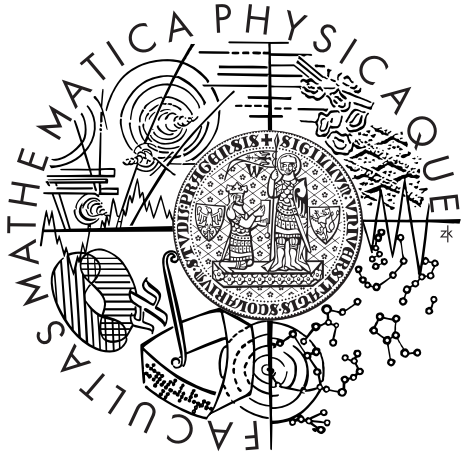
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Michaela Kučerová

Srovnání základních algoritmů pro problém diskrétního logaritmu

Katedra algebry

Vedoucí bakalářské práce:	RNDr. Přemysl Jedlička, Ph.D.
Studijní program:	Matematika
Studijní obor:	matematické metody informační bezpečnosti

Praha 2011

Děkuji RNDr. Přemyslu Jedličkovi, Ph.D., za cenné rady, náměty, jazykovou úpravu a hodiny konzultací, kterými přispěl k napsání bakalářské práce, a Milanu Boháčkovi za pomoc při psaní programové části bakalářské práce. Dále děkuji svým rodičům, protože bez jejich lásky a podpory po dobu mého studia by tato práce nemohla vzniknout, a také všem svým přátelům.

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 4. srpna 2011 Michaela Kučerová

Název práce: Srovnání základních algoritmů pro problém diskrétního logaritmu

Autor: Michaela Kučerová

Katedra: Katedra algebry

Vedoucí bakalářské práce: RNDr. Přemysl Jedlička, Ph.D., Katedra matematiky Technické fakulty České zemědělské univerzity v Praze

Abstrakt: Práce nastiňuje, co je problém diskrétního logaritmu. Také se zde uvádí podrobný popis algoritmů, které tento problém řeší a které mají široké použití (dají se aplikovat ve velkém množství grup). Součástí tohoto popisu jsou i příklady aplikace těchto algoritmů v multiplikativní grupě $\mathbb{Z}_p^*$ konečného tělesa prvočíselného řádu p nebo v podgrupě této grupy. Tento popis také zahrnuje například informace o složitosti zmíněných algoritmů. Dále práce obsahuje výsledky testování efektivity algoritmů baby-step giant-step, Pollardovo ró a Pohlig–Hellman v multiplikativní grupě $\mathbb{Z}_n^*$ okruhu řádu n a na eliptické křivce nad tělesem $\mathbb{Z}_p$.

Klíčová slova: problém diskrétního logaritmu

Title: Comparison of basic algorithms for discrete logarithm problem

Author: Michaela Kučerová

Department: Department of Algebra

Supervisor: RNDr. Přemysl Jedlička, Ph.D., Department of Mathematics of Faculty of Engineering of Czech University of Life Sciences Prague

Abstract: The presented work outlines the discrete logarithm problem. The detailed description of algorithms which solve the problem and which have wide applications (can be used in lots of groups) is presented here. In the description, there are also examples of application of these algorithms in a multiplicative group $\mathbb{Z}_p^*$ of a finite field of order p or in a subgroup of the group. The description includes information about complexity of the algorithms too. The presented work also contains results of testing of efficiency of algorithms baby-step giant-step, Pollard's rho and Pohlig–Hellman in a multiplicative group $\mathbb{Z}_n^*$ of a ring of order n and in elliptic curve over a field $\mathbb{Z}_p$.

Keywords: discrete logarithm problem

Obsah

Úvod

Tato práce si klade za cíl seznámit čtenáře s problémem diskrétního logaritmu a s různými metodami, jak tento problém řešit. Jsou zde podrobně popsány algoritmy pro problém diskrétního logaritmu, které mají široké použití (dají se aplikovat ve velkém množství grup). U každého z těchto algoritmů se uvádí jeho nároky na čas (u většiny i na paměť) a konkrétní příklad použití (buď v multiplikativní grupě $\mathbb{Z}_p^*$ konečného tělesa prvočíselného řádu p, nebo v podgrupě této grupy). Algoritmy baby-step giant-step, Pollardovo ró a Pohlig–Hellman jsou implementovány a testovány nad těmito grupami: multiplikativní grupa $\mathbb{Z}_n^*$ okruhu řádu n a eliptická křivka nad tělesem $\mathbb{Z}_p$.

1. Diskrétní logaritmus a problém diskrétního logaritmu

Cílem této kapitoly je seznámit čtenáře s pojmy diskrétní logaritmus a problém diskrétního logaritmu a nastínit souvislosti. Než se ovšem přesuneme k hlavnímu tématu kapitoly, zopakujeme si některé základní pojmy.

Definice 1. *Grupou* nazýváme množinu G s binární operací $*$, unární operací $^{-1}$ a konstantou e splňující pro každé $a,\ b,\ c \in G$

i. $a * (b * c) = (a * b) * c$,

ii. $a * e = e * a = a$,

iii. $a * a^{-1} = a^{-1} * a = e$.

Prvku e se říká *jednotka*, prvku a^{-1} *inverzní prvek* k a.

Definice 2. *Řádem grupy* G se rozumí počet prvků její nosné množiny a značí se $|G|$.

Značení 3. Nechť G je grupa a $a \in G$. Podgrupu grupy G generovanou prvkem a značíme $\langle a \rangle$.

Definice 4. *Řádem prvku* a v grupě G se rozumí počet prvků podgrupy $\langle a \rangle$. Je-li tato podgrupa nekonečná, řád prvku a je ∞.

Definice 5. Grupa G se nazývá *cyklická*, pokud je generovaná jedním prvkem.

V této kapitole jsou zmíněny tyto cyklické grupy:

- $\mathbb{Z}_n = \{0,\ 1, \ldots,\ n-1\}$ s operacemi $+$, $-$ modulo n a jednotkovým prvkem 0,
- $\mathbb{Z}_n^* = M$ s operacemi $\cdot$, $^{-1}$ modulo n a jednotkovým prvkem 1, kde M je množina obsahující právě všechna čísla

$$a \in \{0,\ 1, \ldots,\ n-1\}$$

nesoudělná s n.

Definice 6. Řekneme, že $D = \mathrm{nsd}(a,\ b)$ (*největší společný dělitel*), pokud

i. $D \mid a$ a $D \mid b$ (tj. D je společný dělitel),

ii. kdykoliv $c \mid a$ a $c \mid b$, pak $c \mid D$ (tj. D je největší takový).

Věta 7. *Rozšířený Euklidův algoritmus najde pro jakýkoliv vstup* $a,\ b \in \mathbb{Z}$ *hodnotu* $\mathrm{nsd}(a,\ b)$ *a nějaká čísla* $u,\ v \in \mathbb{Z}$ *splňující*

$$\mathrm{nsd}(a,\ b) = ua + vb.$$

Definice 8. Nechť $(G,\ *)$ a $(H,\ \circ)$ jsou grupy. Zobrazení $\varphi :\ G \to H$ se nazývá *homomorfismus* grup $(G,\ *)$, $(H,\ \circ)$, pokud platí pro každé $a,\ b \in G$

i. $\varphi(a * b) = \varphi(a) \circ \varphi(b)$,

ii. $\varphi(a^{-1}) = \varphi(a)^{-1}$,

iii. $\varphi(1) = 1$.

Definice 9. *Izomorfismem* rozumíme homomorfismus, který je bijekcí.

Nyní už snad víme vše, co potřebujeme, abychom porozuměli následujícímu. Pokud se ovšem zdá někomu nejasné cokoliv z toho, co bylo doposud řečeno, může nahlédnout například do skript [4].

Definice 10. Nechť G je konečná cyklická grupa řádu d a α je její generátor. Nechť $\beta \in G$. Řekneme, že $x \in \mathbb{Z}$ je *diskrétní logaritmus* β *o základu* α, jestliže $0 \leq x \leq d - 1$ a platí $\alpha^x = \beta$. Diskrétní logaritmus β o základu α značíme $\log_\alpha \beta$.

Poznámka 11. Z předchozí definice plyne, že $\log_\alpha \beta$ je určen jednoznačně.

Příklad 12. Protože multiplikativní grupa $\mathbb{Z}_n^*$ je cyklická pro každé celé číslo $n \geq 2$, může G z Definice 10 představovat například grupu $\mathbb{Z}_{97}^*$. Tato grupa má řád $d = 96$, což plyne mimo jiné z faktu, že 97 je prvočíslo. Generátorem této grupy je například $\alpha = 14$. Platí

$$14^{25} \equiv 17 \pmod{97},$$

a tedy $\log_{14} 17 = 25$ v $\mathbb{Z}_{97}^*$.

Fakt 13. *Nechť* G *je cyklická grupa řádu* $d \in \mathbb{N}$ *a* α *je její generátor. Nechť navíc* $\beta,\ \gamma \in G$ *a* $s \in \mathbb{Z}$. *Pak platí následující:*

- $\log_\alpha(\beta\gamma) = (\log_\alpha \beta + \log_\alpha \gamma) \bmod d$,
- $\log_\alpha \beta^s = s \log_\alpha \beta \bmod d$.

Poznámka 14. Základní pravidla pro počítání s logaritmy uvedená ve Faktu 13 lze snadno odvodit přímo z Definice 10.

Definice 15. *Problémem diskrétního logaritmu* (DLP) se nazývá následující úkol: je dáno prvočíslo p, generátor α grupy $\mathbb{Z}_p^*$ a $\beta \in \mathbb{Z}_p^*$; najdi $x \in \mathbb{Z}$, $0 \leq x \leq p-2$, takové, že $\alpha^x \equiv \beta \pmod{p}$.

Definice 16. *Zobecněným problémem diskrétního logaritmu* (GDLP) se nazývá následující úkol: je dána konečná cyklická grupa G řádu d, generátor α grupy G a $\beta \in G$; najdi $x \in \mathbb{Z}$, $0 \leq x \leq d-1$, takové, že $\alpha^x = \beta$.

Poznámka 17. Nechť α generuje cyklickou grupu G řádu $d \in \mathbb{N}$. Známe-li algoritmus, který umí spočítat diskrétní logaritmus o základu α, můžeme jej použít na spočtení diskrétního logaritmu o libovolném základu γ, kde γ také generuje G.

Důkaz. Nechť $\beta \in G$ a nechť $x = \log_\alpha \beta$, $y = \log_\gamma \beta$ a $z = \log_\alpha \gamma$. Pak platí $\alpha^x = \beta = \gamma^y = (\alpha^z)^y$. Z toho vyplývá, že $x = zy \bmod d$, což můžeme ekvivalentně upravit na $y = xz^{-1} \bmod d$. Poslední rovnost můžeme následně přepsat takto:

$$\log_\gamma \beta = (\log_\alpha \beta)(\log_\alpha \gamma)^{-1} \bmod d.$$

□

Poznámka 18. GDLP můžeme formulovat i obecněji: je dána konečná grupa G a α, $\beta \in G$; existuje-li $x \in \mathbb{Z}$ takové, že $\alpha^x = \beta$, najdi ho.

Poznámka 19. Ve formulaci z Poznámky 18 se tedy GDLP nevztahuje jen na cyklické grupy a i kdyby G byla cyklická, tak α nemusí být jejím generátorem. Toto může úkol poněkud ztížit, ale pokud G je cyklická grupa (například multiplikativní grupa konečného tělesa) a známe řád prvku α, můžeme snadno zjistit, zda-li celé číslo x splňující $\alpha^x = \beta$ existuje. Stačí si uvědomit, že řád prvku α je dělitelný řádem libovolného prvku z podgrupy, kterou α generuje. Pak totiž dostáváme následující fakt, který poslouží jako jednoduché kritérium existence hledaného $x \in \mathbb{Z}$.

Fakt 20. *Nechť G je konečná cyklická grupa, $\alpha \in G$ má řád d a $\beta \in G$. Pak existuje $x \in \mathbb{Z}$ splňující $\alpha^x = \beta$ právě tehdy, když $\beta^d = 1$.*

Poznámka 21. Ačkoliv libovolné dvě cyklické grupy stejného řádu jsou izomorfní, neznamená to, že pokud známe efektivní algoritmus na výpočet diskrétního logaritmu v jedné z nich, známe také efektivní algoritmus na výpočet diskrétního logaritmu v té druhé.

