezpečnějších algoritmů,
- **3DES** – dříve používaná šifra, kterou i dnes je možné nalézt ve starších systémech, avšak už se od ní ustupuje,

2.2.2 Asymetrické šifry

Asymetrické šifry naopak využívají různých klíčů pro šifrování a dešifrování. Klíč pro šifrování většinou nazýváme veřejným klíčem, pro dešifrování pak nazýváme jako soukromý. Není tedy potřeba využívat různých technik, jak si domluvit tajný klíč, ale můžeme jednoduše zaslat svůj veřejný klíč.

Výhodou těchto šifer je, že soukromý klíč nikde nedistribuujeme, takže útočník pak hůř bude získávat klíč k dešifrování. Nevýhodou však je, že je potřeba provést více výpočtů a tím se stávají tyto šifry pomalejší oproti symetrickým šifrám.

Mezi známými zástupci asymetrických šifer jsou:

- **RSA** – dnes běžná šifra, která se využívá společně se symetrickými šiframi, níže si jí popíšeme více podrobněji. Tato šifra se zakládá na problému faktorizace,
- **ElGamal** – je méně využívaná šifra oproti RSA, protože šifrovaná data jsou dvakrát delší než data šifrovaná. Tato šifra se zakládá na problému výpočtu diskrétního logarimu.

[2] `https://courses.fit.cvut.cz/BI-BEZ`

2.3 Princip RSA

Pro úplnost si nejdříve zavedeme definici Eulerovy funkce, která je nutná pro princip algoritmu RSA.

Definice 15. (Eulerova funkce) Eulerova funkce $\varphi(n) : \mathbb{N} \to \mathbb{N}$, kde $n \in \mathbb{N}$, udává počet kladných celých čísel menších nebo rovných n, která jsou nesoudělná s n.

Definice 16. (Eulerova věta) Nechť je dáno $n \in \mathbb{N}$ a $a \in \mathbb{N}$ takové, že $\gcd(a, n) = 1$, potom platí:

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

Jedná se o zobecnění Malé Fermatovy věty[3].

Nyní si ukážeme, jak se generují klíče pro šifrování a dešifrování:

1. Zvolíme náhodně dvě prvočísla p a q.
2. Následně vypočítáme $n = pq$.
3. Vypočítáme hodnotu Eulerovy funkce $\varphi(n)$. Tato hodnota bude jistě rovna $(p-1)(q-1)$.
4. Zvolíme číslo e, pro které platí: $1 < e < n$ a $\gcd(\varphi(n), e) = 1$.
5. Vypočítáme číslo d, pro které platí $de \equiv 1 \pmod{\varphi(n)}$

Veřejným klíčem nazýváme dvojici $V_k = (n, e)$ a soukromý klíč nazýváme dvojici $S_k = (n, d)$. Podle [9] je vhodné, aby číslo n mělo nejméně 2048 bitů.

Veřejný klíč může být rozdistribuován komukoliv po nešifrovaném kanálu, neboť veřejným klíčem můžeme zprávu zašifrovat, ale již není možné jím zprávu dešifrovat. Z toho tedy vyplývá, že soukromý klíč musíme mít uchován pouze u sebe a nikam ho nedistribuovat.

2.4 Šifrování a dešifrování komunikace pomocí RSA

Nyní si ukážeme, jak pomocí RSA lze zprávu šifrovat. Tuto operaci ukážeme na běžném příkladu, kdy Alice a Bob si chtějí vyměnit zprávu po nešifrovaném kanálu.

Nejdříve si Bob vygeneruje svojí dvojici veřejného a soukromého klíče, nazvěme si tuto dvojici:

$$V_B = (n_B, e_B) \text{ a } S_B = (n_B, d_B)$$

[3] $a^{p-1} \equiv 1 \pmod{p}$, pro prvočíslo p a $\gcd(p, a) = 1$

Alice a Bob si vymění své veřejné klíče, tedy Alice zná pouze V_B. Uvažujme konkrétní hodnoty:

$$V_B = (65, 5), S_B = (65, 29)$$

Alice bude chtít Bobovi poslat zprávu O_T. Pro jednoduchost chceme poslat $O_T = 10$. Tuto zprávu tedy zašifruje pomocí veřejného klíče Boba V_B. Šifrovanou zprávu získáme jako

$$C_T = O_T^{e_B} \bmod n_B$$

$$C_T = 10^5 \bmod 65 = 30$$

Bob přijme C_T a začně zprávu dešifrovat.

$$O_T = C_T^{d_B} \bmod n_B$$

$$O_T = 30^{29} \bmod 65 = 10$$

Zprávu jsme úspěšně tedy dešifrovali, analogicky pokud bude chtít Bob poslat stejnou zprávu Alici, tak využije jejího veřejného klíče.

2.5 Využití RSA

V [11] můžeme nalézt informace ohledně využití RSA a že se dneska také využívá společně s ostatními symetrickými šiframi. Využívá se například pro digitální podpisy zpráv. Podpis je vytvořen tak, že ke zprávě je přidána takzvaná haš, která se pomocí soukromého klíče *zašifruje*. Na druhé straně je potom možné právě veřejným klíčem haš opět dešifrovat a můžeme si tak porovnat, že jsme přijali nepodvrženou zprávu.

Dalším využitím může být také již zmíněná výměna klíčů, kde zúčastněné strany pro šifrovanou komunikaci se drží těchto kroků:

1. Zúčastněné strany si vygenerují své dvojice klíčů (veřejný a soukromý).

2. Zúčastněné strany si rozešlou navzájem veřejné klíče.

3. Každý jedinec si vygeneruje klíč, který chce sdílet s ostatními.

4. Každý vygenerovaný klíč zašifruje veřejným klíčem příjemce a zašle ho takto všem.

5. Jakmile mají všichni zúčastnění rozeslané všechny tajné klíče, tak si je podle předem daného pravidla zřetězí a vytvoří tím společný tajný klíč.

2.6 Útoky vedené na RSA

V této práci se zabýváme faktorizačním algoritmem. Jak jsme již zmínili v úvodu, pokud by se nám povedlo faktorizovat číslo v přijatelném čase, tak bychom prolomili bezpečnost této šifry a kryptosystém by nebyl již bezpečný a naše šifrovaná komunikace by mohla být jednoduše rozluštěna.

Na tento kryptosystém jsou vedené i jiné útoky než pomocí faktorizace čísel. Ty si v této části mimojiné také alespoň z části nyní nastíníme, jak fungují. Při popisu vycházíme z přednášek k předmětu MI-KRY [12].

2.6.1 Faktorizace

Jak jsme již popsali v předešlé části, veřejný klíč obsahuje číslo n. Pokud bychom byli schopni toto číslo faktorizovat, tedy získat prvočísla p a q, z kterého je n složeno. Tím bychom potom byli schopni dopočítat i soukromý klíč, tedy dvojici (n, d), jelikož známe veřejný klíč, který je právě dvojicí (n, e).

2.6.2 Časový útok

Tento útok je založen na měření času určitých výpočetních operací, které jsou potřeba při šifrování nebo dešifrování. Při těchto úkonech počítáme modulární mocninu. K tomu může být využit například algoritmus *square and multiply* popsaný v algoritmu 1.

Algorithm 1 Pseudokód SquareAndMultiply

```
1:  function SquareAndMultiply(exponent e, number m, modulus n)
2:                                          ▷ Chceme vypočíst m^e (mod n)
3:      l = e
4:      tmp = m
5:      ret = 1
6:      while l ≠ 0 do
7:          if d_i = 1 then ret = |tmp · ret|_n
8:          end if
9:          tmp = |tmp^2|_n
10:         l = ⌊l/2⌋
11:     end while
12:     return ret
13: end function
```

Měříme časové odchylky v každém cyklu. Vždy když časová odchylka bude delší, signalizuje nám to, že daný bit je nastaven na hodnotu 1. V opačném případě je nastaven na 0. Je však také nutné odhadnout, v jakých intervalech daná smyčka probíhá.

2.6.3 Odběrová analýza

Další možností jak získat utajovaný klíč je pomocí odběrové analýzy. Zde máme nejméně dva způsoby, jak odběrovou analýzu provést. Prvním způsobem je jednoduchá odběrová analýza, kde útok probíhá tak, že si změříme spotřebu při dešifrování. Následně zanalyzujeme spotřebu. Tam, kde modul počítal nejvíce, bude v přijatém signálu mít velkou spotřebu, kde méně, tam naopak menší. Je opět důležité si nalézt v odebraném signálu, kde začíná cyklus při výpočtu. Druhým způsobem je diferenciální odběrová analýza. Zde útok probíhá tak, že změříme více průběhů šifrování nebo dešifrování a zanalyzujeme signál ve stejných okamžicích v čase napříč všemi průběhy.

Kapitola 3

Popis faktorizačních algoritmů

Nyní si popíšeme některé známé faktorizační algoritmy a uvedeme si příklady, jak pomocí nich faktorizovat. Nejdříve začneme s popisem algoritmů, na které se v této práci snažíme zaměřit. Na konci této kapitoly pak popíšeme další známé faktorizační algoritmy.

3.1 Pollardova $p-1$ metoda

Začněme nejdříve s Pollardovou $p-1$ metodou, pro snažší pochopení Lenstrova algoritmu, jež se na této Pollardově $p-1$ metodě zakládá. Pollardova $p-1$ metoda, kterou publikoval J. M. Polard v [13], se zakládá na Malé Fermatově větě[4]. Snažíme se využít znalost toho, co víme o řádu nějakého prvku z multiplikativní grupy $\mathbb{Z}_p$, abychom našli faktor p našeho faktorizovaného čísla $n = pq$. Hlavní myšlenka této metody tedy spočívá v tom, že pokud $p \mid n$, číslo p je prvočíslo a $a \in \mathbb{N}$, potom platí, že pro $a^{p-1} \equiv 1 \pmod{p}$. V důsledku předešlého tvrzení také platí $\gcd(n, a^{p-1} - 1 \pmod{p}) = p$.

Nyní se potřebujeme zaměřit na to, jak takové $p-1$ najít. Předpokládejme nějaké $L \in \mathbb{N}$, které splňuje $(p-1) \mid L$ a $(q-1) \nmid L$. Potom existují nezáporná celá čísla $i, j, k, k \neq 0$ taková, že

$$L = i(p-1), L = j(q-1) + k$$

J. M. Pollard navrhuje, že pro $m = 2, 3, \ldots$ je vhodné volit $a \in \mathbb{N}$ a spočíst $d = \gcd(a^{m!} - 1, n)$, protože pokud $p-1$ je součinem malých prvočísel, potom bude $p-1$ dělit $m!$ již pro relativně malé m. Pro d nám vyvstávají tři možnosti. Pokud $d = 1$, pokračujeme k vyššímu m, pokud $d = n$, d je násobkem n a volíme jiné a, pokud $1 < d < n$, potom d je faktorem n.

Algoritmus 2 nám opět popisuje, jak se celá tato metoda dá přepsat do pseudokódu. Existují i jiné pseudokódy, jak tento algoritmus zapsat a získat stejný výsledek. K vidění může být například v [14].

[4] $a^{p-1} = 1 \mod p, \gcd(a, p) = 1, a \in \mathbb{Z}$, pro prvočíslo p

Algorithm 2 Pseudokód Pollardovy $p-1$ metody

```
1: function PollardP(n)
2:     Vyber libovolné a ∈ {2, 3, ..., n − 1}
3:     d = 1
4:     r = 1
5:     while d = 1 do
6:         d = gcd(n, a^{r!} − 1 (mod n))
7:         r = r + 1
8:     end while
9:     if d ≠ n then
10:        return d
11:    end if
12:    goto 2
13: end function
```

3.1.1 Příklad použití

r	$a^{r!}-1 \pmod n$	$\gcd(n, a^{r!}-1 \pmod n)$
1	1	1
2	3	1
3	63	1
4	26	13

Tabulka 3.1: Průběh výpočtu Pollardovy $p-1$ metody pro $a = 2$

Uveďme opět příklad, kde budeme faktorizovat číslo $n = 143 = 11 \cdot 13$. Zvolme si proměnnou $a = 2$. V tabulce 3.1 můžeme vidět, jak bude vypadat průběh výpočtu. Výpočet hodnoty $a^{r!} - 1$ můžeme počítat v modulu n.

3.2 Lenstrův algoritmus

Než začneme s popisem Lenstrova algoritmu, uvedeme si na začátek jednu důležitou větu.

Tvrzení 3.2.1. (Hasseho věta) Buď E eliptická křivka nad $GF(p)$, pro $p \in \mathbb{N}$. Potom

$$p + 1 - 2\sqrt{p} \leq \#E(GF(p)) \leq p + 1 + 2\sqrt{p}.$$

Důkaz této věty si zde nebudeme psát, avšak více k této větě můžeme najít v [2]. Důležitou roli pro nás bude hrát při určení řádu grupy $E(GF(p))$, kterou využijeme při popisu algoritmu níže.

Jak je popsáno v [15], tak Lenstrův algoritmus vychází z předešlé Pollardovy $p-1$ metody, která pracuje s prvky z okruhu $\mathbb{Z}_n$. Lenstrova metoda

pracuje s prvky grupy, kterou tvoří body eliptické křivky. Namísto mocnění náhodné hodnoty a (viz algoritmus 2 uvedený výše) budeme násobit bod P, ležící na dané eliptické křivce, nějakou hodnotou $k \in \mathbb{N}$. Podobně jako tomu je u Pollardovy $p-1$ metody, pokud řád grupy $\#E(GF(p)))$ dělí násobek k, potom $kP = O$, což znamená, že jsme našli dělitele našeho faktorizovaného čísla n. Rozdílem je reakce na to, jestli násobek k nedělí řád grupy $\#E(GF(p)))$. V takovém případě zvolíme jinou eliptickou křivku E', čímž se nám změní i řád grupy $\#E'(GF(p)))$.

Jak budeme napočítávat násobky nějakého bodu P eliptické křivky jsme si již ukázali v 1. kapitole. Je důležité si ale ještě ukázat, jak poznat, že jsme narazili na dělitele. Nyní předpokládejme, že pracujeme s eliptickou křivkou ve Weierstrassově formě a s afinními souřadnicemi. Dále co se nám ještě změní, nebudeme pracovat s tělesem $GF(p)$ nad eliptickou křivkou, ale pouze s okruhem $\mathbb{Z}_n$, kde n je naše faktorizované číslo. Sice nebudeme mít nyní zajištěno, že pokaždé najdeme inverzi, ale to přesně Lenstrův algoritmus vyžaduje. Pro afinní souřadnicový systém můžeme počítat dvě inverze, jak máme definováno v definici 12 a to buď $(q_1 - p_1)^{-1}$, nebo $(2p_2)^{-1}$. Pokud tyto inverze v okruhu není možné nalézt, potom jsme nejspíše narazili na dělitele našeho faktorizovaného čísla n, neboť $\gcd(q_1 - p_1, n) \neq 1$ anebo $\gcd(2p_2, n) \neq 1$. Tímto mechanismem dospějeme k nalezení dělitele.

S tímto popisem si nyní zaveďme kroky faktorizačního algoritmu pro faktorizaci nějakého čísla $n \in \mathbb{N}, n \geq 2$ nad eliptickou křivkou ve Weierstrassově formě, avšak nyní tedy nad okruhem $\mathbb{Z}_n$:

1. Vyberme si náhodné hodnoty $A, x_P, y_P \in \mathbb{Z}_n$.
2. Položme $B = y_P^2 - x_1^3 - Ax_1 \mod n$. Potom naše křivka je $E : y^2 = x^3 + Ax + B$
3. Ověříme, že křivka je nesingulární, tedy $\gcd(4A^3 + 27B^2, n) = 1$. Pokud se největší společný dělitel rovná číslu n, potom se vracíme k 1. kroku. Pokud je hodnota mezi 1 a n, potom jsme nalezli dělitele.
4. Zvolme si hodnotu $K \in \mathbb{N}$ a položme $k = \mathrm{lcm}(1, 2, \ldots, K)$, jak jsme si definovali v 12.
5. Vypočítejme násobek bodu $Q = kP$. Pokud nový bod Q leží na křivce, můžeme se vrátit k prvnímu kroku, případně zvýšit hodnotu k. Pokud výpočet selhal, poté jsme našli hodnotu d, která nemá inverzi v daném okruhu $\mathbb{Z}_n$. Pokud $d = n$, pak se vracíme ke kroku 1 nebo snížíme hodnotu k. Pokud $1 < d < n$, pak jsme nalezli dělitele.