Názorným příkladem je skutečnost, že každá cyklická grupa řádu n je izomorfní aditivní grupě $\mathbb{Z}_n$. Ukážeme si, že spočítat diskrétní logaritmus v $\mathbb{Z}_n$ je snadné dokonce i v případě nejobecnějšího zadání. Toto zadání problému diskrétního logaritmu přímo pro grupu $\mathbb{Z}_n$ zní takto: jsou dány prvky α, $\beta \in \mathbb{Z}_n$; existuje-li $x \in \mathbb{Z}$ takové, že

$$\alpha x \equiv \beta \pmod{n},$$

najdi ho.

Nyní si všimněme, že řešení x neexistuje, pokud $D = \mathrm{nsd}(\alpha, n)$ nedělí β (obě strany kongruence můžeme vynásobit $\frac{n}{D}$, a zatímco pro levou stranu zřejmě platí $\alpha x \frac{n}{D} \equiv 0 \pmod{n}$, pravá strana jistě není kongruentní s nulou modulo n). Pokud však D dělí β, můžeme použít rozšířený Euklidův algoritmus pro výpočet u, $v \in \mathbb{Z}$, které splňují $\alpha u + nv = D$. Následně poslední rovnost vynásobíme $\frac{\beta}{D}$, čímž dostaneme $\alpha u \frac{\beta}{D} + nv\frac{\beta}{D} = \beta$. Uvážíme-li tuto rovnost modulo n, obdržíme kongruenci $\alpha u \frac{\beta}{D} \equiv \beta \pmod{n}$, z které vyplývá, že řešením je $x = u\frac{\beta}{D} \bmod n$.

2. Eliptické křivky

Eliptické křivky jsou v současné době hodně používané v kryptografii, protože vhodnou volbou lze zajistit bezpečný kryptosystém. Jen pro málo tříd eliptických křivek existuje subexponenciální algoritmus pro řešení GDLP. To je jistě jeden z důvodů, proč jsou předmětem četného zkoumání. Avšak tato kapitola pojednává o eliptických křivkách jen v rozsahu postačujícím pro tuto práci.

Definice 22. Nechť $p > 3$ je prvočíslo a $a, b \in \mathbb{Z}_p$ jsou prvky splňující $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$. *Eliptická křivka* $y^2 = x^3 + ax + b$ nad $\mathbb{Z}_p$ je množina $E(\mathbb{Z}_p)$ řešení $(x,\, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$, které splňují kongruenci

$$y^2 \equiv x^3 + ax + b \pmod{p},$$

spolu se symbolem 0, nazývaným *bod v nekonečnu*, tj.

$$E(\mathbb{Z}_p) = \left\{(x,\, y) \in \mathbb{Z}_p \times \mathbb{Z}_p;\; y^2 = x^3 + ax + b\right\} \cup \{0\}.$$

Definice 23. Na $E(\mathbb{Z}_p)$ definujeme operace $+$, $-$ pomocí aritmetických operací v $\mathbb{Z}_p$ následovně:

Nechť $P = (x_1,\, y_1)$ a $Q = (x_2,\, y_2)$ jsou body na $E(\mathbb{Z}_p)$. Jestliže jsou splněny rovnosti $x_2 = x_1$ a $y_2 = -y_1$, tak platí $P + Q = 0$. V opačném případě platí $P + Q = (x_3,\, y_3)$, kde

$$\begin{aligned} x_3 &= s^2 - x_1 - x_2, \\ y_3 &= s\,(x_1 - x_3) - y_1, \end{aligned}$$

a s je definováno takto:

$$s = \begin{cases} (y_2 - y_1)\,(x_2 - x_1)^{-1}, & P \neq Q \\ \left(3x_1^2 + a\right)(2y_1)^{-1}, & P = Q. \end{cases}$$

Dále platí

$$P + 0 = 0 + P = P$$

pro všechna $P \in E(\mathbb{Z}_p)$.

Fakt 24. *Množina $E(\mathbb{Z}_p)$ spolu s binární operací $+$, unární operací $-$ a konstantou 0 tvoří grupu (viz [5]).*

Fakt 25. *(Hasseho věta) Nechť $E(\mathbb{F}_q)$ je eliptická křivka nad konečným tělesem $\mathbb{F}_q$. Pak platí*

$$\left||E(\mathbb{F}_q)| - (q+1)\right| \leq 2\sqrt{q}.$$

Důkaz Hasseho věty lze najít v knize [7], kde je tato věta uvedena jako Theorem 4.2.

Poznámka 26. Fakt 25 lze využít v algoritmu Pollardova lambda, který je uveden v 3. kapitole.

3. Algoritmy pro řešení problému diskrétního logaritmu

V celé této kapitole je G grupa, $\alpha \in G$ má řád d a $\beta \in \langle\alpha\rangle$. Každý z následujících algoritmů má za úkol najít $x = \log_\alpha \beta$. Vyjádření nároků na čas a paměť při hledání x bude v souladu s následujícími definicemi.

Definice 27.

i. *Čas běhu algoritmu* $t(y)$ pro vstup y měříme jako počet grupových operací, které program provedl při výpočtu se vstupem y.

ii. *Paměť potřebná pro běh algoritmu* $s(y)$ je množství paměti spotřebované při výpočtu se vstupem y.

iii. *Očekávaná časová složitost* je:

$$E(n) = \sum_{x=1}^{\infty} x \Pr[A = x],$$

kde A je náhodná veličina – čas běhu algoritmu pro vstup délky n.

iv. *Časová složitost* (v nejhorším případě) je:

$$T(n) = \max\{t(y);\ y \text{ je vstup délky } n\}.$$

v. *Paměťová složitost* (v nejhorším případě) je:

$$S(n) = \max\{s(y);\ y \text{ je vstup délky } n\}.$$

Definice 28. Pro funkce $f, g : \mathbb{N} \to \mathbb{R}^+$ řekneme, že f je $O(g)$ právě tehdy, když $\exists c \in \mathbb{R},\ c > 0,\ \exists n_0 \in \mathbb{N}\ \forall n \in \mathbb{N},\ n \geq n_0$:

$$f(n) \leq c \cdot g(n).$$

Definice 29. Pro funkce $f, g : \mathbb{N} \to \mathbb{R}^+$ řekneme, že f je $o(g)$ právě tehdy, když $\forall \varepsilon \in \mathbb{R},\ \varepsilon > 0,\ \exists n_0 \in \mathbb{N}\ \forall n \in \mathbb{N},\ n \geq n_0$:

$$f(n) \leq \varepsilon \cdot g(n).$$

3.1 Baby-step giant-step

Nechť $m = \left\lceil \sqrt{d} \right\rceil$. Diskrétní logaritmus x můžeme vyjádřit takto:

$$x = im + j, \tag{3.1}$$

kde $0 \leq i,\ j \leq m - 1$. Nerovnost $i \leq m - 1$ plyne z následujícího:

$$x \leq d - 1 \leq m^2 - 1 = m(m-1) + m - 1.$$

Naším úkolem tedy bude najít $i,\ j$ splňující rovnost (3.1).

Vzhledem k tomu, že pro taková $i,\ j$ platí

$$\alpha^{im+j} = \beta$$

neboli

$$\alpha^j = \beta \left(\alpha^{-m}\right)^i,$$

dostaneme tato čísla, když nalezneme společný prvek v posloupnostech $1,\ \alpha, \ldots,\ \alpha^{m-1}$ a $\beta,\ \beta\alpha^{-m}, \ldots,\ \beta\left(\alpha^{-m}\right)^{m-1}$.

Algoritmus 3.1 Baby-step giant-step

VSTUP: $\alpha \in G$, $\beta \in \langle\alpha\rangle$, kde G je grupa a α má řád d.
VÝSTUP: $x = \log_\alpha \beta$.

1. $m \leftarrow \left\lceil \sqrt{d} \right\rceil$
2. Vytvoř seznam dvojic $(j,\ \alpha^j)$, $0 \leq j < m$, a seřaď ho podle druhé složky.
3. Spočti α^{-m}.
4. $\gamma \leftarrow \beta$
5. **for** $i \leftarrow 0$ to $m - 1$ **do**
6. **if** $\gamma = \alpha^j$ pro nějaké α^j ze seznamu dvojic $(j,\ \alpha^j)$ **then**
7. $x \leftarrow im + j$
8. **return** x
9. **else**
10. $\gamma \leftarrow \gamma\alpha^{-m}$
11. **end if**
12. **end for**

Příklad 30. Nechť $G = \mathbb{Z}_{107}^*$, $\alpha = 5$, $\beta = 21$. Číslo 5 generuje $\mathbb{Z}_{107}^*$ a má tedy řád $d = 106$. Pomocí algoritmu baby-step giant-step spočítáme $\log_5 21$ následovně:

1. $m \leftarrow \left\lceil \sqrt{106} \right\rceil = 11$.
2. Vytvoříme seznam dvojic $\left(j,\ 5^j \bmod 107\right)$, $0 \leq j < 11$:

(0, 1)	(1, 5)	(2, 25)	(3, 18)	(4, 90)	(5, 22)
(6, 3)	(7, 15)	(8, 75)	(9, 54)	(10, 56)	

Tabulka 3.1: Neseřazené dvojice z Příkladu 30

a seřadíme ho podle druhé složky:

(0, 1)	(6, 3)	(1, 5)	(7, 15)	(3, 18)	(5, 22)
(2, 25)	(9, 54)	(10, 56)	(8, 75)	(4, 90)	

Tabulka 3.2: Seřazené dvojice z Příkladu 30

3. Spočteme $\alpha^{-m} = 5^{-11}$:

$$5^{-1} \bmod 107 = 43,$$

tedy

$$5^{-11} = 43^{11} \bmod 107 = 60.$$

4. Počítáme $\beta\alpha^{-mi} = 21 \cdot 60^i \bmod 107$ pro $i = 0, 1, 2, \ldots$ dokud tuto hodnotu nenalezneme v Tabulce 3.2 na druhé pozici některé z dvojic. Postupně tak dostaneme tato čísla:

$$21, 83, 58, 56.$$

S číslem 56 jsme našli $i = 3$ a $j = 10$. Máme tedy rovnost $\beta\alpha^{-3m} = \alpha^{10}$ neboli $\beta = \alpha^{3m+10}$, a tedy $\log_5 21 = 43$.

Fakt 31. *Časová složitost algoritmu baby-step giant-step je*

$$O\left(\sqrt{d}\log d\right).$$

Důkaz. Rozepíšeme, co se děje na řádcích algoritmu, na nichž se provádějí grupové operace:

2. řádek: Spočtení α^j, $0 \leq j < m$ činí $O(m)$ grupových násobení. Kromě toho pro vytvoření seřazeného seznamu m dvojic potřebujeme $O(m \log m)$ grupových porovnání.

3. řádek: Spočtení inverzu prvku α je jedna grupová operace a umocnění α^{-1} na m činí $O(\log m)$ grupových násobení.

5.–12. řádek: Nejvýše m-krát se bude hledat prvek γ v seznamu dvojic. To nás bude stát $O(m \log m)$ grupových porovnání. Dále se provede maximálně $m - 1$ násobení prvku γ prvkem α^{-m}, což činí $O(m)$ grupových násobení.

Časová složitost algoritmu baby-step giant-step se tedy rovná

$$O\left(m \text{ grupových násobení} + m \log m \text{ grupových porovnání}\right),$$

což je totéž jako

$$O\left(\max\left\{m \text{ grupových násobení},\ m \log m \text{ grupových porovnání}\right\}\right).$$

Tedy určitě platí, že je tato složitost $O\left(\sqrt{d} \log \sqrt{d}\right)$ grupových operací, což můžeme ekvivalentně zapsat jako $O\left(\sqrt{d} \log d\right)$ grupových operací. □

Fakt 32. *Paměťová složitost algoritmu baby-step giant-step je* $O\left(\sqrt{d}\right)$.

Důkaz. Jediné, co zabere více než $O(1)$ míst v paměti je seznam dvojic, který zabere $O(m)$ míst v paměti. □

3.2 Pollardovo ró

Nejprve rozdělíme prvky grupy G do třech přibližně stejně velkých skupin S_1, S_2 a S_3 podle nějaké vlastnosti, kterou lze snadno otestovat. Potom definujeme funkci $f : \langle\alpha\rangle \times \mathbb{Z}_d \times \mathbb{Z}_d \to \langle\alpha\rangle \times \mathbb{Z}_d \times \mathbb{Z}_d$ následovně:

$$f(y,\ a,\ b) = \begin{cases} (\beta y,\ a,\ b+1), & y \in S_1 \\ (y^2,\ 2a,\ 2b), & y \in S_2 \\ (\alpha y,\ a+1,\ b), & y \in S_3. \end{cases}$$

Nyní vytvoříme posloupnost $(y_i,\ a_i,\ b_i)$ splňující $y_i = \alpha^{a_i}\beta^{b_i}$. Všimněme si, že když podmínku $y_i = \alpha^{a_i}\beta^{b_i}$ splňuje trojice $(y_i,\ a_i,\ b_i)$, tak ji splňuje také trojice $f(y_i,\ a_i,\ b_i)$. Naši posloupnost tedy můžeme vytvořit takto:

$$(y_i,\ a_i,\ b_i) = \begin{cases} (1,\ 0,\ 0), & i = 0 \\ f(y_{i-1},\ a_{i-1},\ b_{i-1}), & i \geq 1. \end{cases}$$

Vzhledem k tomu, že pracujeme v grupě s konečně mnoha prvky, dříve nebo později najdeme y_i a y_j takové, že $y_i = y_j$ a $i < j$. V takovém případě zřejmě platí

$$\alpha^{a_i}\beta^{b_i} = \alpha^{a_j}\beta^{b_j},$$

což můžeme přepsat následujícím způsobem:

$$\alpha^{a_i + x b_i} = \alpha^{a_j + x b_j}.$$

Protože α má řád d, tak z předchozího plyne kongruence

$$a_i + xb_i \equiv a_j + xb_j \pmod{d},$$

kterou můžeme přepsat do tvaru

$$x(b_i - b_j) \equiv a_j - a_i \pmod{d}.$$

Pokud tedy $\mathrm{nsd}(b_i - b_j,\, d) = 1$, tak $x = (a_j - a_i)(b_i - b_j)^{-1} \bmod d$. Všimněme si však, že pokud bude $1 \in S_2$, tak $y_i = (1,\, 0,\, 0)$ pro všechna $i \geq 0$, čehož bychom se měli vyvarovat. Ale pokud i přesto platí $b_i - b_j = 0$, můžeme vytvořit novou posloupnost, tentokrát s počáteční trojicí $(\alpha^{a_0}\beta^{b_0},\, a_0,\, b_0)$, kde $(a_0,\, b_0)$ je náhodně vybraná dvojice z množiny $\mathbb{Z}_d \times \mathbb{Z}_d \setminus \{0\}$ nebo z množiny $\mathbb{Z}_d \setminus \{0\} \times \mathbb{Z}_d$.

Pokud budeme pracovat v grupě, kde se funkce f dostatečně podobá náhodné funkci, dříve nebo později zmíněným postupem najdeme y_i a y_j takové, že $y_i = y_j$ a $b_i \neq b_j$. Není ovšem vhodné porovnávat dvojici y_i a y_j pro všechna i, j, $i \neq j$, protože by to bylo náročné na čas i na paměť. Budeme tedy porovnávat pouze dvojice y_i a y_{2i}, což je důvod, proč nepoužíváme náhodnou funkci místo funkce f. Z definice f totiž plyne, že jakmile jednou najdeme y_{i_0} a y_{j_0} splňující $y_{i_0} = y_{j_0}$ a $i_0 < j_0$, prvky y_i z posloupnosti $(y_i,\, a_i,\, b_i)$ se začnou od indexu i_0 periodicky opakovat. Ostatně i řecké písmeno ró v názvu tohoto algoritmu má značit posloupnost, která je od jistého prvku periodická.

Algoritmus 3.2 Pollardovo ró

VSTUP: $\alpha \in G$, $\beta \in \langle\alpha\rangle$, kde G je grupa a α má řád d.
VÝSTUP: $x = \log_\alpha \beta$.

1. Rozděl G do skupin S_1, S_2 a S_3.
2. $(y_0, a_0, b_0) \leftarrow (1,\, 0,\, 0)$
3. **repeat**
4. $\quad (y_i,\, a_i,\, b_i) \leftarrow f(y_i,\, a_i,\, b_i)$
5. $\quad (y_{2i},\, a_{2i},\, b_{2i}) \leftarrow f\left(f(y_{2i},\, a_{2i},\, b_{2i})\right)$
6. **until** $y_i = y_{2i}$
7. **if** $\mathrm{nsd}(b_i - b_{2i},\, d) \neq 1$ **then**
8. $\quad$ **return** „Algoritmus Pollardovo ró selhal.“
9. **else**
10. $\quad x \leftarrow (a_{2i} - a_i)\,(b_i - b_{2i})^{-1} \bmod d$
11. $\quad$ **return** x
12. **end if**

Poznámka 33. Do situace, kdy platí $\mathrm{nsd}(b_i - b_{2i},\, d) \neq 1$, se můžeme dostat v následujících případech:

- Dvojice $(a_i,\ b_i)$ a $(a_{2i},\ b_{2i})$ jsou totožné, a tedy $b_i - b_{2i} = 0$.
- Řád d prvku α není prvočíslo.

Vzhledem k předchozí poznámce tedy není vhodné používat tento algoritmus, pokud prvek α nemá prvočíselný řád.

Příklad 34. Nechť $G = \mathbb{Z}_{293}^*$, $\alpha = 40$ a $\beta = 26$. Číslo 40 má řád $d = 73$ v $\mathbb{Z}_{293}^*$ a $26 \in \langle 40 \rangle$. Pomocí algoritmu Pollardovo ró spočítáme $\log_{40} 26$ následovně:

1. Rozdělíme prvky $\mathbb{Z}_{293}^*$ do skupin S_1, S_2 a S_3 takto:

$$\begin{aligned} S_1 &= \{y \in \mathbb{Z}_{293}^* :\ y \equiv 1 \pmod 3\} \\ S_2 &= \{y \in \mathbb{Z}_{293}^* :\ y \equiv 0 \pmod 3\} \\ S_3 &= \{y \in \mathbb{Z}_{293}^* :\ y \equiv 2 \pmod 3\}. \end{aligned}$$

2. Vytváříme trojice $(y_i,\ a_i,\ b_i)$ a $(y_{2i},\ a_{2i},\ b_{2i})$ (Tabulka 3.3), dokud nenajdeme trojice splňující $y_i = y_{2i}$.

i	$(y_i,\ a_i,\ b_i)$	$(y_{2i},\ a_{2i},\ b_{2i})$
1	(26, 0, 1)	(161, 1, 1)
2	(161, 1, 1)	(53, 3, 1)
3	(287, 2, 1)	(73, 8, 2)
4	(53, 3, 1)	(33, 9, 3)
5	(69, 4, 1)	(150, 36, 12)
6	(73, 8, 2)	(172, 72, 25)
7	(140, 8, 3)	(150, 0, 26)
8	(33, 9, 3)	(172, 0, 53)
9	(210, 18, 6)	(150, 1, 54)
10	(150, 36, 12)	(172, 2, 36)
11	(232, 72, 24)	(150, 3, 37)
12	(172, 72, 25)	(172, 6, 2)

Tabulka 3.3: Trojice z Příkladu 34

3. Zjistili jsme, že $y_{12} = 172 = y_{24}$, a tedy hodnoty, které potřebujeme k výpočtu jsou $a_{12} = 72$, $b_{12} = 25$, $a_{24} = 6$ a $b_{24} = 2$. Spočítáme-li

$$(a_{2i} - a_i)\,(b_i - b_{2i})^{-1} \bmod d$$

neboli

$$(6 - 72)\,(25 - 2)^{-1} \bmod 73 = 7 \cdot 54 \bmod 73 = 13,$$

dostáváme, že $\log_{40} 26 = 13$.

Fakt 35. *Očekávaná časová složitost algoritmu Pollardovo ró je* $O\left(\sqrt{d}\right)$.

Důkaz. Jediné řádky, na nichž se provádí grupové operace jsou 3.–6., které obsahují repeat cyklus. Každý průchod tímto cyklem nás stojí $O(1)$ grupových násobení. Otázkou tedy je, kolik průchodů se v průměrném případě provede.

Budeme předpokládat, že se funkce f chová jako náhodná funkce a k důkazu použijeme narozeninový paradox. Označme písmenem A náhodnou veličinu – počet aplikací funkce f nutný k nalezení prvků y_{i_0} a y_{j_0} splňujících $(y_{i_0} = y_{j_0} \wedge i_0 < j_0)$. Podle narozeninového paradoxu platí

$$\Pr[A \leq n] \approx 1 - e^{-n^2/2d}.$$

Odtud plyne

$$\Pr[A = n] \approx \left(1 - e^{-n^2/2d}\right)' = e^{\left(-n^2/2d\right)}\frac{n}{d}.$$

Tedy platí

$$\int_0^\infty z \Pr[A = z]\, dz \approx \sqrt{\frac{\pi}{2}}\sqrt{d},$$

a proto střední hodnota A je $O\left(\sqrt{d}\right)$.

Z periodičnosti posloupnosti y_i od indexu i_0 plyne, že pokud $u, v \geq i_0$ a $u \equiv v \pmod{(j_0 - i_0)}$, tak $y_u = y_v$. Tedy určitě $y_i = y_{2i}$ pro nejmenší $i \geq i_0$, které je zároveň násobkem $(j_0 - i_0)$, neboli pro i splňující $i = (j_0 - i_0)\left\lceil \frac{i_0}{j_0 - i_0} \right\rceil$. Zřejmě platí

$$(j_0 - i_0)\left\lceil \frac{i_0}{j_0 - i_0} \right\rceil < (j_0 - i_0)\left(\frac{i_0}{j_0 - i_0} + 1\right) = i_0 + j_0 - i_0 = j_0,$$

a tedy $i = (j_0 - i_0)\left\lceil \frac{i_0}{j_0 - i_0} \right\rceil$ splňuje

$$i_0 \leq i < j_0.$$

Tedy očekáváme, že se y_i objeví na 4. řádku po $O\left(\sqrt{d}\right)$ průchodech. V takovém případě se provede $O\left(\sqrt{d}\right)$ průchodů cyklem. Tedy očekávaný čas běhu algoritmu Pollardovo ró je

$$O\left(\sqrt{d}\right) \cdot O(1) = O\left(\sqrt{d}\right)$$

grupových operací. □

Poznámka 36. Povšimněme si, že $\Pr[A \leq n] \approx 1 - e^{-n^2/2d}$ z předchozího důkazu je pro každé $n \in \mathbb{N}$ menší než 1, a tedy maximální časovou složitost má algoritmus Pollardovo ró nepříjemně velkou.

Fakt 37. *Paměťová složitost algoritmu Pollardovo ró je* $O(1)$.

3.3 Pohlig–Hellman

Nechť $d = p_1^{e_1} p_2^{e_2} \dots p_r^{e_r}$ je prvočíselný rozklad čísla d. Určíme-li hodnoty

$$x_i = x \pmod{p}_i^{e_i}$$

pro každé $i \in \{1, 2, \dots, r\}$, můžeme podle Čínské věty o zbytcích spočítat $x \pmod{d}$.