3.2.1 Využití projektovního souřadnicového systému

Při popisu Lenstrova algoritmu jsme si uvedli, že používáme afinní souřadnicový systém. V 1. kapitole ale zmiňujeme projektivní souřadnicový systém,

který je efektivnější z důvodu redukce počítání inverze. Zde si ukážeme menší změnu původního algoritmu pro použití projektivního souřadnicového systému.

V sekci 1.4.1 jsme si uvedli, jak převádět projektivní bod na afinní. Pro připomenutí převod z projektivního souřadnicového systému do afinního je $(X, Y, Z) \to (X/Z, Y/Z)$. Této inverze při převodu právě budeme chtít využít. Kroky algoritmu tedy nyní budou dost podobné jako v předešlé části, avšak budeme mít o jeden krok navíc. Abychom neduplikovali popis, zkrátíme popis kroků, které již známe:

1. Zvolíme si křivku E a bod $P = (x_P, y_P, 1)$ na ní a ověříme nesingularitu. Opět první dvě souřadnice jsou náhodné a přidáme 3. souřadnici $Z = 1$.
2. Zvolme si hodnotu $K \in \mathbb{N}$ a položme $k = \mathrm{lcm}(1, 2, \ldots, K)$.
3. Vypočítáme k–tý násobek bodu P, jak jsme si definovali v sekci 1.4.1.
4. Vypočítáme inverzi souřadnice Z. Položíme $d = gcd(Z, n)$. Pokud $d = 1$, tak se buď vrátíme k 1. kroku, nebo zvolíme vyšší hodnotu k. Pokud $d = n$, potom volíme nižší hodnotu k nebo opět vygenerujeme novou eliptickou křivku a bod na ní. Zbývá nám případ, kdy $1 < d < n$. Tím jsme našli prvočinitele složeného čísla n a algoritmus končí.

3.2.2 Lenstrův algoritmus a Edwardsovy křivky

Nyní máme popsaný základní tvar Lenstrova algoritmu pro eliptickou křivku ve Weierstrassově formě. Pro Edwardsovu křivku bude postup velmi podobný. Pro tvar Edwardsovy křivky zvolíme $c = 1$ v rovnici, kterou jsme si definovali v sekci 1.4.3. Budeme tedy používat rovnici ve tvaru

$$x^2 + y^2 = 1 + dx^2y^2, d(1 - d) \neq 0$$

Tento tvar používáme podle doporučení z [7]. Sepíšeme si základní tvar algoritmu za použití afinních souřadnic, který se nám liší pouze tvarem křivky:

1. Vybereme si náhodné hodnoty $x, y \in \mathbb{Z}_n \setminus \{0, 1\}$.
2. Položíme $d = \frac{x^2+y^2-1}{x^2y^2} \in \mathbb{Z}_n$. Pokud $d \in \{0, 1\}$, vrátíme se ke kroku 1.
3. Zvolme si hodnotu $K \in \mathbb{N}$ a položme $k = \mathrm{lcm}(1, 2, \ldots, K)$.
4. Vypočítáme k–tý násobek bodu P, jak jsme si definovali v sekci 1.4.3.
5. Zde je krok stejný jako tomu je v původním algoritmu. Pokud výpočet k–tého násobku proběhl úspěšně, potom zvýšíme k, nebo přejdeme na krok 1. Pokud výpočet byl neúspěšný, pak nám opět vyvstávají dvě možnosti. Pokud inverze $d = n$, pak snížíme hodnotu k, nebo přejdeme ke kroku 1. Poslední možností je, že $1 < d < n$, a tím jsme nalezli prvočinitele čísla n.

3.2.3 Volby parametrů

Popíšeme si ještě volbu parametrů u Lenstrova algoritmu pro varianty s Weierstrassovými a Edwardsovými modely eliptické křivky. Bod $P = (x_P, y_P)$ volíme vždy náhodně. Je to z toho důvodu, že je to jistě jednodušší způsob, než vybírat nejdříve eliptickou křivku a poté na ní hledat bod, což by znamenalo hledání některého z řešení rovnice popisující eliptickou křivku. Edwardsova křivka navíc požaduje, aby souřadnice bodů nebyly rovny hodnotám 0 nebo 1. Pokud se tak stane, generujeme nové hodnoty. Při generování bodů můžeme také v rámci zkoušky urychlení výpočtu zkusit, jestli vygenerované hodnoty nejsou dělitelem faktorizovaného čísla. To se nám ovšem neděje nijak často, neboť by pak bylo jednodušší náhodně generovat čísla a ty zkoušet jako kandidáty dělitelů složeného čísla.

Edwardsova křivka požaduje také hodnotu $d = \frac{x^2+y^2-1}{x^2y^2} \notin \{0, 1\}$. Pokud by nám po generování x, y stále vycházela hodnota rovna 0 nebo 1, pak by bylo potřeba se zamyslet nad jiným způsobem získání hodnoty d. To ovšem také nebude nastávat velmi často, což si ověříme naší implementací. Existují však i jiné způsoby (viz [8]), jak volit hodnotu d, například tak, že se dosadí vhodná hodnota d a následně se pak hledá bod na této křivce.

Tyto zmíněné volby parametrů jsou nejen nejjednodušší, jak už jsme si zmínili, ale také nejrychlejší. Samozřejmě to ale má naopak svá úskalí. Pokud takto náhodně volíme bod a k němu *dopočítáme* křivku, tak nijak nezajišťujeme, že řád vygenerované křivky bude mít nějakou minimální hodnotu. Metody pro hledání křivek vyššího řádu (vyšší než námi zvolená minimální hranice) můžeme najít v [16]. Pro nás bude však postačující, pokud budeme generovat parametry náhodně, neboť i v základním popisu, uvedeném v [15], je řečeno, že se řád náhodně vygenerované křivky pohybuje v intervalu uvedeným Hasseho větou 3.2.1, tedy je mezi $p + 1 - 2\sqrt{p}$ a $p + 1 + 2\sqrt{p}$. Tímto náhodným výběrem však můžeme uškodit Edwardsově formě, neboť nám bude velmi záležet také na parametru d. Například implementace popsaná v [8] generuje parametr d tak, aby splňoval ještě další podmínky, které zvyšují šanci rychlejšího nalezení dělitele složeného čísla. Zde však používají, jak jsme si již dříve zmínili, rozšířený souřadnicový systém. Nám však jde v této práci o porovnání projektivního souřadnicového systému.

3.2.4 Příklad použití

Nyní si uvedeme příklad, jak vypadá průběh výpočt. Využijeme klasickou eliptickou křivku ve Weierstrassově formě a afinním souřadnicovým systémem. Kroky můžeme vidět v tabulce 3.2. Zvolme si jako číslo $n = 221 = 13 \cdot 17$.

3.2.5 Časová složitost

Časová složitost tohoto algoritmu je $O(e^{\sqrt{2((\ln n)(\ln \ln n))}})$. Její odvození můžeme nalézt v [14].

i	$P(x, y)$	$\gcd(d, n)$
1	$(116, 75)$	1
2	$(44, 53)$	1
3	$(157, 100)$	1
4	$(89, 36)$	1
5	SELHALO	17

Tabulka 3.2: Faktorizace čísla $n = 221$ pomocí Lenstrova algoritmu pro eliptickou křivku s $a = 47, b = 200$

3.3 Další faktorizační algoritmy

V této části si ukážeme a alespoň nastíníme, jak fungují další známé faktorizační algoritmy. Společně s tím jsou i zde ukázány jejich časové složitosti a jak si budeme moci povšimnout, jejich časové složitosti můžou být i lepší než časová složitost Lenstrova algoritmu.

3.3.1 Pollardova ρ–metoda

Nejdříve začneme s jednou z jednodušších metod. Tato metoda byla publikována již v 70. letech 20. století jako velmi efektivní pro faktorizaci čísel v [17]. Efektivní z toho důvodu, jakou má časovou složitost, kterou si níže ukážeme.

Mějme číslo n, které chceme faktorizovat. Dále mějme množinu $M = \{0, 1, \ldots, n-1\}$ a zobrazení $f : M \to M$. Snažíme se generovat body posloupnosti pomocí zobrazení f tak, že si zvolíme počáteční bod posloupnosti libovolné $x_0 \in M$ a další body definujeme jako $x_i = f(x_{i-1})$ pro $i \in \mathbb{N}$. Po získání těchto hodnot hledáme největšího společného dělitele absolutní hodnoty rozdílu těchto bodů a faktorizovaného čísla n, tedy $d = \gcd(|x_{2i} - x_i|, \mathrm{n})$. Zde nám mohou nastat tři možnosti:

1. $d = 1$, potom pokračujeme k výpočtu další dvojice bodů posloupnosti,
2. $d = n$, potom číslo d je násobkem faktorizovaného čísla n a musíme zvolit nové x_0 a začít s výpočtem od začátku,
3. $1 < d < n$, potom jsme našli faktor čísla n.

Algoritmus 3 popisuje pseudokódem, jak tato metoda lze realizovat. Vstupem je číslo n, které chceme faktorizovat a zobrazení f, pro generování bodů posloupnosti. Toto zobrazení f definujeme jako $f(x) = x^2 + c \mod n, c \notin \{0, 2\}$.

Algorithm 3 Pseudokód Pollardovy ρ–metody

```
1: function PollardRho(n, f)
2:     Vyber libovolné x ∈ {0, 1, ..., n − 1} ▷ Zde si uchováváme hodnotu x_i
3:     y = x                                  ▷ Zde si uchováváme hodnotu x_2i
4:     divisor = 1
5:     while divisor = 1 do
6:         x = f(x) (mod n)
7:         y = f(f(y)) (mod n)
8:         divisor = gcd(|y − x|, n)
9:     end while
10:    if divisor ≠ n then return divisor
11:    end if
12:    goto 2
13: end function
```

3.3.1.1 Příklad použití

Uveďme příklad. Snažíme se faktorizovat číslo $n = 1517 = 37 \cdot 41$. Jako zobrazení f si zvolíme polynom $f(x) = x^2 + 1$ (mod n). Dále zvolíme počáteční hodnotu $x_0 = 2$. V tabulce 3.3 můžeme vidět průběh výpočtu.

Číslo kroku i	x_i	x_{2i}	$\lvert x_{2i} - x_i \rvert$	gcd($\lvert x_{2i} - x_i \rvert$, n)
1	5	26	21	1
2	26	196	170	1
3	677	862	185	37

Tabulka 3.3: Průběh výpočtu Pollardovy ρ – metody

Časová složitost tohoto algoritmu je $O(n^{1/4})$. Odvození této složitosti můžeme najít v [2].

3.3.2 Kraitchikovo schéma

Nyní si popíšeme také faktorizaci čísel pomocí číselných sít, avšak než začneme s popisem konkrétních sít, ukážeme si, jak obecně fungují. Jak se můžeme dozvědět z [18], tak tyto algoritmy vycházejí z takzvaného Kraitchikova schématu. Idea spočívá v tom, že se budeme snažit násobit kongruence typu $U \equiv V \pmod{n}$, kde $U \neq V$ a číslo n je to, které chceme faktorizovat. Jejich produktem získáme speciální kongruenci $X^2 \equiv Y^2 \pmod{n}$ a z této kongruence je jistá šance, že největší společný dělitel čísel $(X \pm Y, n)$ bude netriviálním faktorem čísla n. V bodech si zapišme kroky těchto algoritmů:

1. Vygeneruj kongruence $U \equiv V \pmod{n}$,

2. Stanovení úplné nebo částečné faktorizace U a V pro některé z kongruencí,

3. Stanovení podmnožiny faktorovaných kongruencí, které mohou být roznásobeny k získání speciální kongruence $X^2 \equiv Y^2 \pmod{n}$,

4. Výpočet největšího společného dělitele dvojice čísel $(X - Y, n)$.

Pro kompletnost si uvedeme příklad. Chceme faktorizovat číslo $n = 33 = 3 \cdot 11$. Budeme postupovat tedy podle popsaných vygenerujeme si kongruence splňující bod 1. Z nich se zaměříme na tyto kongruence[5]:

$$32 \equiv -1 \pmod{n},\ 8 \equiv -25 \pmod{n}$$

Nyní přejdeme k bodu 2. Tyto kongruence můžeme zapsat jako:

$$2^5 \equiv -1 \pmod{n},\ 2^3 \equiv -5^2 \pmod{n}$$

Pokud tyto dvě kongruence roznásobíme (bod 3), získáme:

$$2^5 \cdot 2^3 \equiv -1(-5^2) \pmod{n} \Rightarrow 2^8 \equiv 5^2 \pmod{n}$$

A na závěr z bodu 4 nám potom plyne $11 = \gcd(2^4 - 5, 33)$, tedy nalezli jsme faktor, kterým je číslo 11.

Pokud chceme používat toto schéma, je nutné, aby faktorizované číslo nebylo perfektním čtvercem nebo prvočíslo. Důležité je také si uvědomit, že generování potřebných kongruencí může být výpočetně náročné. To se nám ovšem snaží zjednodušit číselná síta, která si nyní popíšeme.

3.3.3 Kvadratické síto

Na úvod si zde uvedeme definice pro úplnost.

Definice 17. (Hladké čísla, hladké polynomy, faktorová báze) Mějme $n \in \mathbb{Z}$. Číslo n nazveme B–hladké, jestliže má všechny prvočíselné dělitele menší nebo rovny B. Mějme dán polynom f nad konečným tělesem. Polynom f nazveme B–hladký, jestliže jej lze faktorizovat na ireducibilní polynomy stupně maximálně B. Mějme dán okruh R a prvek $x \in R$. Říkáme, že prvek x je hladký nad faktorovu bází $F = \{p_1, p_2, \ldots, p_t\} \subset R$, pokud prvek x lze vyjádřit jako součin prvků z množiny F.

Definice 18. (Kvadratický zbytek) Číslo $a \in \mathbb{Z}_k, k \in \mathbb{N}, k \geq 2$ nazveme **kvadratickým zbytkem**, pokud existuje celé číslo x takové, že $x^2 \equiv a \mod k$

[5]Mohli bychom vybrat i jiné vyhovující kongruence, avšak z těchto získáme výsledek

Definice 19. (Legendreův symbol) Nechť p je liché prvočíslo a $a \in \mathbb{Z}$. Legendreův symbol $\left(\frac{a}{p}\right)$ pak definujeme jako:

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{pro } p|a \\ 1 & \text{pokud } a \text{ je kvadratickým zbytkem a } p \nmid a \\ -1 & \text{pokud } a \text{ není kvadratickým zbytkem} \end{cases}$$

Kvadratické síto popsal Carl Pomerance v [18]. Rozšířil tím výše zmíněné Kraitchikovo schéma a využil Dixonova algoritmu. Tento algoritmus se považoval za nejrychlejší velmi dlouhou dobu, protože nepoužívá výpočetně náročné operace jako je tomu v případě Lenstrova algoritmu při násobení bodu.