Ukážeme tedy, jak spočítat $x_i = x \pmod{p}_i^{e_i}$. Nejprve x_i vyjádříme takto:

$$x_i = a_0 + a_1 p_i + \ldots + a_{e_i - 1} p_i^{e_i - 1},$$

kde $0 \leq a_j \leq p_i - 1$. Abychom získali x_i, stačí tedy zjistit hodnotu prvků $a_0, , a_1, \dots, a_{e_i - 1}$. Všimněme si, že platí rovnost

$$x = x_i + b p_i^{e_i}$$

pro nějaké $b \in \mathbb{Z}$. Pro výpočet a_0 použijeme rovnost

$$\beta^{d/p_i} = \alpha^{a_0 d/p_i}, \tag{3.2}$$

která plyne z následujícího:

$$\begin{aligned}
\beta^{d/p_i} &= (\alpha^x)^{d/p_i} \\
&= \left(\alpha^{a_0 + a_1 p_i + \ldots + a_{e_i-1} p_i^{e_i-1} + b p_i^{e_i}}\right)^{d/p_i} \\
&= \left(\alpha^{a_0 + K p_i}\right)^{d/p_i}, \text{ kde } K \in \mathbb{Z} \\
&= \alpha^{a_0 d/p_i} \alpha^{Kd} \\
&= \alpha^{a_0 d/p_i}.
\end{aligned}$$

Prvek a_0 tedy můžeme získat například použitím algoritmu baby-step giant-step nebo Pollardovo ró na výpočet diskrétního logaritmu $\log_{\overline{\alpha}} \overline{\beta}$, kde $\overline{\alpha} = \alpha^{d/p_i}$ a $\overline{\beta} = \beta^{d/p_i}$. Pokud $e_i > 1$ použijeme zobecněnou verzi rovnosti 3.2 k získání $a_1, \dots, a_{e_i - 1}$. Označme $\beta_0 = \beta$ a definujme

$$\beta_j = \beta \alpha^{-\left(a_0 + a_1 p_i + \ldots + a_{j-1} p_i^{j-1}\right)}$$

pro $1 \leq j \leq e_i - 1$. Nyní zobecníme rovnost 3.2 na následující rovnost:

$$\beta^{d/p_i} = \alpha^{a_0 d/p_i}. \tag{3.3}$$

Můžeme si všimnout, že pro $j = 0$ se z rovnosti 3.3 stává rovnost 3.2. Pro všechna j dokážeme rovnost 3.3 takto:

$$\begin{aligned}
\beta_j^{d/p_i^{j+1}} &= \left(\alpha^{x-\left(a_0+a_1p_i+\ldots+a_{j-1}p_i^{j-1}\right)}\right)^{d/p_i^{j+1}} \\
&= \left(\alpha^{a_jp_i^j+\ldots+a_{e_i-1}p_i^{e_i-1}+bp_i^{e_i}}\right)^{d/p_i^{j+1}} \\
&= \left(\alpha^{a_jp_i^j+K_jp_i^{j+1}}\right)^{d/p_i^{j+1}}, \text{ kde } K_j \in \mathbb{Z} \\
&= \alpha^{a_jd/p_i}\alpha^{K_jd} \\
&= \alpha^{a_jd/p_i}.
\end{aligned}$$

Stejně jako prvek a_0 můžeme tedy i prvky $a_1, \ldots, a_{e_i-1}$ získat pomocí algoritmu baby-step giant-step nebo Pollardovo ró, který bude mít tentokrát za úkol spočítat $\log_{\overline{\alpha}} \overline{\beta}$, kde $\overline{\alpha} = \alpha^{d/p_i}$ a $\overline{\beta} = \beta_j^{d/p_i^{j+1}}$. Nyní zbývá nahlédnout, že β_{j+1} lze dostat z β_j a a_j pomocí rovnosti

$$\beta_{j+1} = \beta_j \alpha^{-a_j p_i^j}.$$

Algoritmus 3.3 Pohlig–Hellman

VSTUP: $\alpha \in G$, $\beta \in \langle\alpha\rangle$, kde G je grupa a α má řád d.
VÝSTUP: $x = \log_\alpha \beta$.

1. Najdi prvočíselný rozklad d: $d = p_1^{e_1}p_2^{e_2}\ldots p_r^{e_r}$, kde $e_i \geq 1$.
2. **for** $i \leftarrow 1$ to r **do**
3. $\quad \overline{\alpha} \leftarrow \alpha^{d/p_i}$
4. $\quad \beta_0 \leftarrow \beta$
5. $\quad$ **for** $j \leftarrow 0$ to $e_i - 1$ **do**
6. $\quad\quad \overline{\beta} \leftarrow \beta_j^{d/p_i^{j+1}}$
7. $\quad\quad a_j \leftarrow \log_{\overline{\alpha}} \overline{\beta}$
8. $\quad\quad \beta_{j+1} \leftarrow \beta_j \alpha^{-a_j p_i^j}$
9. $\quad$ **end for**
10. $\quad x_i \leftarrow a_0 + a_1p_i + \ldots + a_{e_i-1}p_i^{e_i-1}$
11. **end for**
12. Spočti $x \in \mathbb{Z}$, $0 \leq x \leq d-1$, splňující

$$x \equiv x_i \pmod{p_i^{e_i}}$$

pro všechna $i \in \{1, 2, \ldots, r\}$.
13. **return** x

Příklad 38. Nechť $G = \mathbb{Z}_{73}^*$, $\alpha = 11$ generuje $\mathbb{Z}_{73}^*$ a $\beta = 19$. Prvek α má tedy řád $d = 72$. Pomocí algoritmu Pohlig–Hellman spočítáme $\log_{11} 19$ následovně:

1. Najdeme prvočíselný rozklad čísla 72: $72 = 2^3 \cdot 3^2$.

2. Postupně spočteme pro $p_1 = 2$ a $e_1 = 3$ následující:
   $\overline{\alpha} = \alpha^{d/p_1} = 11^{72/2} \bmod 73 = 72$,
   $\overline{\beta} = \beta_0^{d/p_1} = 19^{72/2} \bmod 73 = 1$, $\log_{\overline{\alpha}} \overline{\beta} = 0$,
   $\overline{\beta} = \beta_1^{d/p_1^2} = 19^{72/2} \bmod 73 = 72$, $\log_{\overline{\alpha}} \overline{\beta} = 1$,
   $\overline{\beta} = \beta_2^{d/p_1^3} = 19^{72/2} \bmod 73 = 1$, $\log_{\overline{\alpha}} \overline{\beta} = 0$.
   Tedy $x_1 = 0 + 1 \cdot 2 + 0 \cdot 2^2 = 2$.

3. Postupně spočteme pro $p_2 = 3$ a $e_2 = 2$ následující:
   $\overline{\alpha} = \alpha^{d/p_2} = 11^{72/3} \bmod 73 = 8$,
   $\overline{\beta} = \beta_0^{d/p_2} = 19^{72/3} \bmod 73 = 64$, $\log_{\overline{\alpha}} \overline{\beta} = 2$,
   $\overline{\beta} = \beta_1^{d/p_2^2} = 19^{72/3} \bmod 73 = 64$, $\log_{\overline{\alpha}} \overline{\beta} = 2$.
   Tedy $x_2 = 2 + 2 \cdot 3 = 8$.

4. Nyní zbývá vyřešit systém kongruencí

$$\begin{aligned} x &\equiv 2 \pmod 8, \\ x &\equiv 8 \pmod 9. \end{aligned}$$

   Platí

$$\begin{aligned} x &= \left(x_1 \frac{d}{p_1^{e_1}} \left(\left(\frac{d}{p_1^{e_1}} \right)^{-1} \bmod p_1^{e_1} \right) \right. \\ &+ \left. x_2 \frac{d}{p_2^{e_2}} \left(\left(\frac{d}{p_2^{e_2}} \right)^{-1} \bmod p_2^{e_2} \right) \right) \bmod d \\ &= (2 \cdot 9 \cdot 1 + 8 \cdot 8 \cdot 8) \bmod 72 = 26, \end{aligned}$$

   tedy $\log_{11} 19 = 26$.

V následujících dvou faktech budeme předpokládat, že na výpočet dílčích logaritmů $(\log_{\overline{\alpha}} \overline{\beta})$ se používá algoritmus baby-step giant-step.

Fakt 39. *Časová složitost algoritmu Pohlig–Hellman při předem známé faktorizaci čísla d je $O\left(\sum_{i=1}^{r} e_i \left(\log d + \sqrt{p_i} \log p_i\right)\right)$.*

Důkaz. Všechny grupové operace se provádějí na 2.–11. řádku.

Spočtení α^{d/p_i} činí $O\left(\log(d/p_i)\right.$ grupových násobení. Tedy pro i od jedné do r dostáváme

$$O\left(\sum_{i=1}^{r}\left(\log d - \log p_i\right)\right) = O\left(r \log d\right)$$

grupových násobení.

Pro i od jedné do r se provedou grupové operace na 5.–9. řádku. Spočtení $\beta_j^{d/p_i^{j+1}}$ pro j od nuly do $e_i - 1$ nás bude stát

$$O\left(\sum_{j=0}^{e_i-1}\left(\log d - \log p_i^{j+1}\right)\right)$$

grupových násobení. Dále se e_i-krát spočte $\log_{\overline{\alpha}} \overline{\beta}$, což činí

$$O\left(e_i \sqrt{p_i} \log p_i\right)$$

grupových operací, protože řád prvku $\overline{\alpha}$ je p_i. Výpočet $\beta_j \alpha^{-a_j p_i^j}$ pro j od nuly do $e_i - 1$ nás bude stát

$$\begin{aligned} O\left(\sum_{j=0}^{e_i-1}\left(\log a_j + \log p_i^j\right)\right) &= O\left(\sum_{j=0}^{e_i-1}\left(\log p_i + \log p_i^j\right)\right) \\ &= O\left(\sum_{j=1}^{e_i-1}\left(\log p_i^j\right)\right) \end{aligned}$$

grupových operací. Tedy celkem se na těchto řádcích při jednom průchodu provede $O\left(\sum_{j=0}^{e_i-1}\left(\log d - \log p_i^{j+1} + \log p_i^{j+1}\right) + e_i\sqrt{p_i}\log p_i\right)$ neboli

$$O\left(e_i\left(\log d + \sqrt{p_i}\log p_i\right)\right)$$

grupových operací.

Časová složitost algoritmu Pohlig–Hellman tedy je

$$O\left(r \log d + \sum_{i=1}^{r} e_i\left(\log d + \sqrt{p_i}\log p_i\right)\right)$$

grupových operací, což vzhledem k tomu, že $e_i \geq 1$ pro každé $i \in \{1,\ 2, \ldots,\ r\}$, činí $O\left(\sum_{i=1}^{r} e_i\left(\log d + \sqrt{p_i}\log p_i\right)\right)$ grupových operací. □

Fakt 40. *Paměťová složitost algoritmu Pohlig–Hellman je*

$$O\left(r + \sqrt{\max\left\{p_1,\, p_2, \ldots p_r\right\}}\right).$$

Důkaz. Jediné, co zabere více než $O\left(1\right)$ míst v paměti je prvočíselný rozklad čísla d a výpočet $\log_{\overline{\alpha}} \overline{\beta}$. Prvočíselný rozklad čísla d zabere $O\left(r\right)$ míst v paměti. Výpočet $\log_{\overline{\alpha}} \overline{\beta}$ zabere $O\left(\sqrt{p_i}\right)$ míst v paměti, kde p_i, $1 \leq i \leq r$, je řád prvku $\overline{\alpha}$. Výsledná paměťová složitost dále plyne z faktu, že nikdy nepočítáme více dílčích logaritmů najednou. □

3.4 Pollardova lambda

Tomuto algoritmu na řešení diskrétního logaritmu se také říká „kangaroo method“. Můžeme ho totiž popsat pomocí dvou klokanů, jak si brzy ukážeme.

Nechť a a b jsou celá čísla splňující

$$a \leq x < b.$$

Uvažujme dva klokany, krotkého a divokého. Krotkému budeme říkat K a tomu divokému L. Každý z klokanů začíná skákat z určitého bodu grupy G. Startovní bod K je $k_0 = \alpha^b$ a startovní bod L je $l_0 = \beta$.

Označme u_i vzdálenost klokana K od α^0 po právě i skocích a v_i vzdálenost klokana L od α^x po právě i skocích. Pak platí $u_0 = b$ a $v_0 = 0$. Klokan K tedy začíná ve vzdálenosti b od α^0 a klokan L je na začátku od α^0 vzdálen x. Fakt, že x na rozdíl od b neznáme, je důvodem, proč se klokanovi L říká divoký.

Nechť

$$S = \{\alpha^{s_1},\, \alpha^{s_2}, \ldots, \alpha^{s_r}\}$$

je množina skoků a

$$h:\, G \to \{1,\, 2, \ldots,\, r\}$$

je hašovací funkce. Prvky $s_1\, , s_2, \ldots, s_r$ interpretujeme jako vzdálenosti, které by měly být malé v porovnání s délkou intervalu $[a,\, b]$. Vhodnou volbou pro dosazení za $s_1\, , s_2, \ldots, s_r$ by mohly být podle Pollarda [2] mocniny dvojky počínaje 2^0 a konče při určité velikosti.

Necháme skákat klokana K grupou G po cestě tvořené prvky k_i, $i \in \mathbb{N}_0$, které jsou určeny takto:

$$k_{i+1} = k_i \cdot \alpha^{s_{h(k_i)}}.$$

Po každém skoku aktualizujeme informaci o vzdálenosti K od α^0:

$$u_{i+1} = u_i + s_{h(k_i)}.$$

Po určitém počtu skoků se K zastaví a nalíčí past. Označme tento počet skoků písmenem M. Očekávaná doba běhu algoritmu je minimální, když M je přibližně $0,7\sqrt{b-a}$ a průměrná hodnota prvků $s_1, s_2, \ldots, s_r$ se rovná $0,51\sqrt{b-a}$.

Nyní začne skákat klokan L. Jeho cestu tvoří prvky l_j, $j \in \mathbb{N}_0$, s následujícím rekurentním vzorcem:

$$l_{j+1} = l_j \cdot \alpha^{s_{h(l_j)}}.$$

Opět aktualizujeme informaci o vzdálenosti, ale tentokrát o vzdálenosti L od α^x:

$$v_{j+1} = v_j + s_{h(l_j)}.$$

Po každém skoku zkontrolujeme, jestli klokan L nespadl do pasti.

1. Klokan spadl do pasti: Označíme počet skoků, které naskákal, než se dostal do pasti, písmenem N. Protože $k_M = l_N$, známe vzdálenost pasti od α^0 i vzdálenost pasti od α^x, což nám poskytuje možnost spočítat x následovně:

$$x = u_M - v_N.$$

2. Klokan nespadl do pasti: Pokud $v_j < u_M + b - a$ necháme klokana skočit znovu. V opačném případě ho zastavíme, protože je jasné, že past přeskočil. Místo něj necháme skákat dalšího divokého klokana, ale tentokrát z bodu $l_0 = \beta \cdot \alpha^z$, který má počáteční vzdálenost od α^x rovnou nějakému malému číslu z.

Povšimněme si, že pokud divoký klokan spadl do pasti, jeho cesta se musela někde napojit na cestu krotkého klokana. Cesty vytvořené klokany tedy můžeme zakreslit ve tvaru písmene λ, od čehož je odvozen název algoritmu.

Algoritmus 3.4 Pollardova lambda

VSTUP: $\alpha \in G$, $\beta \in \langle\alpha\rangle$, kde G je grupa a α má řád d, a a, $b \in \mathbb{Z}_d$ splňující $a \leq \log_\alpha \beta < b$.

VÝSTUP: $x = \log_\alpha \beta$.

1. $k_0 \leftarrow \alpha^b$
2. $l_0 \leftarrow \beta$
3. $u_0 \leftarrow b$
4. $v_0 \leftarrow 0$
5. **for** $i \leftarrow 1$ to r **do**
6. $\quad \gamma_i \leftarrow \alpha^{s_i}$
7. **end for**
8. Připrav hašovací funkci $h : G \to \{1, 2, \ldots, r\}$.
9. **for** $i \leftarrow 0$ to $M - 1$ **do**
10. $\quad k_{i+1} \leftarrow k_i \cdot \alpha^{s_{h(k_i)}}$
11. $\quad u_{i+1} \leftarrow u_i + s_{h(k_i)}$
12. **end for**
13. **while** true **do**
14. $\quad i \leftarrow 1$
15. $\quad j \leftarrow 0$
16. $\quad$ **while** $v_j < u_M + b - a$ **do**
17. $\quad\quad l_{j+1} \leftarrow l_j \cdot \alpha^{s_{h(l_j)}}$.
18. $\quad\quad v_{j+1} = v_j + s_{h(l_j)}$
19. $\quad\quad j \leftarrow j + 1$
20. $\quad\quad$ **if** $l_j = k_M$ **then**
21. $\quad\quad\quad x \leftarrow u_M - v_j$
22. $\quad\quad\quad$ **return** x
23. $\quad\quad$ **end if**
24. $\quad$ **end while**
25. $\quad l_0 \leftarrow \beta\alpha^{z_i}$
26. $\quad v_0 \leftarrow z_i$
27. $\quad i \leftarrow i + 1$
28. **end while**

Příklad 41. Nechť $G = \mathbb{Z}_{41}^*$, $\alpha = 6$ generuje $\mathbb{Z}_{41}^*$ a $\beta = 35$. Prvek α má tedy řád $d = 40$, a proto platí $0 \leq x < 40$. Položíme tedy $a = 0$ a $b = 40$. Pomocí algoritmu Pollardova lambda spočítáme $\log_6 35$ následovně:

1. $k_0 \leftarrow \alpha^b = 1$, $l_0 \leftarrow \beta = 35$, $u_0 \leftarrow b = 40$, $v_0 \leftarrow 0$.
2. Vytvoříme množinu

$$\begin{aligned} S &= \{\alpha^{s_1}, \alpha^{s_2}, \ldots, \alpha^{s_r}\} \\ &= \left\{6^{2^0}, 6^{2^1}, 6^{2^2}, 6^{2^3}\right\} \\ &= \{6, 36, 25, 10\}. \end{aligned}$$

Množinu S jsme zvolili tak, aby průměrná hodnota mocnin dvojky, které odpovídají prvkům $s_1, s_2, \ldots, s_r$, byla co nejblíže číslu $0,51\sqrt{b-a}$.

3. Definujeme hašovací funkci $h : \mathbb{Z}_{41}^* \to \{1, 2, 3, 4\}$ takto:

$$h(y) \mapsto i,$$

kde $i = y \bmod 4 + 1$.

4. Necháme krotkého klokana skočit šestkrát než nastraží past. Platí $4 < 0,7\sqrt{b-a} < 5$, takže volbou čísla 6 výrazně nezvyšujeme očekávaný čas běhu algoritmu a máme tak větší šanci, že se divoký klokan stihne napojit na trasu krotkého.

i	k_i	u_i
0	1	40
1	36	42
2	11	43
3	28	51
4	4	52
5	24	53
6	21	54

Tabulka 3.4: Skoky krotkého klokana z Příkladu 41

5. Nyní necháme skákat divokého klokana. Pokud klokan nespadne do pasti, dokud platí $(v_j < u_6 + b - a) \wedge (v_j < d)$, zastavíme ho a vyšleme dalšího:

j	l_j	v_j	j	l_j	v_j	j	l_j	v_j
0	35	0	0	5	1	0	30	2
1	22	8	1	16	3	1	12	6
2	17	12	2	14	4	2	31	7
3	38	14	3	22	8	3	23	15
4	7	18	4	17	12	4	25	23
5	29	26	5	38	14	5	39	25
6	19	28	6	7	18	6	21	33
7	26	36	7	29	26			
8	35	40	8	19	28			
			9	26	36			
			10	35	40			

Tabulka 3.5: Skoky divokých klokanů z Příkladu 41

6. Protože $k_6 = 21 = l_6$, můžeme pomocí $u_6 = 54$ a $v_6 = 33$ spočítat $\log_6 35 = u_6 - v_6 = 21$.

Fakt 42. *Očekávaná časová složitost algoritmu Pollardova lambda je* $O\left(\sqrt{b-a}\right)$ *(viz [6]).*

Fakt 43. *Paměťová složitost algoritmu Pollardova lambda je* $O\left(\log\left(b-a\right)\right)$*, pokud prvky* s_i *jsou mocniny dvojky (viz [6]).*

3.5 Indexový kalkulus

Na rozdíl od předchozích algoritmů nemůžeme indexový kalkulus použít v jakékoliv grupě, ale tam, kde se použít dá, má obvykle časovou složitost menší než exponenciální. Algoritmy, které jsou uvedeny v předchozích podkapitolách mají složitost exponenciální, protože složitost chápeme vzhledem k velikosti problému vyjádřené v bitech, což je $\log\left(d\right)$ nebo v případě Pollardovy lambda $\log\left(b-a\right)$. Složitost baby-step giant-step vzhledem k velikosti problému je tedy $O\left(e^{\log d/2}\right)$ a ostatní dosud uvedené složitosti můžeme přepsat obdobně.

Indexový kalkulus využívá množinu $S = \{p_1, , p_2, \ldots, p_r\}$ s prvky grupy G, které jsou vybrány tak, aby splňovaly následující:

1. pro všechna $p_i \in S$ platí $p_i \leq B$, pro nějaké „malé“ B,
2. významnou část prvků z G lze efektivně rozložit na součin prvků z množiny S.

Množina S se nazývá báze. Číslům, která se dají vyjádřit jako součin prvočísel menších než B, se říká B-hladká. Protože není známo, jak zavést B-hladká čísla v eliptických křivkách, patří eliptické křivky do kategorie grup, ve kterých indexový kalkulus použít nemůžeme. Naopak, indexový kalkulus funguje efektivně například v grupě $\mathbb{Z}_p^*$, kde p je prvočíslo.

Nechť tedy máme množinu S. Náhodně zvolíme $k \in \mathbb{Z}$, $1 \leq k \leq d-1$, a spočteme α^k. Nyní zjistíme, zda-li je α^k B-hladké. Pokud ano, platí rovnost

$$\alpha^k = p_1^{a_1} p_2^{a_2} \cdots p_r^{a_r},$$

kde $a_1, , a_2, \ldots, a_r \geq 0$. Logaritmováním obou stran poslední rovnosti získáme následující kongruenci:

$$k \equiv a_1 \log_\alpha p_1 + a_2 \log_\alpha p_2 + \cdots + a_r \log_\alpha p_r \pmod{d}.$$

Najdeme-li tedy dostatečné množství B-hladkých čísel, získáme alespoň r různých rovnic o r neznámých. Proto budeme hledat $r + c$ B-hladkých čísel, kde c je malé kladné celé číslo (například 10). Pokud se nám povedlo získat r různých rovnic o r neznámých, můžeme zjistit hodnoty logaritmů $\log_\alpha p_i$, $1 \leq i \leq r$.

Nyní volíme náhodně $k \in \mathbb{Z}$, $1 \leq k \leq d-1$, dokud nenalezneme takové, pro které je $\beta\alpha^k$ B-hladké. Pak totiž platí

$$x + k \equiv b_1 \log_\alpha p_1 + b_2 \log_\alpha p_2 + \cdots + b_r \log_\alpha p_r \pmod{d},$$

kde $b_1, , b_2, \ldots, b_r \geq 0$, a tedy můžeme x snadno dopočítat z předchozího.

Algoritmus 3.5 Indexový kalkulus

VSTUP: α, $\beta \in G$, kde G je cyklická grupa řádu d a α její generátor.
VÝSTUP: $x = \log_\alpha \beta$.