Kvadratické síto se snaží hledat kongruence v určitém tvaru a pro nějakou hranici. Popišme si tedy jak to funguje. Jak již bylo zmíněno tento algoritmus vychází z Kraitchikova schématu, pro které je důležité, aby faktorizované číslo nebylo perfektním čtvercem a prvočíslo. Kvadratické síto používá pro hledání kongruencí kvadratický polynom:

$$Q(x) = (x + \lfloor\sqrt{n}\rfloor)^2 - n$$

Našim cílem je získat tuto kongruenci:

$$Q(x_{i_1})Q(x_{i_2}) \cdots Q(x_{i_r}) \equiv (x_{i_1} x_{i_2} \cdots x_{i_r})^2 \pmod{n}$$

Nyní si ukážeme, jak takový algoritmus celý vypadá a na něm si popíšeme, co z výše uvedené kongruence znamenají nepopsané členy x_i a r.

V pseudokódu 4 můžeme vidět fungování tohoto algoritmu. Jsou zde dvě hodnoty B a M. Jejich přesné hodnoty si určíme níže, v popisu časové složitosti. Berme je nyní pouze jako nějaké hranice. V kroku 2 vidíme, že nejdříve je nutné si vygenerovat faktor bázi. Ta je sestavena z prvočísel, která jsou menší než nějaká hranice B a zároveň Legendreův symbol čísel n a prvočísla je roven 1. Následně hledáme takové relace $Q(x_i)$, které jsou hladké nad faktor bází neboli $Q(x_i) = p_1^{k_1} p_2^{k_2} \cdots p_f^{k_f}$ pro $p_f \in$ faktor báze, $k \in \mathbb{N}$. Následně si sestavíme matici z relací. V každém řádku budeme mít vektor, který reprezentuje exponenty prvočísel z faktor báze pro rozklad daného $Q(x)$, tyto exponenty jsou nad $\mathbb{Z}_2$. Například, mějme faktor bázi rovnu množině $\{-1, 2, 5, 11, 19\}$, a rozklad $Q(x) = 2 \cdot 11^2 \cdot 19$. Potom v řádce máme vektor, který vypadá následovně $(0, 1, 0, 0, 1)$. Do faktorové báze přidáváme ještě číslo -1, kvůli možným zaporným hodnotám polynomu $Q(x)$. Ve sloupcích potom této matice máme exponenty prvočísel z faktor báze. Dále nalezneme množinu řešení S homogenní soustavy rovnic z kroku 12. Každé řešení představuje množinu vybraných $Q(x_i)$, jejichž produktem je hledaný čtverec. Nakonec postupně procházíme všechna řešení a utváříme z nich hledané čtverce, z jejichž rozdílů kořenů hledáme největšího společného dělitele.

Takto funguje ve vší obecnosti tento algoritmus. Pojďme se na chvíli zaměřit na řešení rovnice. Může se zde naskýtat otázka, proč zrovna musíme takto

Algorithm 4 Pseudokód kvadratického síta

1: **function** QuadraticSieve(n, B)
2: faktor_báze = $\{p_i | LegendreSymbol(n, p_i) = 1, p_i < B, i \in \mathbb{N}\}$
3: relace = $\emptyset$
4: definuj interval síta = $[-M, M]$ ▷ Zde začíná prosévací část
5: **while** #relace< #faktor_báze **do**
6: Vypočítej $Q(x_i)$ pro $x_i \in [-M, M]$
7: **if** $Q(x_i)$ je hladká nad faktorovou bází **then**
8: ulož $Q(x_i)$ mezi relace
9: **end if**
10: **end while** ▷ Konec prosévací části
11: Sestav matici $\mathbb{A}$, kde každá řádka je sestavená z exponentů faktorů $r \in$ relace
12: Najdi množinu řešení S homogenní soustavy lineárních kongruencí $\vec{z}\mathbb{A} \equiv \vec{0} \pmod 2$
13: **for** $s \in S$ **do**
14: $x^2 = \prod_{Q(x_i) \in s} Q(x_i)$
15: $y^2 = \prod_{x_i \in s} x_i$
16: **if** $1 < \gcd(x \pm y,\, n) < n$ **then return** $\gcd(x \pm y,\, n)$
17: **end if**
18: **end for**
19: **return** Nepovedlo se faktorizovat
20: **end function**

řešit požadovanou homogenní rovnici. Je to z toho důvodu, že chceme nalézt součin podmnožiny nějakých $Q(x_i)$, jejichž souřadnice (exponenty) jsou liché. Tímto v podstatě říkáme, že hledáme takovou podmnožinu vektorů, jejichž součet všech souřadnic je sudý.

Mějme danou množinu všech $Q(x)$. Snažíme se nalézt řešení kongruence

$$Q(x_1)a_1 + Q(x_2)a_2 + \cdots + Q(x_n)a_n \equiv 0 \pmod 2, a_i \in \mathbb{Z}_2$$

Pokud přepíšeme všechna $Q(x_i)$ na dané vektory (nazvěme si je třeba $\vec{z}_i$), jak je výše uvedeno, získáme z původní konruence

$$\vec{z}_1 a_1 + \vec{z}_2 a_2 + \cdots + \vec{z}_n a_n \equiv 0 \pmod 2$$

To odpovídá homogenní soustavě lineárních kongruencí z algoritmu 4, kroku 12

$$\vec{z}\mathbb{A} \equiv \vec{0} \pmod 2$$

Algoritmus existuje i v jiných variantách, kdy se snažíme optimalizovat prosívací část, zejména prosívací interval. Například existuje další varianta zvaná MPQS(*Multiple Polynomial Quadratic Sieve*), která nepracuje pouze s jedním polynomem, ale s více polynomy najednou. Těmito optimalizacemi

se tu však zabývat nebudeme, postačí nám pouze základní varianta tohoto algoritmu, ale je vhodné je alespoň zmínit.

Tyto algoritmy dokážou nalézt netriviální faktor s pravděpodobností 2/3. Vycházejme z posledního bodu algoritmu tedy $x^2 \equiv y^2 \pmod{n}$. Jestliže $n = pq$, potom platí:

$$x^2 \equiv y^2 \pmod{pq} \Leftrightarrow pq|(x^2 - y^2) \Leftrightarrow pq|(x+y)(x-y) \Leftrightarrow$$
$$(p|(x+y) \vee p|(x-y)) \wedge (q|(x+y) \vee q|(x-y))$$

$p\|(x+y)$	$p\|(x-y)$	$q\|(x+y)$	$q\|(x-y)$	gcd $x+y, n$	gcd $x-y, n$	Úspěch
lze	lze	lze	lze	n	n	ne
lze	lze	lze	nelze	n	p	ano
lze	lze	nelze	lze	p	n	ano
lze	nelze	lze	lze	n	q	ano
lze	nelze	lze	nelze	n	1	ne
lze	nelze	nelze	lze	p	q	ano
nelze	lze	lze	lze	q	n	ano
nelze	lze	lze	nelze	q	p	ano
nelze	lze	nelze	lze	1	n	ne

Tabulka 3.4: Možné výstupy pro kongruenci $x^2 \equiv y^2 \pmod{N}$, tabulka převzata z [19].

Z tabulky 3.4 můžeme vidět, že v šesti z devíti případů nalezneme hledaný faktor. Proto tedy je pravděpodobnost nalezení určitého faktoru 2/3.

3.3.3.1 Příklad použití

Mějme číslo $n = 221 = 13 \cdot 17$, které chceme faktorizovat. Nejdříve si určíme náš polynom, kterým bude $Q(x) = (x + \lfloor\sqrt{n}\rfloor)^2 - n$, tedy pro $n = 221$ bude nasledující $Q(x) = (x + 14)^2 - 221$. Zvolme si naší faktorovu bázi do čísla 37 $F = \{-1, 5, 7, 11, 31, 37\}$. Dále vypočítáme některé hodnoty:

i	x_i	$Q(x_i)$
1	-8	-185
2	-7	-172
5	-4	-121
11	2	35

Tabulka 3.5: Zvolené hodnoty z prosívacího intervalu

První hodnota v tabulce je hladká na faktorové bázi. Ověříme si to rozepsáním na prvočísla

$$-185 = -1 \cdot 5 \cdot 37$$

Další hodnota již ale hladká na faktor bázi není. Pokud si jej opět rozepíšeme, získáme

$$-172 = -1 \cdot 2^2 \cdot 43$$

Zbývající hodnoty hladké jsou. Jejich roznásobením získáme kongruenci

$$Q(x_1)Q(x_5)Q(x_{11}) \equiv (-8 \cdot (-4) \cdot 2) \pmod{n}$$

což jsou naše $x^2 \equiv y^2 \pmod{n}$. Kořen hodnoty $y^2 = 30$ není na první pohled zřejmý. Musíme nalézt kvadratické reziduum q, které splňuje $y^2 \equiv q \pmod{n}$. Jeho nalezení můžeme učinit například postupným zkoušením hodnot z $\mathbb{Z}_{221}$. Nalezené $q = 76$, kořen čtverce x^2 je na první pohled zřejmý, neboť $x^2 = 64 = 2^6 \Rightarrow x = 2^3$. Získáním největšího společného dělitele $\gcd(76 - 8, 221)$ získáme faktor 17.

Základní časová složitost (viz [20]) kvadratického síta bez různých optimalizací se odhaduje jako $O(e^{\sqrt{1.125 \ln(n) \ln \ln(n)}})$. Složitost je skoro stejná jako tomu bylo u Lenstrova algoritmu, avšak jak již bylo řečeno, kvadratické síto používá méně náročné operace a proto je považováno za rychlejší.

3.3.4 Obecné číselné síto

Posledním vybraným algoritmem a taky doposud nejrychlejším známým, je číselné síto nad obecným tělesem[6]. V mnoha krocích je podobný kvadratickému sítu, avšak s tím rozdílem, že pracuje nad obecným tělesem, tedy s tělesem polynomů. Dokonce má i stejnou pravděpodobnost pro nalezení hledaného faktoru.

Uveďme si na začátek důležité tvrzení, které algoritmus využívá.

Tvrzení 3.3.1. Nechť je dán polynom $f(x) \in \mathbb{Z}[x]$ a jeho komplexní kořen $\alpha \in \mathbb{C}$ a $m \in \mathbb{Z}/n\mathbb{Z}$ takové, že $f(m) \equiv 0 \pmod{n}$, pro $n \in \mathbb{N}, n \geq 2$, potom existuje jednoznačné zobrazení $\phi : \mathbb{Z}[\alpha] \to \mathbb{Z}n\mathbb{Z}$, pro které platí:

1. $\phi(ab) = \phi(a)\phi(b) \forall a, b \in \mathbb{Z}[\alpha]$
2. $\phi(a + b) = \phi(a) + \phi(b) \forall a, b \in \mathbb{Z}[\alpha]$
3. $\phi(1) \equiv 1 \pmod{n}$
4. $\phi(\alpha) \equiv m \pmod{n}$

Začněme opět pseudokódem, kterým si popíšeme, jak funguje tento algoritmus.

Algoritmus 5 si nyní detailněji rozebereme. V prvním kroce vybíráme polynom $f(x)$ takový, jehož kořenem je $m \in \mathbb{Z}$. Vybrat takový polynom není nijak obtížné. Postup funguje podobně jako zápis čísla v jiné reprezentaci

[6] zkráceně jej budeme nazývat GNFS z anglického General Number Field Sieve

Algorithm 5 Pseudokód GNFS

1: **function** GNFS(n)
2: Zvol ireducibilní polynom $f(x)$, jehož kořenem je $m \in \mathbb{Z}$ takový, že $f(m) \equiv 0 \pmod{n}, f(x) \in \mathbb{Z}[x]$
3: Zvol velikost faktorové báze a sestav racionální a algebraickou faktorovou bázi a kvadratickou bázi
4: Nalezni relace (dvojice $(a, b)a, b \in \mathbb{Z}$), pro které platí: $\gcd(a, b) = 1$, $a + bm$ je hladké na racionální faktorové bázi a $b^{st(f)}f(a/b)$ je hladké na algebraické bázi.
5: Z relací utvoř matici $\mathbb{A}$
6: Nalezni množinu řešení S homogenní soustavy rovnice $\vec{x}\mathbb{A} \equiv 0 \pmod{2}$
7: Nalezni racionální kořen hodnoty y, kde $y^2 = \prod_{(a,b)\in S}(a + bm)$
8: Nalezni algebraický kořen hodnoty x, kde $x^2 = \prod_{(a,b)\in S}(a + b\alpha)$, kde α je komplexním kořenem polynomu $f(x)$.
9: **return** $\gcd(x - y, n)$, $\gcd(x + y, n)$
10: **end function**

(například binární). Vybereme si nejdříve nějaké m a následně zvolíme polynom tak, že $n = \sum_{i=0}^{d} a_i m^i$, kde $0 < a_i < m$. Tímto získáme polynom $f(x) = a_d x^d + a_{d-1}x^{d-1} + \cdots + a_0$ a zároveň nám bude platit $f(m) \equiv 0 \pmod{n}$. Koeficienty polynomu můžeme navíc upravit takto:

$$a_i = a_i - m$$

$$a_{i+1} = a_{i+1} + 1$$

Jak vybírat takový polynom se zabývají například v práci [21].

Dalším krokem je zvolit vhodnou hranici pro faktorové báze (označíme si je B_r pro racionální faktorovu bázi a B_a pro algebraickou). Velikosti těchto bází jsou dány empiricky a jsou také mnohdy závislé na prosévací části algoritmu. Racionální báze obsahuje všechny prvočísla p_i do námi zvolené hranice. K tomuto prvku ukládáme také $p_i \pmod{m}$, tedy racionální báze je množina:

$$RFB = \{(p_i, p_i \pmod{m}) | p_i < B_r\}$$

Další bázi, kterou vybíráme je algebraická faktorová báze. To je pro nás množina:

$$AFB = \{(p_i, r) | p_i < B_a, f(r) \equiv 0 \pmod{p}\}$$

Nakonec vybíráme kvadratickou bázi. Ta splňuje stejné požadavky, jako algebraická faktorová báze, avšak prvočísla zde jsou větší, než největší prvočíslo v algebraické faktorové bázi.

Navazuje prosévací část. Nejdříve si zvolíme inteval $I = [-C; C]$. Následně si zvolíme fixní hodnotu b, většinou $b = 0$ a v každé iteraci následně tuto

Algorithm 6 Prosévací část GNFS

function síto
  relace = ∅
  $b = 0$
  while #relace< #RFB+#AFB+#KB+1 **do**
    $a_i = i + bm, e_i = b^{st(f)} f(i/b) \forall i \in [-C; C]$
    for all $(p, r) \in$ RFB **do**
      Odstraň největší mocninu prvočísla p z $a_i, \forall i \in [-C; C]$.
    end for
    for all $(p, r) \in$ AFB **do**
      Odstraň největší mocninu prvočísla p z $e_i, \forall i \in [-C; C]$.
    end for
    Přidej do množiny relací množinu $M = \{(i, b) | a_i = e_i = 1, \gcd(i, b) = 1, i \in [-C; C]\}$
    Zvedni číslo b o 1.
  end while
end function

hodnotu zvýšíme o 1. Hodnotu a vypočítáme podle uvedeného postupu v algoritmu 6. Tuto část si popíšeme pseudokódem, aby bylo zřetelnější, jak tento postup funguje.

Z algoritmu 6 můžeme tedy vidět, jak prosévací část funguje. Je velmi podobná té části, která již byla v kvadratickém sítu, avšak zde neoperujeme pouze nad jednou bází, ale nad dvěma bázemi. Tato část v algoritmu je právě ta, která je nejvíce časově náročná, jelikož se prochází celkem široký interval a to vždy do té doby, než se nám naplní množina relací.

Dalším krokem v algoritmu 5 je vytvoření matice $\mathbb{A}$ a následné získání řešení homogenní soustavy rovnic $\vec{x}\mathbb{A} = 0 \pmod 2$. Tuto rovnici jsme si již popsali v kapitole 3.5. Její princip spočívá v tom samém, jako tomu bylo u kvadratického síta. Ukážeme si ale, jak se taková matice z těchto relací skládá opět pseudokódem.