1. Najdi bázi $S = \{p_1, , p_2, \ldots, p_r\}$ takovou, aby splňovala $p_i \leq B$, $1 \leq i \leq r$, pro vhodně zvolené B a aby významnou část prvků z G bylo možné efektivně rozložit na součin prvků z této báze.
2. $j \leftarrow 0$
3. **repeat**
4. Zvol náhodně $k \in \{1, 2, \ldots, d-1\}$.
5. **if** $\alpha^k = p_1^{a_1} p_2^{a_2} \cdots p_r^{a_r}$, kde $a_1, , a_2, \ldots, a_r \geq 0$ **then**
6. $j \leftarrow j + 1$
7. $g_j \leftarrow (k, a_1, \ldots, a_r)$
8. **end if**
9. **until** $j \geq r + c$
10. Vyřeš soustavu kongruencí, která je určená posloupností g_j, a zjisti tak hodnoty $\log_\alpha p_i$ pro každé $i \in \{1, 2, \ldots, r\}$.
11. **while** true **do**
12. Zvol náhodně $k \in \{1, 2, \ldots, d-1\}$.
13. **if** $\beta\alpha^k = p_1^{b_1} p_2^{b_2} \cdots p_r^{b_r}$, kde $b_1, b_2, \ldots, b_r \geq 0$ **then**
14. $x \leftarrow (b_1 \log_\alpha p_1 + b_2 \log_\alpha p_2 + \cdots + b_r \log_\alpha p_r - k) \bmod d$
15. **return** x
16. **end if**
17. **end while**

Příklad 44. Nechť $G = \mathbb{Z}_{101}^*$, $\alpha = 2$ generuje $\mathbb{Z}_{101}^*$ a $\beta = 69$. Prvek α má tedy řád $d = 100$. Pomocí algoritmu indexový kalkulus spočítáme $\log_2 69$ následovně:

1. Zvolíme $B = 5$ a naše báze tedy bude $S = \{2, 3, 5\}$.

2. Nyní můžeme postupovat tak, že pro náhodná k aplikujeme na 2^k zkusmé dělení prvky z báze. Zjistíme tak, jaký má 2^k rozklad

nebo že není 5-hladké. Pokusíme se tak získat rozklad čtyř 5-hladkých čísel.
Pro patnáct z devatenácti náhodně vygenerovaných čísel k nebylo 2^k 5-hladké. Pro $k = 8, 5, 28, 69$ platí tyto rovnosti:

$$\begin{aligned} 2^8 \bmod 101 &= 54 = 2 \cdot 3^3, \\ 2^5 \bmod 101 &= 32 = 2^5, \\ 2^{28} \bmod 101 &= 80 = 2^4 \cdot 5, \\ 2^{69} \bmod 101 &= 3. \end{aligned}$$

Rovnosti upravíme na následující kongruence:

$$\begin{aligned} 8 &\equiv \log_2 2 + 3\log_2 3 \pmod{100}, \\ 5 &\equiv 5\log_2 2 \pmod{100}, \\ 28 &\equiv 4\log_2 2 + \log_2 5 \pmod{100}, \\ 69 &\equiv \log_2 3 \pmod{100}. \end{aligned}$$

3. Vyřešení soustavy kongruencí nám dává tyto hodnoty: $\log_2 2 = 1$, $\log_2 3 = 69$ a $\log_2 5 = 24$.

4. Pro šest náhodně vygenerovaných k nebylo číslo $\beta\alpha^k$ 5-hladké. Jako sedmé se vygenerovalo $k = 71$, a tedy jsme získali rovnost

$$\beta\alpha^k = 69 \cdot 2^{71} \bmod 101 = 20 = 2^2 \cdot 5.$$

Z rovnosti plyne kongruence

$$x + 71 \equiv 2\log_2 2 + \log_2 5 \pmod{100},$$

a tedy $\log_2 69 = (2\log_2 2 + \log_2 5 - 71) \bmod 100 = 55$.

Fakt 45. *Očekávaná časová složitost algoritmu indexový kalkulus v grupě $\mathbb{Z}_p^*$ při vhodné volbě B je $O\left(e^{(c+o(1))\sqrt{\log p \log\log p}}\right)$, kde c je kladná konstanta (viz [1]).*

4. Rozsah vlastní implementace a její výsledky

Vlastní implementace zahrnuje algoritmy baby-step giant-step, Pollardovo ró a Pohlig–Hellman. Součástí této implementace jsou dvě z grup, v kterých můžeme tyto algoritmy použít, jmenovitě multiplikativní grupa $\mathbb{Z}_n^*$ okruhu řádu n a eliptická křivka nad tělesem $\mathbb{Z}_p$, kde p je prvočíslo. Baby-step giant-step je implementován ve dvou verzích, z nichž jedna hledá diskrétní logaritmus a druhá umí určit řád prvku. Dvě verze má také Pohlig–Hellman, jedna z nich využívá na hledání diskrétního logaritmu algoritmus baby-step giant-step a druhá algoritmus Pollardovo ró.

Implementované algoritmy pro problém diskrétního logaritmu byly testovány nad grupami $\mathbb{Z}_n^*$ a eliptickou křivkou nad tělesem $\mathbb{Z}_p$. Čísla p byla v testech náhodně volena z intervalu $\left[2,\, 2^{30}\right]$, čísla n se náhodně vybírala z intervalů $\left[2,\, 2^{35}\right]$ a $\left[2,\, 2^{40}\right]$ pro všechny algoritmy kromě algoritmu Pollardovo ró, kde se n vybírala z intervalu $\left[2,\, 2^{30}\right]$ a z důvodů uvedených v podkapitole 3.2 jen prvočíselná. Úkolem pak bylo najít $\log_\alpha \beta$ pro prvky $\alpha \in G$ a $\beta \in \langle\alpha\rangle$, kde G představuje $\mathbb{Z}_n^*$ nebo $E\left(\mathbb{Z}_p\right)$. V případě Pollardova ró bylo žádoucí, aby prvek α měl prvočíselný řád. Abychom snadno získali prvek prvočíselného řádu p, byla vybírána n, která nejenže byla prvočíselná, ale navíc splňovala $n - 1 = 2^k p$ pro nějakou konstantu k. Při měření počtu grupových operací bylo zanedbáno porovnávání. Grafy vytvořené na základě výsledků těchto testů jsou uvedeny v přílohách. Ke každému algoritmu jsou poskytnuty dva grafy nad $\mathbb{Z}_n^*$ a dva nad $E\left(\mathbb{Z}_p\right)$. Pro algoritmy baby-step giant-step a Pollardovo ró grafy uvádí:

- počet grupových operací provedených během výpočtu v závislosti na velikosti vstupů – znázorněno šedými tečkami; aproximaci „šedých teček“ (neboli časové složitosti) křivkou s vyjádřením tvaru $c\sqrt{x}$ pro vstup x (vyjádření je umístěné v horní části grafu);
- počet instrukcí vykonaných procesorem během výpočtu v závislosti na velikosti vstupů – znázorněno šedými tečkami; aproximaci „šedých teček“ (neboli složitosti) křivkou s vyjádřením ve tvaru $c\sqrt{x}$ pro vstup x (vyjádření je umístěné v horní části grafu).

Časovou složitost $O\left(\sum_{i=1}^{r} e_i \left(\log d + \sqrt{p_i}\right)\right)$ algoritmu Pohlig–Hellman nemůžeme aproximovat křivkou, a proto grafy pro tento algoritmus

uvádí počet grupových operací (respektive instrukcí procesoru) provedených během výpočtu v závislosti na funkci

$$f(d) = \sum_{i=1}^{r} e_i \left(\log d + \sqrt{p_i}\right)$$

– opět znázorněno šedými tečkami. Všimněme si, že pokud je dobrou aproximací „šedých teček“ lineární funkce, testovací data potvrzují uvedenou složitost algoritmu Pohlig–Hellman.

Je vidět, že algoritmus baby-step giant-step se na rozdíl od algoritmu Pollardovo ró chová velmi „ukázněně“, především v počtu grupových operací. Vlastnosti algoritmů baby-step giant-step a Pollardovo ró se evidentně projevují i v algoritmu Pohlig-Hellman. Nicméně tento algoritmus má za použití algoritmu baby-step giant-step na použitých testovacích datech lepší výsledky než ostatní algoritmy. Dále je nutné zmínit, že algoritmus Pollardovo ró (sám i jako součást algoritmu Pohlig–Hellman) často selže pro nenáhodnost zvolené funkce (rozdělení do skupin jako v Příkladu 34). Pro zajímavost ještě následuje tabulka, která zaznamenává zaokrouhlený průměrný počet procesorových instrukcí na jednu grupovou operaci:

	$\mathbb{Z}_n^*$	$E(\mathbb{Z}_p)$
Baby-step giant-step	$13,7$	$54,5$
Pollardovo ró	$18,6$	$57,5$
Pohlig–Hellman (Baby-step giant-step)	$15,8$	$54,3$
Pohlig–Hellman (Pollardovo ró)	$17,7$	$55,1$

Tabulka 4.1: Počet instrukcí procesoru na grupovou operaci

Závěr

Vhodnost použití algoritmů pro problém diskrétního logaritmu velmi závisí na okolnostech, i když jsou tyto algoritmy považované za obecné. Některé algoritmy vyžadují velmi citlivé nastavení parametrů, aby dobře fungovaly. Zásadní otázkou také je, kolik nám naše možnosti dovolují spotřebovat paměti.

Seznam použité literatury

[1] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

[2] J. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32:918–924, 1978.

[3] Carl Pomerance. *Elementary thoughts on discrete logarithms*, pages 385–396. 2008.

[4] David Stanovský. *Základy algebry*. Matfyzpress, 2010.

[5] Douglas Stinson. *Cryptography: Theory and Practice,Second Edition*. CRC/C&H, 2nd edition, 2002.

[6] Edlyn Teske. Algorithms for finite abelian groups. `http://www.cacr.math.uwaterloo.ca/~eteske/teske/course/course_notes.html`, 1999. Lecture notes.

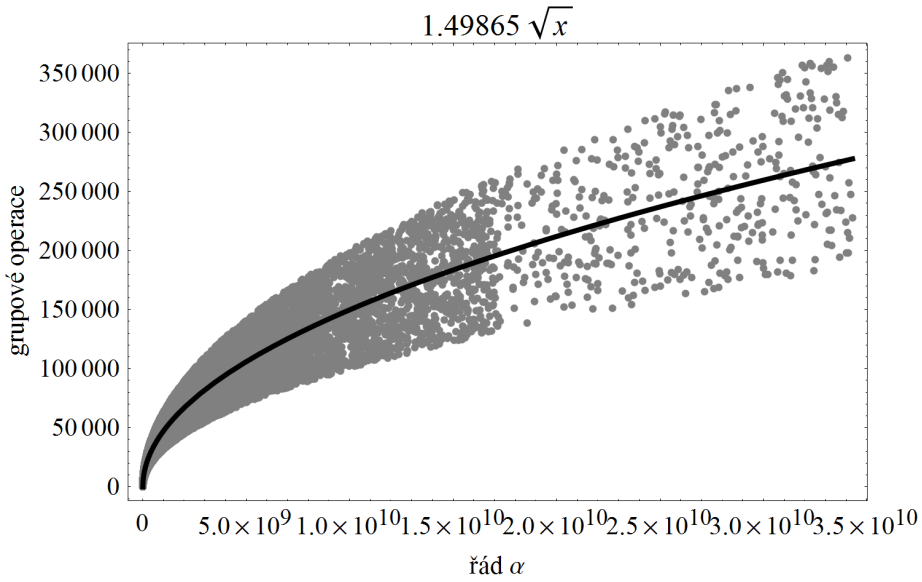
[7] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography*. Discrete Mathematics and Its Applications. Chapman & Hall/CRC, May 2003.