Sestavení matice můžeme vidět v algoritmu 7. Opět vkládáme do řádku exponenty tak, abychom vždy získali požadovaný perfektní čtverec. Tato část není sice až tak náročná časově, ale naopak je velmi náročná paměťově, neboť tato matice může mít příliš velké rozměry. Zde by mohlo být spíše paměťově rozumnější využít řídkých matic.

Posledním krokem je určení čtverců hodnot x a y. Množina řešení nám opět říká, které a_i a b_i z relací máme použít. Tím získáme jejich produkt a následně otestujeme, zda rozdíl kořenů těchto čtverců je naším hledaným faktorem.

Tímto máme určený obecný postup, jak tento algoritmus funguje. Nyní následuje příklad, na kterém si objasníme, jak to vlastně celé funguje a jak to použít.

Algorithm 7 Sestavení matice GNFS

function SÍTO(Polynom $f(x)$, kořen m polynomu $f(x)$, relace)
  Připravme si nulovou matici $\mathbb{A} \in \mathbb{Z}_2^{\#\text{relace},\#\text{RFB}+\#\text{AFB}+\#\text{KB}+1}$
  for all $(a_i, b_i) \in$ relace **do**
    if $a_i + b_i m < 0$ **then** $\mathbb{A}_{i,0} = 1$
    end if
    for all $(p_j, r_j) \in$ RFB **do**
      exponent = nejvyšší mocnina p_j, která dělí $a_i + b_i m$
      if exponent je lichý **then** $\mathbb{A}_{i,1+j} = 1$
      end if
    end for
    for all $(p_j, r_j) \in$ AFB **do**
      exponent = nejvyšší mocnina p_j, která dělí $(-b_i)^{st(f)} f(-a_i/b_i)$
      if exponent je lichý **then** $\mathbb{A}_{i,1+\#\text{RFB}+j} = 1$
      end if
    end for
    for all $(p_j, r_j) \in$ KB **do**
      if Legendreův symbol dvojice $(a_i + b_i p_j, r_j) \neq 1$ **then** $\mathbb{A}_{i,1+\#\text{RFB}+\#\text{AFB}+j} = 1$
      end if
    end for
  end for
end function

3.3.4.1 Příklad použití

Nyní použijeme příklad z [22]. Budeme se snažit faktorizovat číslo $n = 45113 = 197 \cdot 229$. První bod algoritmu nám říká, abychom vybrali nějaký vhodný polynom $f(x)$. Zvolme si nejdříve náš kořen polynomu $m = 31$ a rozepišme číslo n v reprezentaci o základu 31.

$$45113 = 31^3 + 1531^2 + 29 \cdot 31 + 8$$

náš polynom bude mít následující tvar:

$$f(x) = x^3 + 15x^2 + 29x + 8$$

Nyní máme náš polynom vybraný a k němu i přiřazen kořen. Dalším krokem je určení našich faktorových bází [7]. Pro RFB si zvolíme hranici prvočísel $B_r = 30$ a pro AFB si zvolíme $B_a = 90$. Báze budou mít následující podobu:

$$RFB = \{(2,2), (3,3), (5,5), (7,7), (11,11), (13,13), (17,17), (19,19),$$

$$(23,23), (29,29)\}$$

[7] Budeme zde používat zkratky jejich názvů, jak jsou uvedeny v algoritmech

$$AFB = \{(2,0),(7,6),(17,13),(23,11),(29,26),(31,18),(41,19),(43,13),$$
$$(53,1),(61,46),(67,2),(67,6),(67,44),(73,50),(79,23),(79,47),(79,73),$$
$$(89,28),(89,62),(89,73)\}$$
$$KB = \{(97,28),(101,87),(103,47),(107,4),(107,8),(107,80)\}$$

Všechny báze máme nastavené, nyní tedy přistoupíme k prosévací části. Nastavíme si interval $I = [-400; 400]$ a budeme hledat takové dvojice (a_i, b_i), které jsou hladké na RFB a AFB. Naše množina relací (dvojice (a, b)) je následující:

$$\{(-73,1),(-13,1),(-6,1),(-2,1),(-1,1),(1,1),(2,1),(3,1),(13,1),(15,1),$$
$$(23,1),(61,1),(1,2),(3,2),(33,2),(2,3),(5,3),(19,4),(14,5),(37,5),(313,5),$$
$$(11,7),(15,7),(-7,9),(119,11),(-247,12),(175,13),(5,17),(-1,19),(35,19),$$
$$(17,25),(49,26),(375,29),(9,32),(1,33),(78,37),(5,41),(9,41)\}$$

Nyní musíme sestavit matici $\mathbb{A}$. Uvedeme si příklad zde jednoho řádku, jak v této matici vypadá. Řádka matice je opět vektor. Rozepišme si, co se v tomto vektoru nachází například pro relaci $(119, 11)$:

$$(\overbrace{0}^{|\operatorname{sgn} a+bm|}, \underbrace{0,0,1,0,0,0,0,0,1,0}_{\text{exponenty RFB}}, \overbrace{0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0}^{\text{exponenty AFB}},$$
$$\underbrace{1,1,0,0,0,0}_{\text{KB Legendre}})$$

Po sestavení matice $\mathbb{A}$ a nalezení řešení homogenní soustavy rovnic uvedené v algoritmu 5 kroku 6, získáme následující řešení:

$$S_0 = \{(-2,1),(1,1),(13,1),(15,1),(23,1),(3,2),(33,2),(5,3),(19,4),(14,5),$$
$$(15,7),(119,11),(175,13),(-1,19),(49,26)\}$$

Nyní vypočítáme požadované x^2 a y^2:

$$y^2 = \prod_{(a,b)\in S_0} (a+bm) = 459997127517951955826063769600000$$

$$x^2 = (\prod_{(a,b)\in S_0} (a+b\alpha)) \mod f(x) =$$

$$58251363820606365\alpha^2 + 149816899035790332\alpha + 75158930297695972$$

Tedy našimi hledanými kořeny jsou:

$$x^2 = 2553045317222400^2$$

$$y^2 = (108141021\alpha^2 + 235698019\alpha + 62585630)^2$$

Pro hodnotu y použijeme větu 3.3.1, z čehož získáme

$$y^2 = (\phi(108141021\alpha^2 + 235698019\alpha + 62585630)^2)^2 = 111292745400^2$$

Faktory potom získáme pomocí

$$\gcd(45113, 111292745400 + 2553045317222400) = 197$$

a

$$\gcd(45113, 111292745400 - 2553045317222400) = 229$$

Časová složitost (viz [23]) vychází z optimálního výběru stupně generovaného polynomu a v závisloti na stupni polynomu určení velikosti bází. Složitost definujeme jako

$$O(\exp(\sqrt[3]{\frac{64}{9}}(\ln n)^{1/3}(\ln \ln n)^{2/3}))$$

Kapitola 4

Návrh implementace a paralelizace výpočtů

Nyní si rozebereme implementaci a paralelizaci faktorizace pomocí eliptických křivek. První začneme s výběrem použité technologie a následně se sekvenčním výpočtem a dále si rozebereme paralelizaci výpočtů.

4.1 Výběr programovacího jazyka

Nejdůležitější technologií použitou pro implementaci je jistě vybraný programovací jazyk. Nejdříve jsem se rozhodoval mezi jazyky C++ a Python. Pro oba tyto jazyky jsou implementovány knihovny, respektive moduly, které obsahují efektivní implementace různých matematických operací. Pro Python navíc existuje kompilátor jménem Cython, který je i sám o sobě programovacím jazykem. Dokáže převést Pythonní kód do jazyka C, stejně tak i kód napsaný v jazyce Cython a tím urychlit celý průběh, neboť bude kód zkompilovaný. Ovšem stále zde používá struktury z jazyka Python, které průběh zpomalují. Pro jazyky C++ a Python existuje implementace stejné technologie pro paralelizaci výpočtů MPI, jež byl vybrán a je popsán níže.

Nakonec jsem se rozhodl pro jazyk C++, respektive standard C++17 z dvou důvodů. První důvodem je fakt, že výpočet probíhal na školním klastru STAR (viz [24]), na kterém je připravené prostředí pro použití C++, MPI a OpenMP. Druhým důvodem je ten, že s použitím OpenMP, MPI a C++ mám větší zkušenosti.

V implementaci algoritmu je nutné pracovat s velkými čísly. Toto nám zajišťuje knihovna NTL, jejíž popis je uveden níže. Další knihovna, která pro implementaci algoritmů byla využita, je OpenMP společně s OpenMPI, respektive její nadstavbou *Boost.MPI*. Tyto knihovny napomáhají paralelizaci výpočtů.

4.2 NTL

Number Theory Library[25] neboli zkráceně NTL, je knihovna, která nám umožňuje reprezentovat celá čísla s neomezenou přesností a také s nimi pracovat. Využijeme zde hlavně funkce pro modulární aritmetiku a také pro určení *nejmenšího společného dělitele*, abychom mohli odhalit hledaného dělitele.

Další výhodou, proč tuto knihovnu použít, je fakt, že se jeví jako efektivnější než některé konkurenční (jako je například knihovna FLINT [26]), které se také zabývají implementací funkcí pro obor teorie čísel. Tato měření můžeme taktéž nalézt v dokumentaci (viz [25]) knihovny NTL.

4.3 OpenMP

Náš program budeme spouštět na školním klastru STAR, kde budeme chtít využít všechna dostupná vlákna, protože výše zmíněná knihovna by je všechna nevyužila kvůli konfiguraci. Zde se nám nabízejí dvě možnosti. Jednou takovou možností je využít klasická POSIXová vlákna, respektive POSIX API, na úkor složitější implementace a manipulace s vlákny. Druhou možností je, a kterou i volíme, využít knihovnu OpenMP [27]. Tato knihovna nám zjednodušuje práci s vlákny, protože se vývojáři této technologie snaží standardizovat vysokoúrovňovou paralelizaci, která je i přenosná, což by v případě POSIX API nebylo vždy možné, protože například operační systém Windows jich nevyužívá a využívají vlastní implementace vláken, kterou můžeme nalézt v [28].

Narozdíl od knihovny OpenMPI, tato knihovna pracuje nad sdílenou pamětí, je tedy na programátorovi, aby si hlídal různé datové závislosti. Abychom mohli určit v kódu, kde požadujeme paralelizaci, tak k tomu slouží direktivy pro překladač.

Uvedeme si nyní dvě základní direktivy, které jsou pro práci velmi důležité:

parallel direktiva definuje blok kódu, tzv. **paralelní region**, který bude spuštěn každým vláknem zvlášť. Na konci tohoto bloku je vytvořena bariéra, která čeká na všechna vlákna, než dokončí výpočet.

critical direktiva určuje kritickou sekci kódu. To nám zajistí, abychom byli schopni předcházet datovým závislostem a nepřepisovali si tak hodnoty sdílených proměnných.

V této knihovně je také možné určit v paralelním bloku, která vlákna jsou pracovní a které vlákno je hlavní.

4.4 MPI

Nejdříve než začneme s popisem knihovny jako takové, je vhodné na začátek uvést fakt, že MPI je pouze standard, který implementujou některé knihovny,

jako například Boost.MPI knihovna, kterou si popíšeme níže. Dokumentaci k tomuto standardu můžeme nalézt v [29]. Tento standard nám definuje rozhraní, podle kterého se implementace drží. Implementace se snaží zachovávat stejné názvosloví, jako má právě daný standard.

Další důležitou použitou technologií je knihovna projektu Boost MPI [30] (*Message Passing Interface*). Je důležitým faktorem pro komunikaci mezi procesy v reálném čase. Pokud chceme tuto technologii využít, je nutné využít i jejich aplikaci **mpiexec**, případně **mpirun**, které spustí naši aplikaci na všech procesorech ve stejnou dobu. Využívají se veškerá jádra procesorů, tedy pokud bychom tuto aplikaci spouštěli na systému se dvěma procesory, kde každý procesor má 4 jádra, tak chování programu bude takové, že celá komunikace bude probíhat mezi všemi 8 jádry. Zde také samozřejmě záleží i na tom, s jakou konfigurací aplikace **mpirun** nebo **mpiexec** program spouštíme.

Každý proces operuje se svojí přidělenou pamětí a komunikace mezi procesy může probíhat pouze přes funkce, které nám rozhraní Boost MPI nabízí. Toto rozhraní je v souladu se standardem MPI.

Abychom v naší aplikaci mohli používat komunikaci mezi těmito procesy, je nutné si nejdříve zařadit daný proces do komunikačního prostředí. Po této inicializaci jsou všechny procesy v komunikačním prostředí zvaném **MPI_COMM_WORLD**. Každý proces má svůj takzvaný *rank* neboli číslo procesu. Číslování je vždy od 0 do $p - 1$, kde p je počet spuštěných procesů.

Komunikace mezi procesy může probíhat ve dvou variantách, které jsou blokující a neblokující. Blokující komunikace znamená, že proces, který odesílá nějakou zprávu jinému procesu čeká, dokud ji nepřijme. Přijetí můžeme zde chápat jako splnění určité podmínky. Neblokující naopak pošle zprávu a může pracovat dál a nečeká na potvrzení, že si ji proces vyzvedl.

Pomocí funkcí pro zasílání zpráv může být uskutečněna komunikace. Tyto funkce také umožňují zasílat zprávy se specifickou značkou (**TAG**), která nám může například napomoci tomu, abychom věděli o co se ve zprávě jedná. Značky jsou určeny programátorem, jež za ně odpovídá. Použité značky si popíšeme níže s implementací.

Nyní máme popsané veškeré technologie 3. stran, které jsme pro implementaci využili. Dále je již příliš rozebírat nebudeme a případně se na ně pouze odkážeme.

4.5 Vstupní argumenty programu

Naše implementace přijímá některé argumenty z příkazové řádky. Díky nim budeme určovat například maximální počet iterací nebo jaké číslo se má faktorizovat. Mezi těmito argumenty jsou:

1. Přepínač pro určení, který model eliptické křivky použít (Weierstrassův nebo Edwardsův model),

2. Indikátor startu paralelního výpočtu,

3. Maximální velikost násobku bodu (výchozí hodnota je nastavena na $\sqrt{n}$),

4. Faktorizované číslo n.

Program přijímá i další argumenty, které jsou podpůrné pro nápovědu nebo měření v poslední kapitole a není tedy potřeba je zde uvádět.

Dalším parametrem naší implementace je hodnota, po které má algoritmus otestovat, zda jsme již nenarazili na dělitele složeného čísla. Tuto hodnotu máme pevně nastavenou jako:

$$test\_after = max\left(100, \left\lfloor \frac{\sqrt{n}}{1000000} \right\rfloor\right).$$

4.6 Sekvenční výpočet Lenstrova algoritmu

Kroky Lenstrova algoritmu jsme si již popsali v předešlé sekci 3.2.2. Přepišme si tedy do pseudokódu, jak budou jednotlivé kroky ještě jednou vypadat. Nyní si však kód přepíšeme obecněji, stejně jako to máme následně i v naší implementaci, a pak jednotlivé kroky popíšeme.