Seznam použitých zkratek

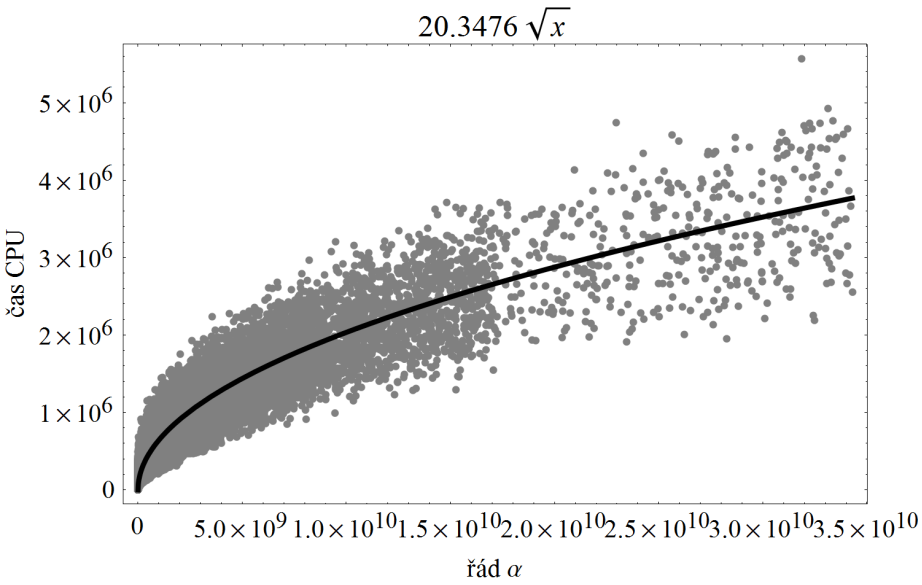
DLP problém diskrétního logaritmu

GDLP zobecněný problém diskrétního logaritmu

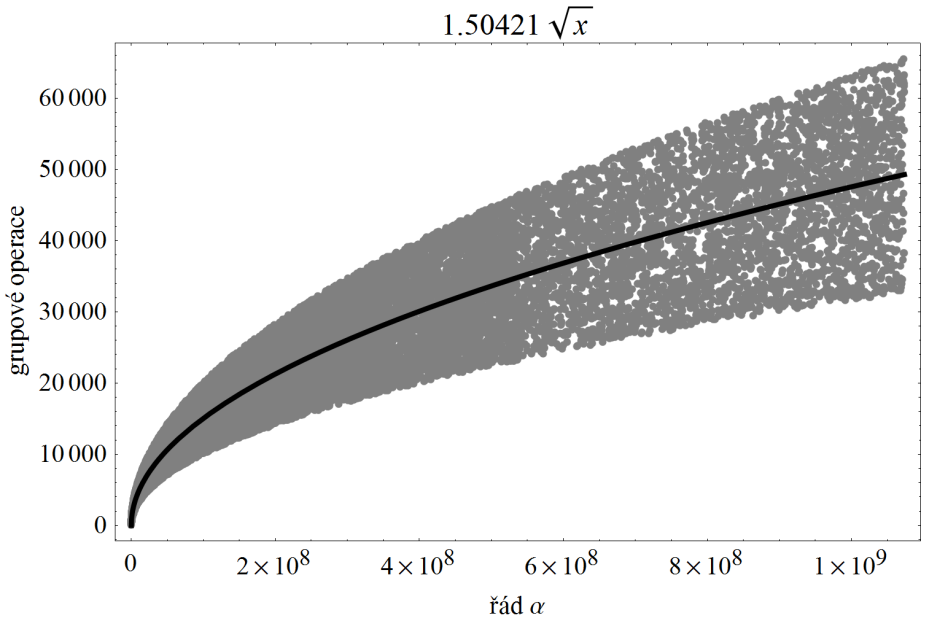
Přílohy



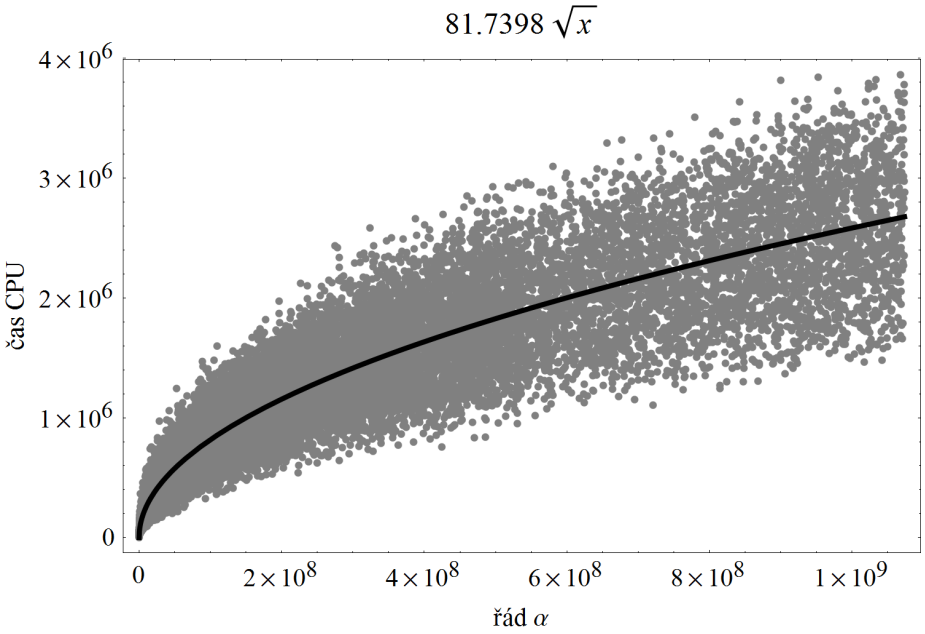
Obrázek 4.1: Baby-step giant-step nad $\mathbb{Z}_n^*$ – grupové operace



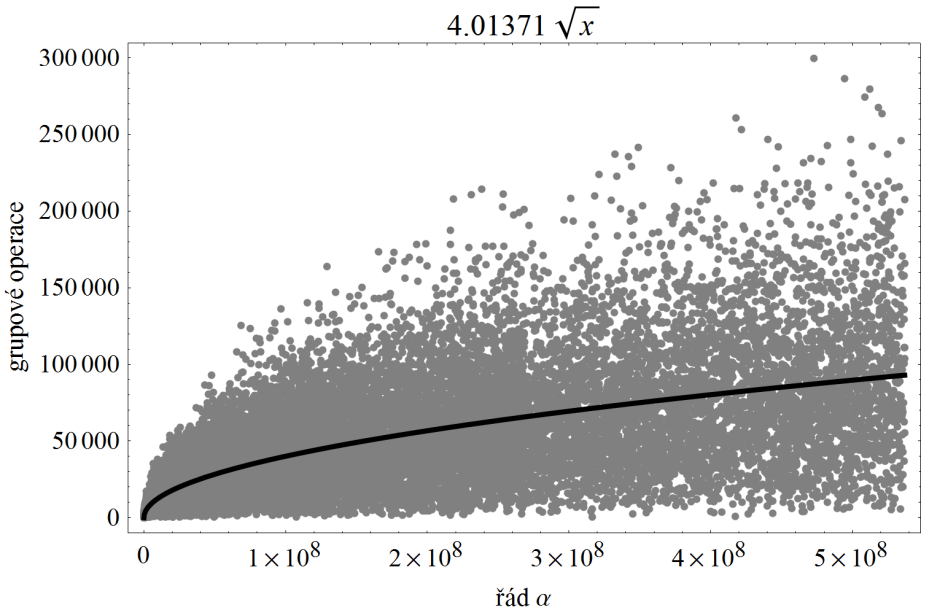
Obrázek 4.2: Baby-step giant-step nad $\mathbb{Z}_n^*$ – čas CPU



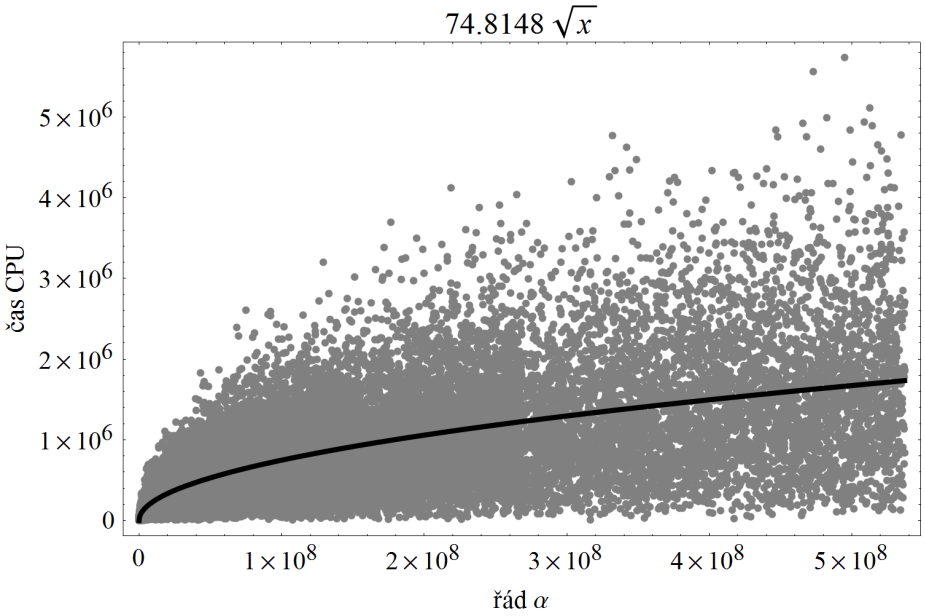
Obrázek 4.3: Baby-step giant-step nad $E(\mathbb{Z}_p)$ – grupové operace



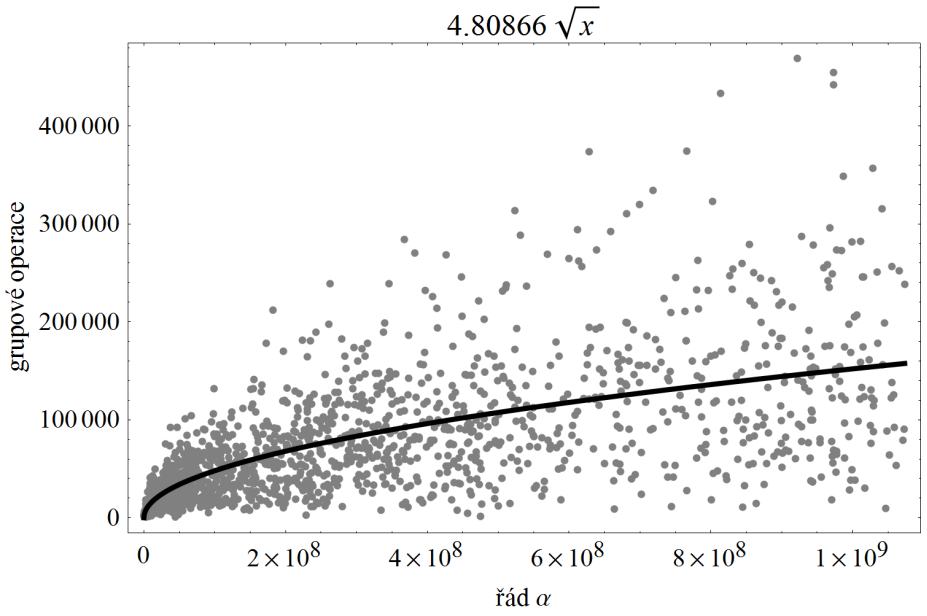
Obrázek 4.4: Baby-step giant-step nad $E(\mathbb{Z}_p)$ – čas CPU



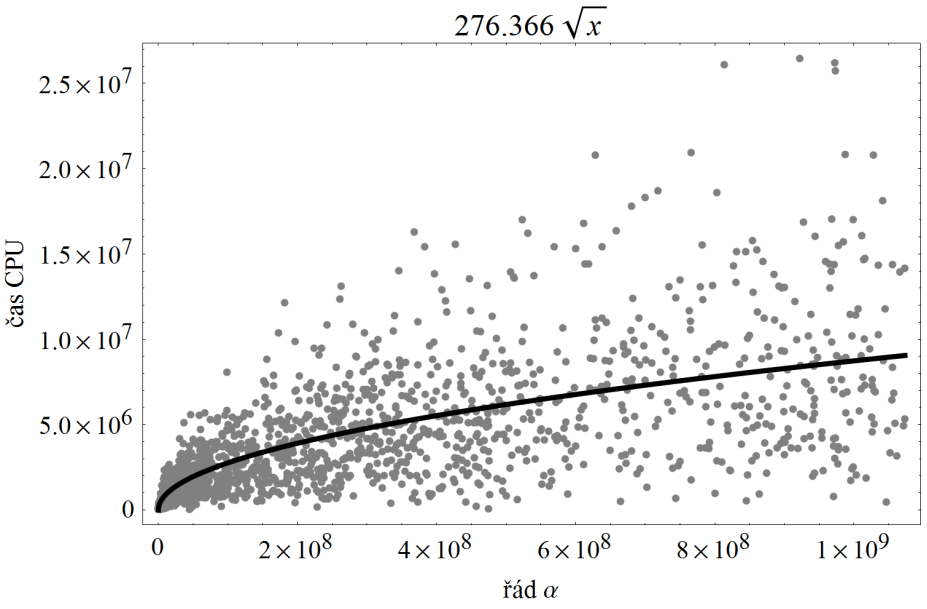
Obrázek 4.5: Pollardovo ró nad $\mathbb{Z}_n^*$ – grupové operace