Algoritmus 8 nám popisuje obecně jak půjdou naše kroky tohoto algoritmu. Vstupem funkce je složené číslo n, které chceme faktorizovat a model, na kterém budeme počítat Na druhé řádce si vygenerujeme nejdříve eliptickou křivku a bod na ní. Následně si nastavíme hodnotu nějakého pomyslného počitadla na 0. Toto počitadlo nám bude signalizovat, kdy máme zkusit provést výpočet inverze a tím tedy i získat dělitele složeného čísla n. Následně začíná *forcyklus*, kde začínáme od 2 a iterujeme do nějaké námi stanovené hranice K. V každé iteraci počítáme k–tý násobek bodu *point*. Zde však postupně získáváme faktoriál hodnoty K, neboť v každé iteraci si ukládáme nový bod do proměnné *point*. Získáváme tedy postupně $2 \cdot 3 \cdot 4 \cdots K \cdot point = K! \cdot point$. Dále vyzkoušíme, pokud jsme urazili určitý počet cyklů, zda jsme nalezli hledaného dělitele. Poslední podmínka v cyklu určuje, zda jsme dosáhli neutrálního prvku. Pokud tomu tak je, tak cyklus ukončíme a vrátíme se k prvnímu kroku funkce.

V původním návrhu jsme si ukázali, že budeme počítat $\mathrm{lcm}(1, 2, \ldots, K)$, tak to samozřejmě ničemu nevadí, tento způsob výpočtu můžeme nalézt v [2]. Tímto se nám výpočet optimalizuje, neboť při postupném napočítávání můžeme dříve dosáhnout neutrálního prvku a tím cyklus ukončit dříve.

V definicích uvedených v sekcích 1.4.1 a 1.4.3 jsme si uvedli pouze vzorce pro sčítání dvou bodů eliptické křivky. Pokud bychom chtěli násobit tento bod, tak nejprimitvnější způsob je udělat tolik součtu, jako je násobek bodu. To však je zřejmě velmi neefektivní. Větší efektivity dosáhneme pomocí algoritmu *Double-and-Add*, který si popíšeme níže.

Algorithm 8 Lenstrův algoritmus obecně

```
1: function LENSTRA_FACTORIZE(CompositeNumber n, AbstractModel model)
2:     point, curve = model.generate_elliptic_curve_with_point
3:     test_counter = 0
4:     for k ∈ {2, 3, ..., K} do
5:         point = k · point    ▷ Násobí bod tak, jak je definováno pro model eliptické křivky v sekcích 1.4.1 a 1.4.3
6:         if test_counter = test_bound then
7:             factor = model.try_get_factor(point)
8:             if factor > 1 ∧ factor < n then
9:                 return factor
10:            end if
11:            test_counter = 0
12:        end if
13:        if model.is_infinity_point(point) then
14:            go to step 2.
15:        end if
16:        test_counter = test_counter + 1
17:    end for
18:    if factor not found then
19:        go to step 2.
20:    end if
21: end function
```

Algorithm 9 Double–and–Add

```
1: function DOUBLE_AND_ADD(násobek k, bod P)
2:     N = P                                  ▷ Uložíme do proměnné N bod P
3:     Q = O                                  ▷ Zde O značí neutrální prvek
4:     for i ∈ {0, ..., m} do
5:         if k_i = 1 then                    ▷ Zde testujeme i–tý bit hodnoty k
6:             Q = Q + N    ▷ Zde využijeme vzorce pro sečtení dvou různých bodů
7:         end if
8:         N = N + N                          ▷ Zde využijeme vzorce pro zdvojení bodu
9:     end for
10:    return Q
11: end function
```

Pomocí algoritmu 9 můžeme efektivněji násobit body eliptické křivky. Důležité pro tento algoritmus je, že si můžeme násobek k přepsat v binárním rozvoji jako $\sum_{i=0}^{m} a_i 2^i$ pro $a_i \in \{0, 1\}$. Popíšeme si jednotlivé kroky algoritmu pro násobení. Nejdříve si uložíme do dočasné proměnné N, resp. Q hodnotu

P, resp. O (značící neutrální prvek). Pak v cyklu testujeme jednotlivé bity násobku k. Pokud je bit nastaven na hodnotu 1, tak proběhne sečtení dvou bodů. Na konci iterace vždy zdvojíme bod N.

Tímto máme popsané obecné algoritmy, jak bude probíhat výpočet. Tyto výpočty v naší implementaci použijeme. Existují však i další optimalizace, které urychlují celkový výpočet.

4.7 Implementace modelů eliptických křivek

V implementaci se nacházejí 2 modely eliptických křivek, kterými jsou Weierstrassův a Edwardsův. Tyto modely se starají o to, aby udržovaly vygenerovanou eliptickou křivku, vygenerovaly nám novou eliptickou křivku společně s bodem a prováděly výpočty s body. Oba tyto modely operují pouze s projektivními souřadnicemi.

Tyto modely obsahují dvě podpůrné funkce pro sečtení dvou bodů a zdvojení bodu, jimiž jsou **add_points** a **double_point**. V těchto funkcích se nachází výpočty tak, jak jsme si je definovali v sekcích 1.4.1 a 1.4.3.

Oba modely udržují strukturu svých eliptických křivek, tedy Weierstrassův model udržuje strukturu s hodnotami a, b z definice 11 a Edwardsův model udržuje pouze hodnotu d uvedenou v definici 14. Dále tyto struktury udržují složené číslo N, aby bylo zřejmé nad jakým okruhem z definice 6 operujeme.

Modely si navíc také drží množinu již vygenerovaných křivek, aby nedocházelo ke generování duplikátů.

4.8 Optimalizace sekvenčních výpočtů

Nyní si popíšeme některé optimalizace, které například využívá dostupná implementace GMP-ECM [31].

4.8.1 Rozdělení výpočtu na fáze

Jednou z optimalizací je rozdělení algoritmu na dvě fáze. Nyní budeme vycházet z popisu uvedeném v [32]. První fáze je stejná, jako jsme si již popsali výše, tedy napočítáme si násobek bodu P, $Q = k!P$. Ten je také určen hranicí, kterou si označíme B_1. Pokud se nám při počítání bodu Q nepodařilo faktorizovat číslo, přecházíme na druhou fázi. Jedna z variant druhé fáze je založena na narozeninovém paradoxu. Tato varianta využije předpočítaného nenulového bodu Q z první fáze a předpokládá tři kladná celá čísla $e, r, \overline{r}$, kde $r < \overline{r}, r\overline{r} \approx B_2$. Hodnotu e volíme záměrně *malou*. Dále si zvolíme čtyři *malá* náhodná celá čísla $u, v, \overline{u}, \overline{v}$. Výpočet druhé fáze je pak:

1. Vypočítáme $(x_i, y_i) = (ui + v)^e Q$ pro $i \in \{1, 2, \ldots, r\}$

2. Najdeme polynom $f = \sum_{j=0}^{r} f_j X^j = \prod_{i=1}^{j}(X - x_i) \mod n$, kde n je faktorizované číslo

3. Vypočítáme $(\overline{x}_j, \overline{y}_j) = (\overline{u}i + \overline{v})^e Q$ pro $i \in \{1, 2, \ldots, \overline{r}\}$

4. Následně vypočteme $d = \prod_{j=1}^{\overline{r}} f(\overline{x}_j) \mod N$

5. A nakonec vyzkoušíme nalézt společného dělitele $\gcd(n, d)$

Více o této variantě jak volit dané hodnoty můžeme nalézt v [32].

Další variantou je zvolit si druhou hranici, označme si ji B_2. Druhá fáze opět začne, pokud při první fázi nedošlo k nalezení dělitele nebo jsme dosáhli hranice B_1. Zde opět využijeme předpočítaného nenulového bodu Q, jako tomu bylo u předešlé varianty. Tato varianta spočívá následně v tom, že nebudeme počítat násobky k, jako tomu bylo u první fáze, ale začneme počítat nový bod $Q = \prod_{B_1 \leq p \leq B_2}(pQ)$, kde p je prvočíslo, tedy používáme pouze prvočísla v daném intervalu mezi $B1$ a B_2. Tuto variantu používá například implementace GMP-ECM (viz [33]).

4.8.2 Optimalizace algoritmu Double-and-Add

V [32] je uvedena optimalizace algoritmu *Double–and–Add*, který jsme si popsali výše (viz algoritmus 9). Optimalizace spočívá v redukci sčítání různých bodů, protože jak si můžeme všimnout v tabulce 1.1 s počtama operací uvedenou v první kapitole, zdvojení bodu je méně náročná operace. Můžeme například využít zápisu ve *kvartérní* soustavě. Násobek k přepíšeme jako $k = \sum_{i=0}^{m} a_i 4^i$, pro $a_i \in \{0, 1, 2, 3\}$. Úpravu algoritmu můžeme vidět v pseudokódu 10.

Nyní jsme zredukovali počet sčítání a zvýšili tedy počet zdvojení. V [32] uvádí zrychlení tímto algoritmem a touto soustavou až o 18%. Dalším faktorem optimalizace je používání násobků k takových, aby obsahovaly co nejméně hodnot $a_i = 1$.

4.9 Návrh paralelizace

Naše implementace bude využívat hybridního modelu paralelizace pomocí BoostMPI a OpenMP. To znamená, že bude probíhat komunikace mezi procesy (práce nad distribuovanou pamětí) a zároveň každý proces bude využívat veškerá dostupná vlákna (práce nad sdílenou pamětí), více viz [34].

4.9.1 Paralelizace mezi procesy

Pro paralelizaci jednotlivých výpočtů je zvolen algoritmus *Master–slave* (převzat s menšími úpravami z [34]) pro práci nad distribuovanou pamětí. Tento

Algorithm 10 Double–and–Add_optimized

```
1: function DOUBLE_AND_ADD(násobek k, bod P)
2:     N = P // Uložíme do proměnné N bod P
3:     Q = O // Zde O značí neutrální prvek
4:     for i ∈ {0, ..., m} do
5:         if k_i = 1 then
6:             Q = Q + N
7:         end if
8:         if k_i = 2 then
9:             Q = Q + 2N
10:        end if
11:        if k_i = 3 then
12:            Q = Q + 3N
13:        end if
14:        N = 2 · 2N
15:    end for
16:    return Q
17: end function
```

Algorithm 11 Pseudokód Master–slave algoritmu

```
1: function FACTORIZE_PARALLEL(composite_number)
2:     MPI_Init()                                   ▷ Inicializace procesu
3:     MPI_Comm_rank(MPI_COMM_WORLD, rank)          ▷ Získej číslo
   procesu
4:     if rank = 0 then // Část pro hlavní proces
5:         master_part                              ▷ Rozděl práci všem procesům
6:         start_parallel_factorization             ▷ Začni s vlastním výpočtem
7:         for i ∈ {1, ..., processNumber − 1} do
8:             send_stop_message ▷ Zašli všem ostantím procesům zprávu o
   ukončení výpočtu
9:         end for
10:    else
11:        slave_part                               ▷ Získej úlohu od hlavního procesu
12:        start_parallel_factorization             ▷ Začni výpočet
13:    end if
14:    MPI_Finalize()
15: end function
```

algoritmus spočívá v tom, že máme jeden hlavní proces (**master**), který rozděluje úkoly a přijímá výsledky, které mu zašlou pracovní procesy (**slaves**). Hlavní proces většinou určujeme tak, že jeho hodnota *rank* je 0. Algoritmus 11 popisuje pseudokódem jeho podobu.

Aby nedocházelo ke zbytečnému čekání na odpovědi od ostatních pracují-

cích procesů, hlavní proces také počítá a v průběhu svého výpočtu vždy zkouší, jestli mu nepřišla nějaká řešení od pracujících procesů.

Kroky hlavního procesu (funkce master_part) v pseudokódu 11 jsou následující:

1. Vygeneruj eliptickou křivku a bod na ní.

2. Ověř, že tato eliptická křivka není duplikátní v množině již vygenerovaných křivek. Tuto křivku následně ulož do množiny již vygenerovaných křivek.

3. Eliptickou křivku a bod serializuj a pošli příslušnému procesu.

4. Pokud existuje proces, který čeká na křivku, vrať se ke kroku 1.

Abychom předcházeli duplikátním výpočtům, ukládáme si veškeré vygenerované eliptické křivky do nějaké množiny vygenerovaných křivek.

Jakmile jsou pro ostatní procesy vygenerovány křivky, hlavní proces si pro své výpočty vygeneruje unikátní křivku společně s bodem a začne se svým výpočtem definovanou ve funkci parallel_factorization.

Kroky ostatních procesů jsou poté takovéto:

1. Získej vygenerovanou křivku a bod od hlavního procesu.

2. Přesuň se do funkce s výpočtem nad získanou eliptickou křivkou.

Tím, že nám nyní bude běžet několik procesů nad více eliptickými křivkami najednou, tak tím také zvýšíme naší šanci, že vygenerujeme eliptickou křivku s vhodným řádem a tím celkově náš výpočet opět urychlíme.

Nyní si ještě rozebereme funkci s výpočtem, jak se změní oproti sekvenčnímu výpočtu z pohledu paralelizace nad distribuovanou pamětí. Změna zde nebude nijak výrazná z tohoto pohledu, neboť proces uvnitř cyklu uvedeném v algoritmu 8 vždy při každé iteraci zkontroluje příchozí zprávu o ukončení výpočtu. V cyklu budeme mít tedy navíc jednu podmínku, která ukončí předčasně cyklus, pokud jiný proces našel hledaného dělitele.

Pro zasílání zpráv se používají takzvané *TAGY*, které jsme již zmínili v popisu knihovny MPI. V naší implementaci používáme pouze 2 tyto *TAGY*:

1. **NEW_ECC** – značka označující žádost o novou eliptickou křivku, jedná-li se o pracovní proces. V případě hlavního procesu se jedná o značku, se kterou zasílá novou vygenerovanou křivku.

2. **STOP** – značka signalizující ukončení výpočtu.

Algorithm 12 Paralelní Lenstrův algoritmus

```
1: function LENSTRA_FACTORIZE_PAR(CompositeNumber N, AbstractModel model)
2:     test_counter = 0
3:     for k ∈ {2, 3, ..., K} do                    ▷ Spusť paralelní forcyklus
4:         point = k · point      ▷ Každé vlákno si udržuje vlastní proměnnou point
5:         if test_counter = test_bound then      ▷ Vlákna mají společnou proměnnou test_counter
6:             factor = model.try_get_factor(point)
7:             if factor > 1 ∧ factor < N then
8:                 stop_all_threads
9:                 return factor
10:            end if
11:            test_counter = 0
12:        end if
13:        if model.is_infinity_point(point) then
14:            break
15:        end if
16:        test_counter = test_counter + 1
17:        end_indicator = single_thread_check_message()   ▷ Některé dostupné vlákno zkontroluje zprávu o ukončení, ostatní vlákna toto přeskočí a pokračují v další iteraci.
18:        if end_indicator then stop_all_threads
19:        end if
20:    end for
21:    wait_on_barrier                ▷ Synchronizační bod pro všechna vlákna
22:    point = single_thread_sum_all_points() ▷ Sečti všechny body point od všech vláken. Nech ostatní vlákna pozastavená.
23:    factor = model.try_get_factor(point)
24:    if factor > 1 ∧ factor < N then
25:        stop_all_threads
26:        return factor
27:    end if
28:    get_new_elliptic_curve()    ▷ Zašli hlavnímu procesu žádost o novou eliptickou křivku a bod.
29:    all_threads_go_to_step_2    ▷ Všechna vlákna začínou opět na třetí řádce.
30: end function
```

4.9.2 Paralelizace nad sdílenou pamětí

Nyní si rozebereme paralelizaci nad sdílenou pamětí. Každý proces má dostupný určitý počet vláken. Tato vlákna využijeme k paralelizaci cyklu uvedeném v algoritmu 8.

Nyní si rozebereme paralelizaci nad sdílenou pamětí. Každý proces má dostupný určitý počet vláken. Tato vlákna využijeme k paralelizaci cyklu uvedeném v algoritmu 8.

Naše paralelizace tedy spočívá v tom, že každé vlákno procesu bude vybírat následující, dosud nevybranou hodnotu k z množiny $\{2, 3, \cdots, K\}$. Touto hodnotou k vynásobí svojí kopii vygenerovaného bodu a tento nový bod si uloží, aby jej v další iteraci mohl opět roznásobit další hodnotou k. Jakmile všechna vlákna dopočítají, a to ať už byla předběžně ukončena nebo doběhla na konec intervalu, jedno vlákno sesbírá všechny vypočtené body a sečte je dohromady. Nad tímto novým bodem, který vznikl součtem všech ostatních, se vyzkouší vyhledání dělitele. Pokud dělitel nebyl nalezen, požádá některé vlákno hlavní proces o novou eliptickou křivku, pokud se jedná o vlakno pracovního procesu. Popišme si tento princip pseudokódem.

Algoritmus 12 obsahuje popis paralelizace výpočtu. Objasněme si nyní kroky algoritmu. Proměnná *test_counter* nám zůstává stejná jako u sekvenčního algoritmu, to je však nyní pro všechna vlákna společná proměnná, abychom indikovali, kolik iterací opravdu proběhlo (zde při inkrementaci počítadla předpokládáme atomicitu operace). Jak již bylo zmíněno, každé vlákno nám nyní napočítává násobky bodu. Pokud již vlákno ukončilo svůj výpočet, čeká na ostatní vlákna pomocí bariéry. Dále v každé iteraci některé vlákno zkontroluje příchozí zprávu, jestli je potřeba ještě dále počítat. Jakmile všechna vlákna ukončí forcyklus vybere se náhodně jedno vlákno a začne výpočet součtu všech bodů. To nám znázorňuje řádka 22. Dále už se jen zkontroluje dělitel a případně se zašle zpráva (pokud se jedná o pracovní proces) s žádostí o novou eliptickou křivku.

Existují i další možnosti paralelizace (viz [35]), kdy rozdělí výpočet souřadnic bodů mezi procesory a získají tak tím nižší počet operací než je uvedeno v tabulce 1.1, avšak ty buď pracují nad jinými modely, než které zde používáme nebo pracují s Edwardsovými křivkami, avšak za použití jiného souřadnicového systému. Dalším možná paralelizace je algoritmu *Double–and–Add*, více o jeho paralelizaci můžeme nalézt v [36].

4.10 Dostupné implementace

Nyní máme popsanou naší implementaci vybraných algoritmů. Existují i jiné projekty, které implementují algoritmy popsané ve 3. kapitole. Jedním z takových projektů je SageMath [37], který implementuje Pollardovu $p - 1$ a ρ metodu pro čísla do jisté velikosti, Lenstrův algoritmus pro určitou velikost čísla a kvadratické síto.

Lenstrův algoritmus implementuje projekt GMP-ECM [33], jehož vývojáři se zabývají zejména touto metodou. Ti využívají Montgomeryho tvar eliptické křivky, což je popsáno v [31]. Tento projekt využívá také SageMath, jak se můžeme dočíst v dokumentaci [37]. Další implementací je také GMP-EECM[8], která využívá Edwardsových křivek a odvíjí se od GMP-ECM. Tento projekt se však zdá být již neudržovaný, jelikož poslední verze je zde z roku 2010. S těmito implementacemi si naší implementaci porovnáme v následující kapitole. Je však nutné si uvědomit, že nad těmito projekty bylo stráveno více času, než nad naší implementací. Například první verze GMP-ECM se objevili v roce 2005.

Některé z těchto projektů však nezahrnují propojení OpenMPI a OpenMP pro paralelizaci algoritmů. Můžeme v těchto projektech spíše najít použití pouze jedné z knihoven.

[8] `https://eecm.cr.yp.to/`

Kapitola 5

Analýza výsledků

V této kapitole si jako první ukážeme, jak vypadá běžná analýza moderních faktorizačních algoritmů. Dále si ukážeme naměřené výsledky implementace, která byla spuštěna na školním klastru STAR. Pro měření využijeme několik hodnot, které mají jistou délku v cifrách, abychom mohli následně rozhodnout, který model byl efektivnější pro danou sadu čísel.

Implementace budeme spouštět jak v paralelní verzi tak i v sekvenční verzi. Testování probíhá tak, že spustíme implementace nad testovacími daty 50krát a budeme vybírat minimální čas, který je potřeba pro výpočet.

5.1 Analýza moderních faktorizačních algoritmů

Dříve existovala *RSA Challenge* (viz [38]) od roku 1991, která dříve také nabízela i peněžní výhru, pokud se někomu povede faktorizovat čísla vygenerované společností RSA Laboratories.

Od roku 2007 je tato výzva zrušena a společnost nevytváří nové klíče RSA, které byly pro výzvu použity. Společnost však stále nechává vygenerované klíče dostupné, ale již bez ceny.

Faktorizace nad těmito klíči se stále provádí. Příkladem může být zpráva z dubna roku 2020, kdy bylo faktorizováno 250 ciferné číslo pomocí algoritmu GNFS (více viz [39]).

Stroje, na kterých tyto faktorizace probíhají, jsou vždy popsané v publikacích, které popisují skutečnost, že byl prolomen další klíč. Například u posledního, tedy 250 ciferného čísla, běžel algoritmus na několika desítkách tísíc strojích po celém světě a výpočet trval několik měsíců.

5.2 Konfigurace klastru STAR

Nyní si popíšeme konfiguraci klastru STAR. V popisu vycházíme z [24]. Dostupných uzlů pro měření máme 3. Každý uzel reprezentuje jeden procesor,

který má tyto parametry:

- **Model: Intel® Xeon® CPU E5-2630 v4 @ 2.20GHz**
- **#CPU jader: 20**
- **#sockets, #cores per socket, #threads per socket: (2, 10, 1)**
- **CPU Cache L1 - L2 - L3: 32KB - 256KB - 25600KB**

Dostupná paměť pro uzel je **64GB RAM**. V měření jsme omezeni 10 minutami na úlohu. Po uplynutí této doby je náš program automaticky ukončen. Je tedy nutné volit vhodné délky hodnot, aby implementace zvládly v tomto čase číslo faktorizovat.

5.3 Parametry kompilace

Program jsme zkompilovali pomocí **mpic++**, který je zde použit z důvodu použití knihovny OpenMPI, která tento kompilátor vyžaduje. Byly použity následující přepínače:

- **-fopenmp** je nutný pro použití knihovny **OpenMP**,
- **-std=c++17** pro využití standardu C++17,
- **-O3** pro optimalizace 3. úrovně.

5.4 Použité testovací hodnoty

Testovací hodnoty použijeme jako složená čísla typu $n = pq$, kde p, q jsou prvočísla, jejichž délka v cifrách je přibližně stejná. Prvočísla byla volena náhodně. Zvolili jsme těchto 15 hodnot:

1. Hodnota $100003 \cdot 10007 =$

   1000730021, má 10 cifer, počet bitů je 32

2. $169399 \cdot 732097 =$

   124016499703, má 12 cifer, počet bitů je 39,

3. $9015689 \cdot 2179753 =$

   19651975144817, má 14 cifer, počet bitů je 47,

4. $66877373 \cdot 37411007 =$

   2501949869444611, má 16 cifer, počet bitů je 54,

5. $616711813 \cdot 193781603 =$
   119507403712176239, má 18 cifer, počet bitů je 59,
6. $7156658203 \cdot 3334377187 =$
   23862997847239614961, má 20 cifer, počet bitů je 67,
7. $69042605417 \cdot 73174579949 =$
   5052163649973526983733, má 22 cifer, počet bitů je 75,
8. $498796229131 \cdot 797594702249 =$
   397837229856663925015619, má 24 cifer, počet bitů je 81,
9. $6557409984317 \cdot 4779898644569 =$
   $31343755095920055847224373$, má 26 cifer, počet bitů je 87,
10. $62176039100899 \cdot 46283009871953 =$
    $28776942315058441472371857 47$, má 28 cifer, počet bitů je 94,
11. $817931922805289 \cdot 501274865319727 =$
    $410008714444926584643751636103$, má 30 cifer, počet bitů je 101,
12. $7878127367160683 \cdot 6144986336391269 =$
    $484109850275525191682490812767 27$, má 32 cifer, počet bitů je 108,
13. $78227414794747619 \cdot 13127269084928071 =$
    $1026912323828935219557287213512949$, má 34 cifer, počet bitů je 112,
14. $819251619519795947 \cdot 244278502983555437 =$
    $200125559183149097928582026802413839$, má 36 cifer, počet bitů je 120,
15. $27863963419333103749 \cdot 28010878135901419763 =$
    $780494083722154599386246544114477991487$, má 39 cifer, počet bitů je 130.

V textu s měřením budeme používat číselné pořadí těchto hodnot.

5.5 Použité hranice pro výpočet

Dále budeme také analyzovat, jak si algoritmus vede při změně maximální hranice pro násobení. Jako hraniční jsme zvolili tyto hodnoty:

1. 5000
2. 10000

3. 15000

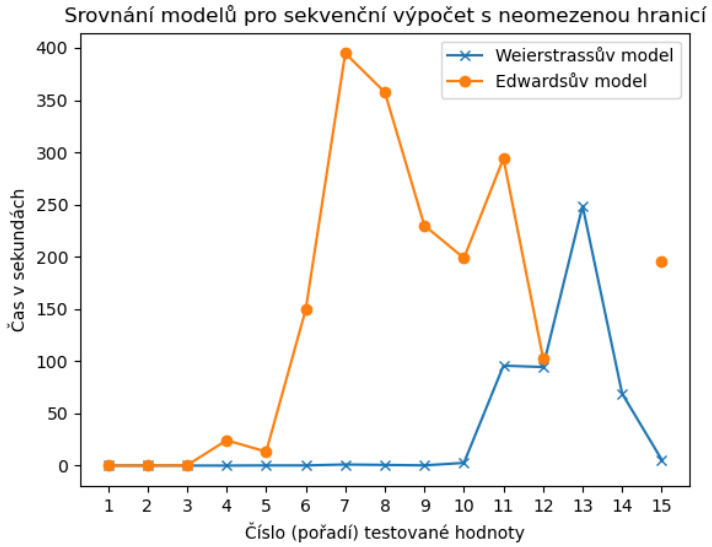
4. 25000

Jedná se o přibližné hodnoty, které by měly být použity, jak zmiňuje implementace GMP-ECM v jejich *README*. Testování probíhalo také bez určení hranice, což znamená, že se vezme maximální možná hranice, což je $\sqrt{n}$, kde n je faktorizované číslo.

5.6 Výsledky měření

V této části si ukážeme naměřené hodnoty pro jednotlivé modely. Nejdříve si vyobrazíme výsledky pro sekvenční výpočet společně s porovnáním modelů mezi sebou a následně si ukážeme, jak si vedly modely při paralelizaci. Poslední část popisuje porovnání s implementací GMP-ECM.

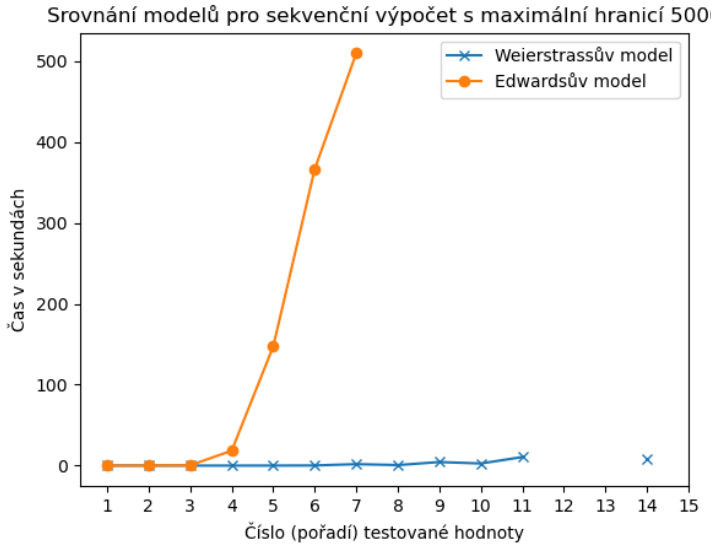
5.6.1 Výsledek sekvenčního výpočtu



Obrázek 5.1: Porovnání modelů pro neomezenou hranici. Sekvenční výpočet.

Nejdříve začneme s popisem výsledku pro neomezenou hranici u sekvenčního výpočtu, který můžeme vidět na obrázku 5.1. Zde si všimněme, že Weierstrassův model zde dokázal nalézt všechny dělitele při neomezené hranici. Edwardsův model je o něco méně efektivní v tomto případě. Pomocí Lenstrova algoritmu nad Edwardsovym modelem eliptické křivky se nepodařilo nalézt všechny dělitele, respektive nepodařilo se nalézt dělitele 13. a 14. testovaného

složeného čísla. Poslední testované číslo se však povedlo faktorizovat v řádech minut.

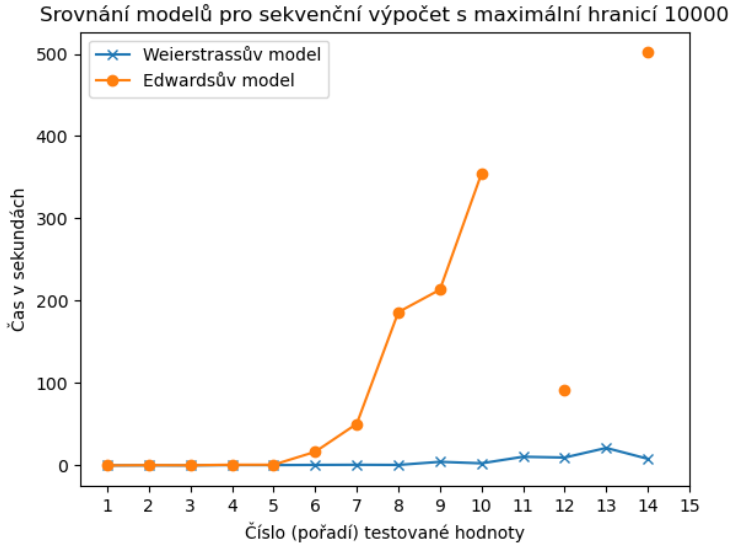


Obrázek 5.2: Porovnání modelů pro hraniční hodnotu 5000. Sekvenční výpočet.

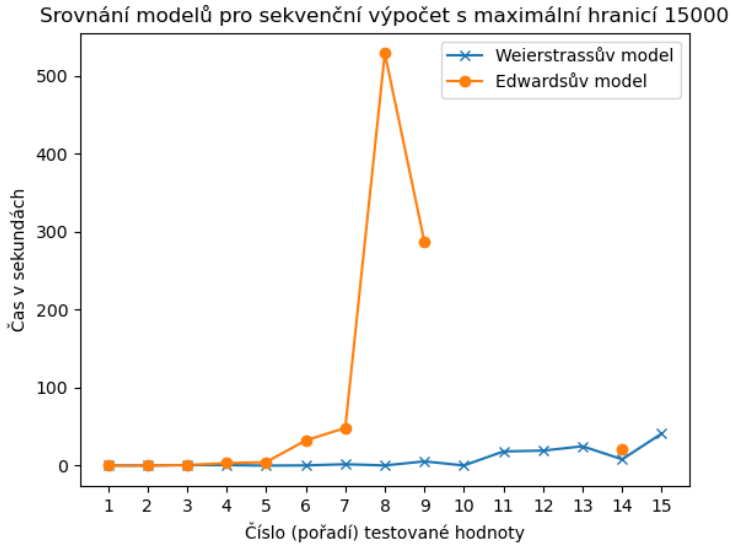
Na obrázku 5.2 můžeme vidět porovnání Weierstrassova a Edwardsova modelu při zvolené hraniční hodnotě 5000. Můžeme si všimnout, že Weierstrassův model je v tomto případě od 4. testované hodnoty mnohem efektivnější, než Edwardsův model. Oba tyto modely mají však od jisté velikosti testovaného složeného čísla problém nalézt jeho dělitele. Pro Weierstrassův model je tomu od 13. testovaného čisla, které má více než 30 cifer. Pro Edwardsův model se pak jedná o 8. číslo, které má více než 22 cifer.

Další obrázek 5.3 ukazuje jisté zlepšení pro oba modely, opět však zde vyhrává Weierstrassův model. Nyní jsme však dosáhli rychlejšího nalezení dělitele při zvýšení hraniční hodnoty na dvojnásobek oproti předešlé variantě. Pro Edwardsův model se však zdá, že opět od jisté velikosti nastává problém s nalezením dělitele. Zde se do 10 minut nepodařilo nalézt dělitele pro 11., 13. a 15. testované číslo. Weierstrassův model naopak dokázal nalézt rychle dělitele dalších testovaných hodnot. Čas se stále pohybuje v řádech jednotek sekund.

Na obrázku 5.4 můžeme opět vidět jisté zlepšení pro Weierstrassův model. Edwardsovu modelu naopak neprospělo toto zvýšení hraniční hodnoty. Pro malá testovaná čísla je Edwardsův model efektivnější, avšak ne nijak výrazně. Oba modely si počínají s faktorizací v řádech milisekund. Pro Weierstrassův model stačí tato hraniční hodnota pro nalezení dělitelů všech testovaných hodnot a to stále v řádech sekund. Maximální hodnota pro Weierstrassův model



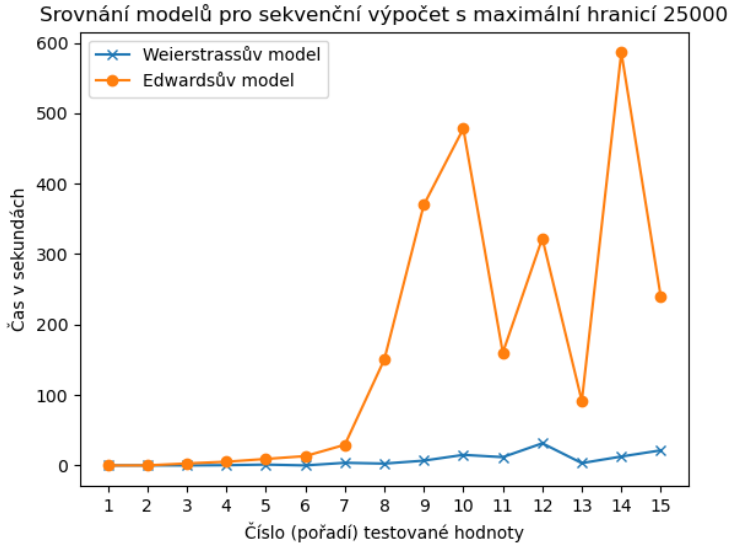
Obrázek 5.3: Porovnání modelů pro hraniční hodnotu 10000. Sekvenční výpočet.



Obrázek 5.4: Porovnání modelů pro hraniční hodnotu 15000. Sekvenční výpočet.

je zde 41 sekund. Za tuto dobu našel dělitele 15. testovaného čísla.

Obrázek 5.5 obsahuje poslední testovanou hraniční hodnotu. Tato hodnota prospěla evidentně oboum modelům, neboť se podařilo faktorizovat všechna



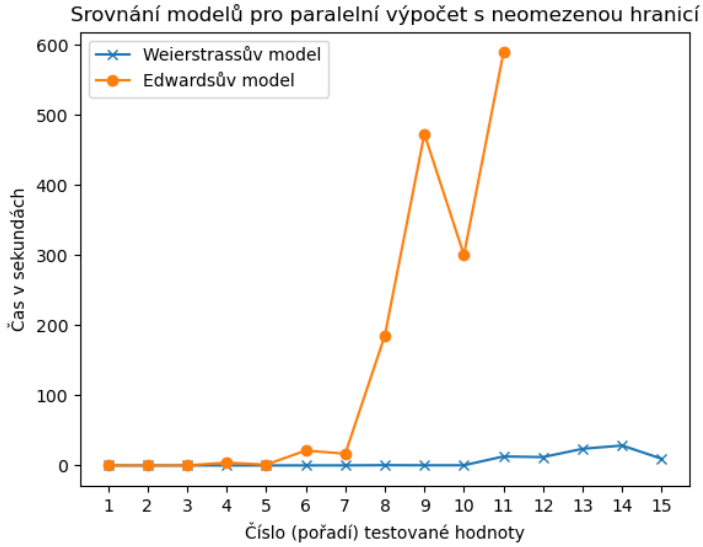
Obrázek 5.5: Porovnání modelů pro hraniční hodnotu 25000. Sekvenční výpočet.

čísla. Nejefektivnější však stále zůstává Weierstrassův model, který stále zvládá faktorizovat všechna testovaná čísla maximálně v řádech pár desítek sekund. U Edwardsova modelu si můžeme všimnout, že graf této křivky nám může vzdáleně připomínat exponenciální funkci. Takový výsledek je ovšem očekávaný, protože jak jsme si již uvedli u popisu Lenstrova algoritmu, tak jeho časová složitost je právě subexponenciální.

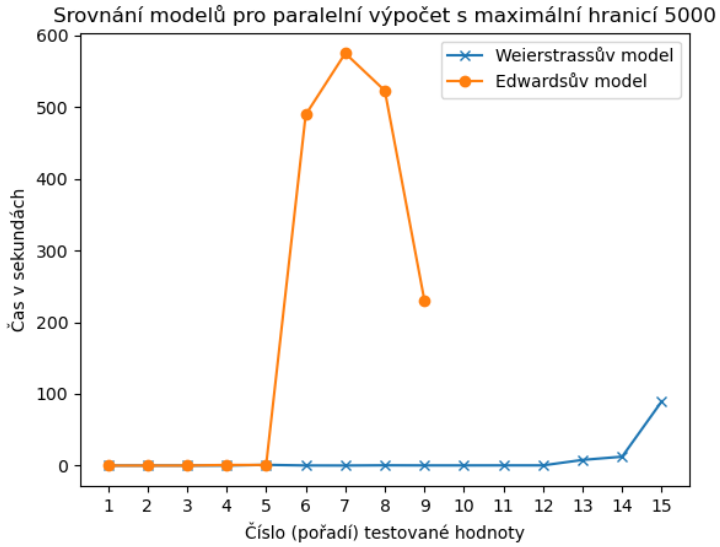
5.6.2 Výsledek paralelního výpočtu

Zde si ukážeme jak vypadají výsledky paralelizace výpočtů. Začněme opět s popisem výsledků pro neomezenou hraniční hodnotu, výsledek můžeme vidět na obrázku 5.6. Paralelizace jistě prospěla oboum modelům. Ty nyní uspěly u více testovaných čísel oproti sekvenční variantě. Pomocí Edwardsova modelu jsme byli schopni faktorizovat prvních 12 testovaných čísel. U Weierstrassova modelu se povedlo faktorizovat všechny v řádech sekund. Edwardsův model má pouhé zhoršení pro poslední testované číslo, neboť nebyl nalezen dělitel, přičemž sekvenční řešení ho bylo schopné nalézt.

Obrázek 5.7 zobrazuje porovnání modelů při spuštění paralelního výpočtu. Pro Weierstrassův model vidíme velké zlepšení při hraniční hodnotě 5000. Nyní algoritmus nalezne všechny dělitele pro testovaná čísla. Paralelizace tedy jistě tomuto modelu pomohla. Zlepšení si můžeme všimnout také u Edwardsova modelu, neboť nyní je algoritmus schopen nalézt více dělitelů. Pro prvních 5 testovaných hodnot je srovnatelný s Weierstrassovým modelem, avšak od



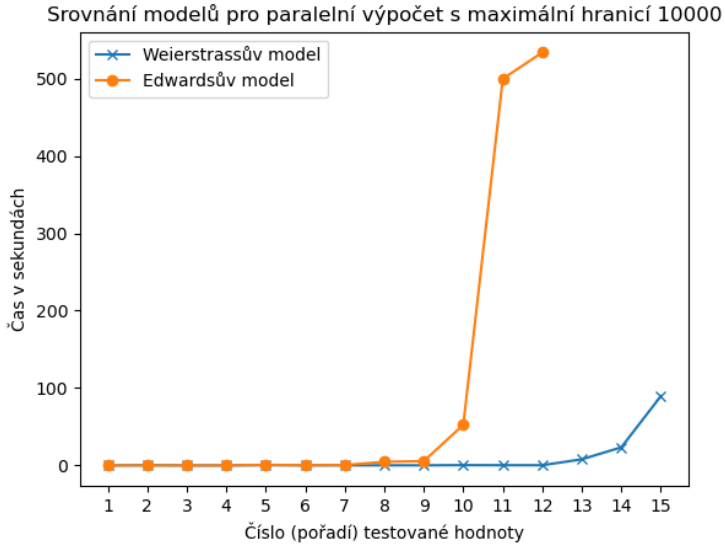
Obrázek 5.6: Porovnání modelů pro neomezenou hranici. Paralelní výpočet.



Obrázek 5.7: Porovnání modelů pro hraniční hodnotu 5000. Paralelní výpočet.

tohoto čísla pak opět roste jeho čas na nalezení.

Zvýšení hraniční hodnoty na dvojnásobek nám opět pomohlo, jak můžeme vidět na obrázku 5.8. Lenstrův algoritmus nad Edwardsovým modelem nyní dokázal nalézt více dělitelů. Weierstrassův model nyní čelí nepatrnému



Obrázek 5.8: Porovnání modelů pro hraniční hodnotu 10000. Paralelní výpočet.

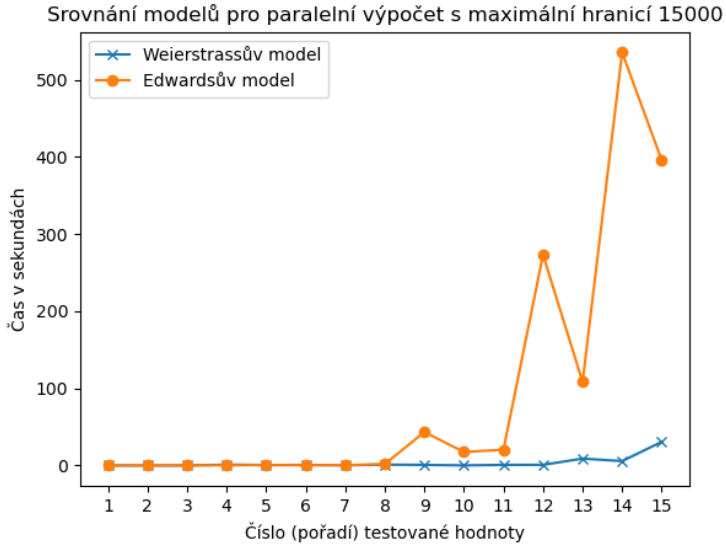
zhoršení u 14. testované hodnoty. Zde si můžeme všimnout mírného zvýšení potřebného času k nálezu dělitele. Opět si však můžeme všimnout, že potřebný čas pro výpočet se nyní mírně zvedá se zvětšující se velikostí testovaného čísla. Pomalu nám může připomínat začátek exponenciální funkce.

Na obrázku 5.4 vidíme opět jisté zlepšení pro Edwardsův model. Paralelizace dopomohla k nalezení dělitelů všech testovaných čísel. U této hranice při sekvenčním výpočtu Lenstrův algoritmus s Edwardsovým modelem stále nenalezl všechny dělitele. Weierstrassův model se také mírně zlepšil, neboť si můžeme povšimnout, že křivka je nyní mírně zploštělá oproti předešlé. Trvalo tedy méně času k nálezu dělitelů testovaných hodnot.

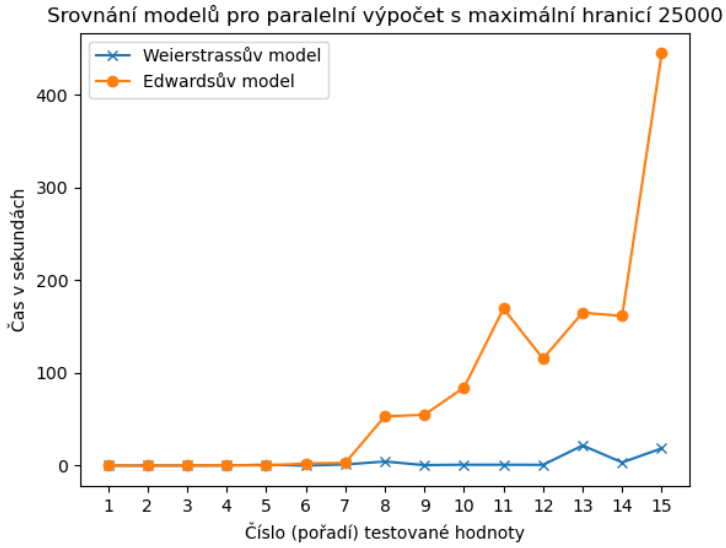
Na posledním obrázku 5.10 můžeme vidět poslední testovanou hraniční hodnotu pro paralelní výpočet, která opět dopomohla lepším výsledkům pro Edwardsův model, tedy výpočet trval kratší dobu než u ostatních hraničních hodnot. Weierstrassův model se nepatrně zhoršil oproti předešlé hraniční hodnotě.

5.6.3 Porovnání Weierstrassova modelu s jinou implementací

Nyní si porovnáme naměřené časy s implementací GMP-ECM. Tato implementace jako povinný vstupní parametr přijímá taktéž hraniční hodnotu. Porovnáme zde pouze poslední hraniční hodnotu 25000 a pouze s Weierstrassovým modelem, neboť při této hodnotě byl model nejefektivnější pro všechna testovaná čísla. Očekáváme, že implementace GMP-ECM je efektivnější, pro-

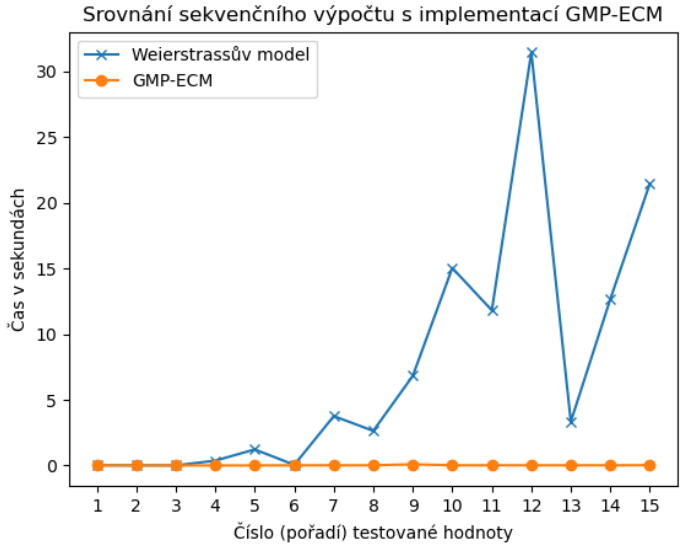


Obrázek 5.9: Porovnání modelů pro hraniční hodnotu 15000. Paralelní výpočet.



Obrázek 5.10: Porovnání modelů pro hraniční hodnotu 25000. Paralelní výpočet.

tože jak již bylo řečeno dříve, tak tato implementace využívá různých optimalizací. Toto porovnání bude ovšem postačující, protože si ukážeme, jak si počíná naše implementace při nejlepších výsledcích oproti jiné implementaci.



Obrázek 5.11: Porovnání naměřených výsledků s GMP-ECM implementací. Hraniční hodnota je 25000.

Obrázek 5.11 zobrazuje porovnání našich naměřených výsledků pomocí za použití Weierstrassova modelu při hraniční hodnotě, která se rovná 25000. GMP-ECM implementace dokáže nalézt všechny dělitele v řádech desítek milisekund. Naše implementace si zezačátku počíná velmi podobně, avšak u testovaných čísel, které dosáhnou jisté velikosti, zde tedy od 22 cifer, což je 7. testované číslo, začne doba výpočtu růst. Rozdíly jsou zde až v řádech desítek sekund.

5.7 Shrnutí výsledků

Z měření se nám zdá být nejefektivnějším modelem Weierstrassův model, pokud se zaměříme tedy pouze na projektivní souřadnicový systém. Můžeme si povšimnout, že pouze zaměření se na počet operací není pouze jediný faktor, který sledovat. Jak jsme si již naznačili při popisu Lenstrova algoritmu, je vhodné také volit lépe ostatní parametry, které například definují eliptickou křivku. Co se týče náhodné volby bodu a k němu následné dopočítání eliptické křivky, tak ta je pro Weierstrassův model optimální, neboť jsme schopni najít dělitele v řádech sekund. Pro Edwardsův model se zdá být tato volba náhodného bodu méně efektivní, protože si můžeme z výsledků všimnout, že je oproti Weierstrassovu modelu výpočet pomalejší i přes to, že operace nad tímto modelem by měly být rychlejší.

V 3. kapitole jsme si uvedli také jiné faktorizační algoritmy. V [40] jsou

uvedené doposud největší rozklady nejen pomocí Lenstrova algoritmu. Pro Pollardovu $p-1$ metodu měl nalezený prvočinitel 58 cifer. Pro Lenstrův algoritmus je to 67 cifer, pro kvadratické síto je to pak 135 cifer a pro GNFS 250 cifer. Tyto výpočty však trvaly mnohem déle než pouhých 10 minut. Někdy se jedná i o řády měsíců.

Pokud se rozhodujeme, kdy který algoritmus použít, pak v [41] je to uvedeno takto:

- Pollardova $p-1$ metoda je vhodná pro B-hladká čísla, kde B je relativně nízká hranice. Hranice hodnoty B bývá do 10^6. Algoritmus je vhodný tedy pro složená čísla do 10^{12}.
- Pollardova ρ–metoda je vhodná pro hodnoty do 2^{68}.
- Lenstrův algoritmus pak pro hodnoty do 10^{50}.
- Kvadratické síto je uváděno jako vhodné pro hodnoty od 10^{50} do 10^{100}.
- GNFS je poté nejvhodnější pro hodnoty vyšší než 10^{100}.

Závěr

Cílem této práce bylo seznámit se s Lenstrovym faktorizačním algoritmem, popsat jak funguje a implementovat jej. K tomu bylo také potřeba si popsat modely eliptických křivek, nad kterými jsme potom také Lenstrův algoritmus spouštěli, jimiž byly Edwardsovy a Weierstrassovy modely. Dále bylo nutné také tuto implementaci vhodně paralelizovat tak, aby výpočty dosáhly zefektivnění.

V první kapitole jsme si popsali základní matematické struktury, které jsou nutným základem pro Lenstrův algoritmus a také jsme si společně s tím uvedli detailnější popis modelů eliptických křivek a operace s body nad nimi.

V druhé kapitole jsme si popsali problém faktorazice a ukázali jsme si využití problému faktorizace v kryptografii. Popsali jsme si jak šifrovací algoritmus RSA funguje a jaké jiné útoky taky proti tomuto kryptosystému můžou být vedeny.

Ve třetí kapitole se věnujeme Lenstrovu faktorizačnímu algoritmu, abychom pochopili jeho fungování. Popsali jsme si zde teoretickou časovou složitost a společně s tímto algoritmem jsme si ukázali i jiné některé i efektivnější faktorizační algoritmy.

Čtvrtá kapitola se zabývá návrhem implementace a paralelizace jednotlivých algoritmů. Popsali jsme si zde také využité technologie pro implementaci.

Poslední kapitola se zabývá analýzou výsledků naměřených hodnot implementovaného faktorizačního algoritmu a to sekvenčního řešení i paralelního.

V praktické části této práce jsme úspěšně implementovali Lenstrův faktorizační algoritmus. Povedlo se také implementovat jejich paralelizaci, která jistě výpočet urychlila pro Weierstrassův i Edwardsův model. Ukázali jsme si pro určitou sadu čísel, které mají jistou délku v počtu cifer, který model je nejvhodnější. Porovnali jsme naši implementaci také s jinou, dostupnou implementací.

Jak již bylo zmíněno, mezi cíle této práce také patřilo porovnání Weierstrassova a Edwardsova modelu za použití projektivního souřadnicového systému. Z tohoto porovnání nám jistě vyšel efektivnější Weierstrassův model.

Bylo by vhodné také implementovat zmíněné optimalizace Lenstrova algoritmu. Stejně tak by určitě mohlo zlepšit výsledky použití Edwardsova modelu, pokud bychom se zaměřili na lepší volbu parametrů pro generování Edwardsovy křivky.

Příloha A

Použití aplikace

Pro kompilaci tohoto programu je nutné mít doinstalované knihovny, které jsou popsané v předešlé kapitole. Doporučený kompilátor pro tuto implementaci nosí název **mpic++**. Implementace využívá některých technik, které jsou od standardu C++11. Může být tedy potřeba tento standard specifikovat.

```
➜  ./dip --help
./dip [OPTIONS] --composite-number/-n COMPOSITE NUMBER
OPTIONS:
  -h [ --help ]                 produce help message
  -w [ --weierstrass_model ]    set Weierstrass model
  -e [ --edwards_model ]        set Edwards model
  -t [ --timer ]                time measurement
  -p [ --parallel ]             start parallel
  -b [ --bound ] arg            Maximal bound for iterations (Default square
                                root of composite number)
  -n [ --composite-number ] arg Positive integer bigger than 1 to factorize
```

Obrázek A.1: Návod k použití

Na obrázku A.1 můžeme vidět použití naší aplikace po zkompilování. Rozebereme si jednotlivé přepínače.

Přepínače aplikace jsou:

- Přepínač **-h** zobrazí nápovědu zobrazenou na obrázku A.1,
- přepínač **-w** reprezentuje použití Weierstrassova modelu,
- přepínač **-e** reprezentuje použití Edwardsova modelu,
- přepínač **-t** zapne měření času, které je následně vypsáno,
- přepínač **-p** zapne paralelní výpočet,
- přepínač **-b** určuje maximální hranici cyklu pro násobení,
- přepínač **-n** pro určení faktorizovaného čísla.

Pokud spouštět program tak, aby byl prováděn paralelní výpočet, je nutné k tomu využít spouštěcí aplikaci pro danou implementaci standardu MPI. Například takovou aplikací může být **mpirun** nebo **mpiexec**.

Příloha B

Seznam použitých zkratek

GCD – Greatest Common Divisor (největší společný násobek)

MPI – Message Passing Interface

RSA – Rivest, Shamir, Adleman (šifrova pojmenovaná podle jejich tvůrců)

SPMD – Single Program Multiple Data

Příloha C

Obsah přiložené SD karty

- `readme.txt` ..... stručný popis obsahu SD karty
- `implementation` ..... zdrojové kódy implementace
  - `CMakeLists.txt` ..... CMake konfigurační soubor
  - `src` ..... zdrojové kódy implementace Lenstrova algoritmu
  - `tests` ..... zdrojové kódy testů implementací
- `thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `src` ..... zdrojové soubory LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF

Bibliografie

1. BAUMHOF, Andreas. *Breaking RSA art*. [B.r.]. Dostupné také z: https://www.quintessencelabs.com/blog/breaking-rsa-encryption-update-state-art/. [cit. 2020-01-05].
2. KALVODA, T.; STAROSTA, Š.; PETR, I. *Matematika pro kryptologii – poznámky k přednáškám*. [B.r.]. Dostupné také z: https://courses.fit.cvut.cz/MI-MKY/@B182/media/lectures/mi-mky-poznamky-v17.pdf. [cit. 2020-11-30].
3. PELANTOVÁ, Edita; MASÁKOVÁ, Zuzana. *Teorie čísel*. České vysoké učení technické v Praze, 2017. ISBN 978-80-01-06030-8.
4. COHEN, Henri; MIYAJI, Atsuko; ONO, Takatoshi. Efficient Elliptic Curve Exponentiation Using Mixed Coordinates. In: OHTA, Kazuo; PEI, Dingyi (ed.). *Advances in Cryptology — ASIACRYPT'98*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, s. 51–65. ISBN 978-3-540-49649-6.
5. MONTGOMERY, Peter L. Speeding the Pollard and Elliptic Curve Methods of Factorization. 1987. Dostupné také z: https://pdfs.semanticscholar.org/9f50/019aa8161577e4fc62f79da41083ba03f70b.pdf?_ga=2.166466174.1973646862.1608395294-874209627.1608395294. [cit. 2020-12-01].
6. COSTELLO, Craig; SMITH, Benjamin. Montgomery curves and their arithmetic. 2017. Dostupné také z: https://eprint.iacr.org/2017/212.pdf. [cit. 2020-12-01].
7. BERNSTEIN, Daniel J.; LANGE, Tanja. Faster addition and doubling on elliptic curves. 2007. Dostupné také z: https://eprint.iacr.org/2007/286.pdf. [cit. 2020-12-01].
8. BERNSTEIN, DANIEL J.; BIRKNER, PETER; LANGE, TANJA; PETERS, CHRISTIANE. ECM USING EDWARDS CURVES. 2008. Dostupné také z: https://eprint.iacr.org/2008/016.pdf. [cit. 2020-12-31].

9. *Digital Signature Standard (DSS)*. National Institute of Standards a Technology, 2013. Tech. zpr. Dostupné z DOI: http://dx.doi.org/10.6028/nist.fips.186-4.
10. HRBEK, Michal. *ZAKLADNÍ VĚTA ARITMETIKY*. [B.r.]. Dostupné také z: http://artax.karlin.mff.cuni.cz/~michalh/nmai062/zva.pdf. [cit. 2020-12-30].
11. ZHOU, Xin; TANG, Xiaofei. Research and implementation of RSA algorithm for encryption and decryption. 2011. Dostupné z DOI: 10.1109/IFOST.2011.6021216.
12. BUČEK, Ing. Jiří. *Pokročilá kryptologie*. [B.r.]. Dostupné také z: https://moodle-vyuka.cvut.cz/pluginfile.php/218501/mod_resource/content/0/side_en.pdf. [cit. 2020-12-15].
13. POLLARD, J. M. Theorems on factorization and primality testing. *Mathematical Proceedings of the Cambridge Philosophical Society*. 1974, roč. 76, č. 3, s. 521–528. Dostupné z DOI: 10.1017/S0305004100049252.
14. CHAREST, Anna-Sophie. Pollard's p-1 and Lenstra's factoring algorithms. 2005.
15. PARKER, Daniel. ELLIPTIC CURVES AND LENSTRA'S FACTORIZATION ALGORITHM. 2004. Dostupné také z: http://math.uchicago.edu/~may/REU2014/REUPapers/Parker.pdf. [cit. 2020-11-28].
16. *IEEE Standard Specifications for Public-Key Cryptography*. [B.r.]. Dostupné také z: https://perso.telecom-paristech.fr/guilley/recherche/cryptoprocesseurs/ieee/00891000.pdf. [cit. 2021-01-01].
17. POLLARD, J. M. A monte carlo method for factorization. 1975, s. 331–334. Dostupné z DOI: https://doi.org/10.1007/BF01933667.
18. POMERANCE, Carl. The Quadratic Sieve Factoring Algorithm. 1985, s. 169–182.
19. BRIGGS, Matthew E. An Introduction to the General Number Field Sieve. 1998. Dostupné také z: https://intranet.math.vt.edu/people/brown/doc/briggs_gnfs_thesis.pdf. [2021-01-01].
20. LANDQUIST, Eric. The Quadratic Sieve Factoring Algorithm. 2001. Dostupné také z: https://www.cs.virginia.edu/crab/QFS_Simple.pdf. [cit. 2021-01-02].
21. MURPHY, Brian Antony. *Polynomial Selection for the Number Field Sieve Integer Factorisation Algorithm*. The Australian National University, 1999.
22. CASE, Michael. A Beginner's Guide To The General Number Field Sieve. [B.r.]. Dostupné také z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.219.2389&rep=rep1&type=pdf. [cit. 2021-01-01].

23. RIESEL, Hans. *Prime Numbers and Computer Methods for Factorization*. The Royal Institute of Technology, 1994. ISBN 978-0-8176-8297-2. Dostupné také z: https://epdf.pub/prime-numbers-and-computer-methods-for-factorization-second-edition.html. [cit. 2021-01-01].
24. *Popis klastru STAR*. [B.r.]. Dostupné také z: https://courses.fit.cvut.cz/MI-PDP/labs/klastr-star.html. [cit. 2020-11-29].
25. SHOUP, Victor. *NTL dokumentace*. [B.r.]. Dostupné také z: https://libntl.org/. [cit. 2020-12-30].
26. HART, William; JOHANSSON, Fredrik; PANCRATZ, Sebastian. *FLINT dokumentace*. [B.r.]. Dostupné také z: http://www.flintlib.org/index.html. [cit. 2020-12-29].
27. *Open MP*. [B.r.]. Dostupné také z: www.openmp.org. [cit. 2020-12-21].
28. *Microsoft dokumentace*. [B.r.]. Dostupné také z: https://docs.microsoft.com/en-us/windows/win32/procthread/processes-and-threads. [cit. 2021-01-02].
29. *MPI dokumentace*. [B.r.]. Dostupné také z: https://www.mpi-forum.org/docs/. [cit. 2021-01-02].
30. *Boost MPI*. [B.r.]. Dostupné také z: https://www.boost.org/doc/libs/1_75_0/doc/html/mpi.html. [cit. 2021-01-01].
31. ZIMMERMANN, Paul; DODSON, Bruce. 20 years of ECM. In: 2006, sv. 4076. ISBN 978-3-540-36075-9. Dostupné z DOI: 10.1007/11792086_37.
32. LENSTRA, Arjen; BOSMA, Wieb. An Implementation of the Elliptic Curve Integer Factorization Method. 1995. Dostupné z DOI: 10.1007/978-94-017-1108-1_9.
33. ZIMMERMANN, Paul. [B.r.]. Dostupné také z: https://gforge.inria.fr/projects/ecm/. [cit. 2021-01-01].
34. LANGR, Daniel; TVRDÍK, Pavel. *Úvod do MPI*. [B.r.]. Dostupné také z: https://courses.fit.cvut.cz/MI-PDP/media/lectures/MI-PDP-Prednaska06-MPIIntro.pdf. [cit. 2020-12-31].
35. HONE, Andrew. Efficient ECM factorization in parallel with the Lyness map. [B.r.]. Dostupné také z: https://eprint.iacr.org/2020/238.pdf. [cit. 2021-01-01].
36. NEGRE, Christophe; ROBERT, Jean-Marc. New Parallel Approaches for Scalar Multiplication in Elliptic Curve over Fields of Small Characteristic. *IEEE Transactions on Computers*. 2015, roč. 64, s. 1–1. Dostupné z DOI: 10.1109/TC.2015.2389817.
37. *Dokumentace SageMath*. [B.r.]. Dostupné také z: https://doc.sagemath.org/html/en/a_tour_of_sage/. [cit. 2020-12-27].

38. *RSA Challenge*. [B.r.]. Dostupné také z: https://web.archive.org/web/20131110032059/http://www.emc.com/emc-plus/rsa-labs/historical/the-rsa-factoring-challenge-faq.htm#. [cit. 2021-01-06].

39. *Factoring records*. [B.r.]. Dostupné také z: https://www.schneier.com/blog/archives/2020/04/rsa-250_factore.html. [cit. 2021-01-06].

40. *Methods of Computation: Factorization*. [B.r.]. Dostupné také z: https://dlmf.nist.gov/27.19. [cit. 2021-01-06].

41. LENSTRA, Arjen K. Integer Factoring. 2000, s. 31–58.