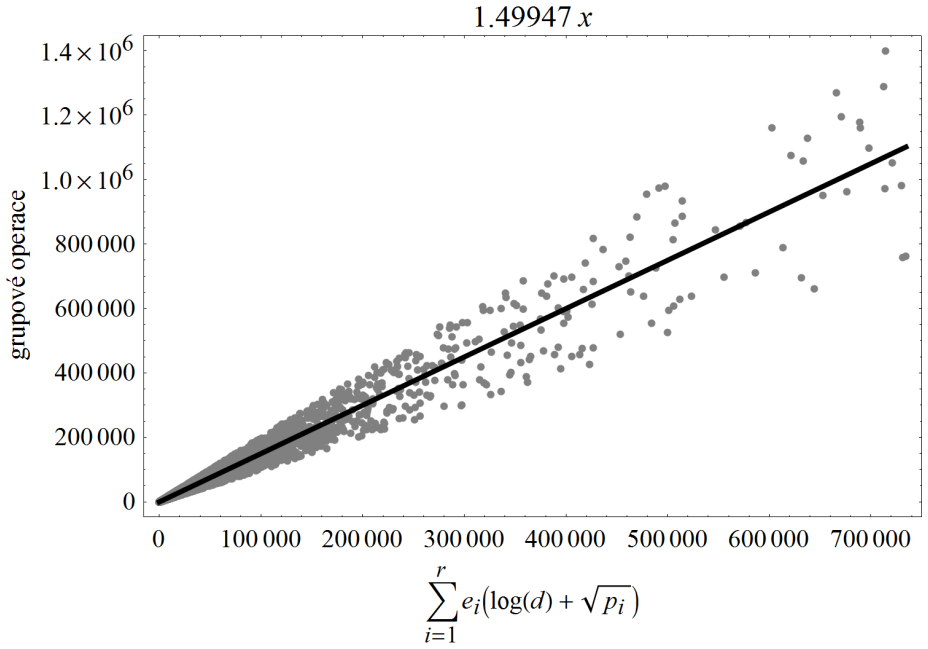
Obrázek 4.6: Pollardovo ró nad $\mathbb{Z}_n^*$ – čas CPU



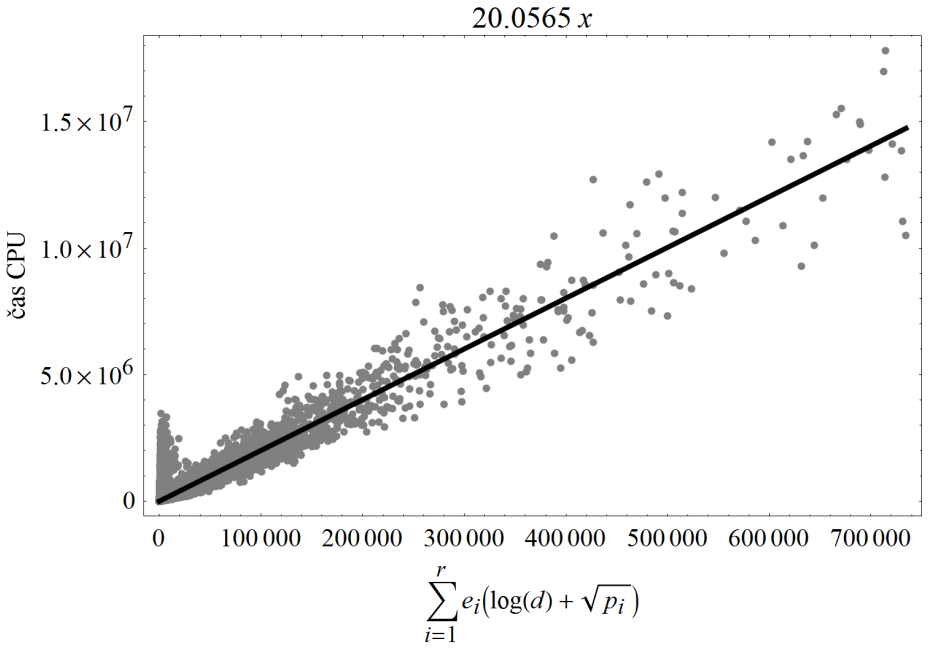
Obrázek 4.7: Pollardovo ró nad $E(\mathbb{Z}_p)$ – grupové operace



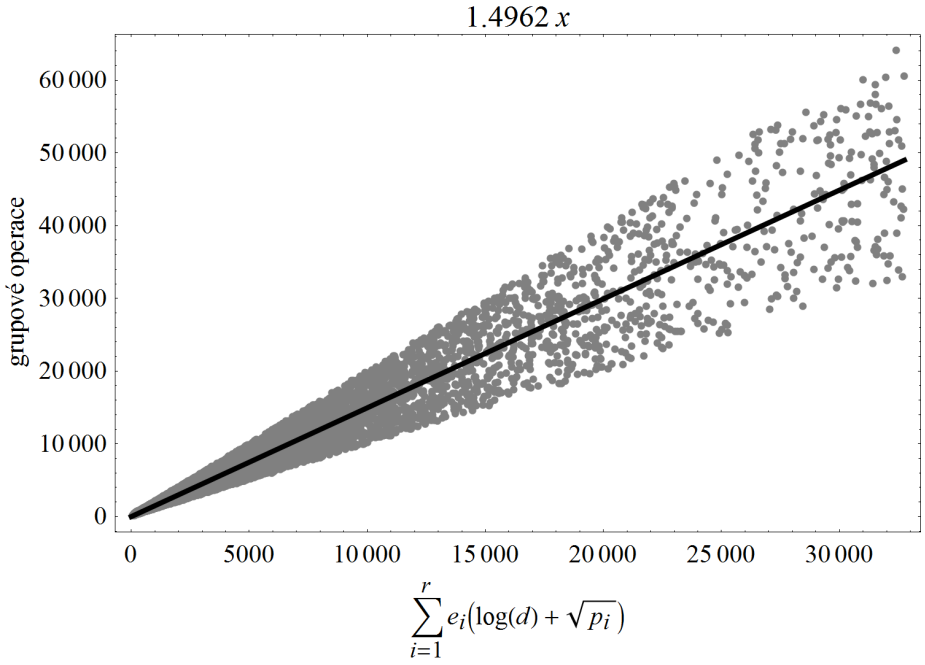
Obrázek 4.8: Pollardovo ró nad $E(\mathbb{Z}_p)$ – čas CPU



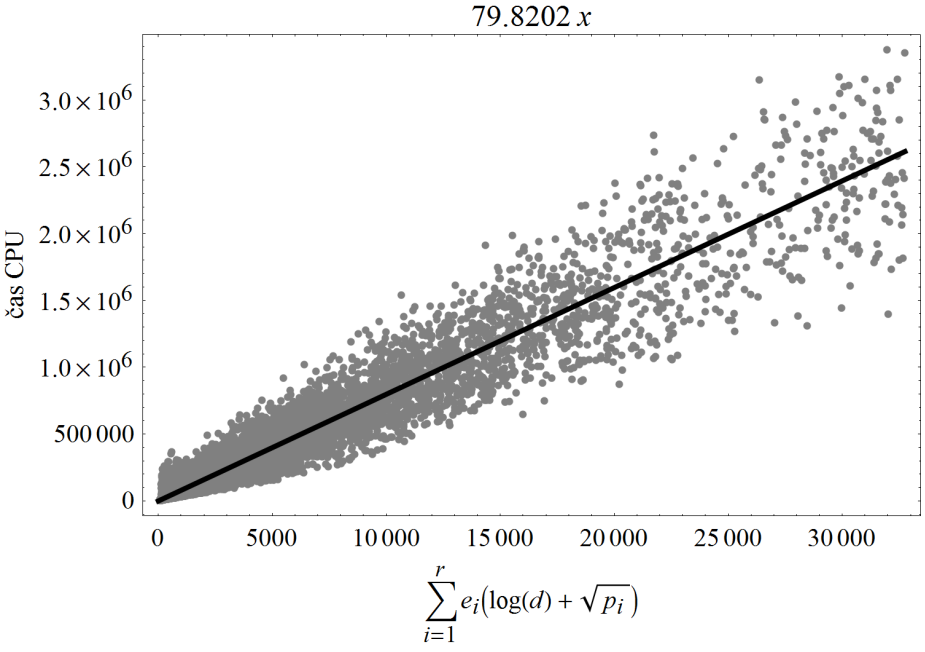
Obrázek 4.9: Pohlig–Hellman (Baby-step giant-step) nad $\mathbb{Z}_n^*$ – grupové operace



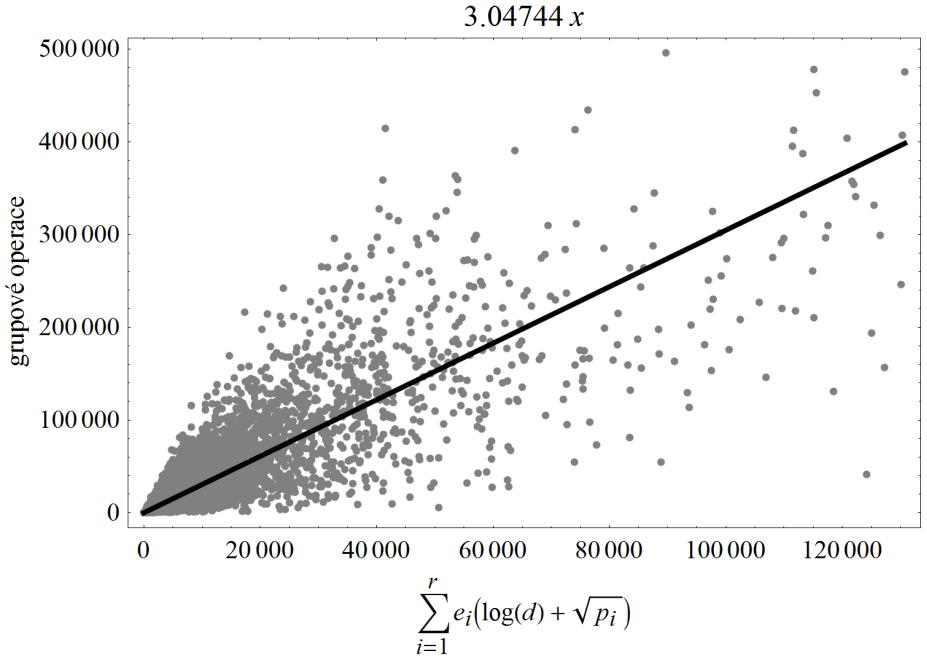
Obrázek 4.10: Pohlig–Hellman (Baby-step giant-step) nad $\mathbb{Z}_n^*$ – čas CPU



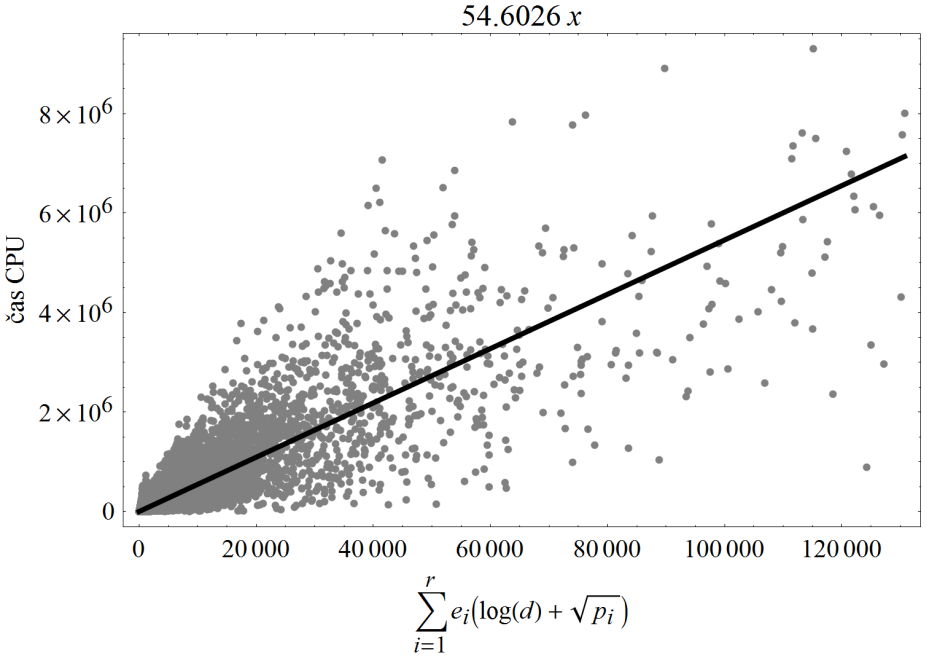
Obrázek 4.11: Pohlig–Hellman (Baby-step giant-step) nad $E(\mathbb{Z}_p)$ – grupové operace



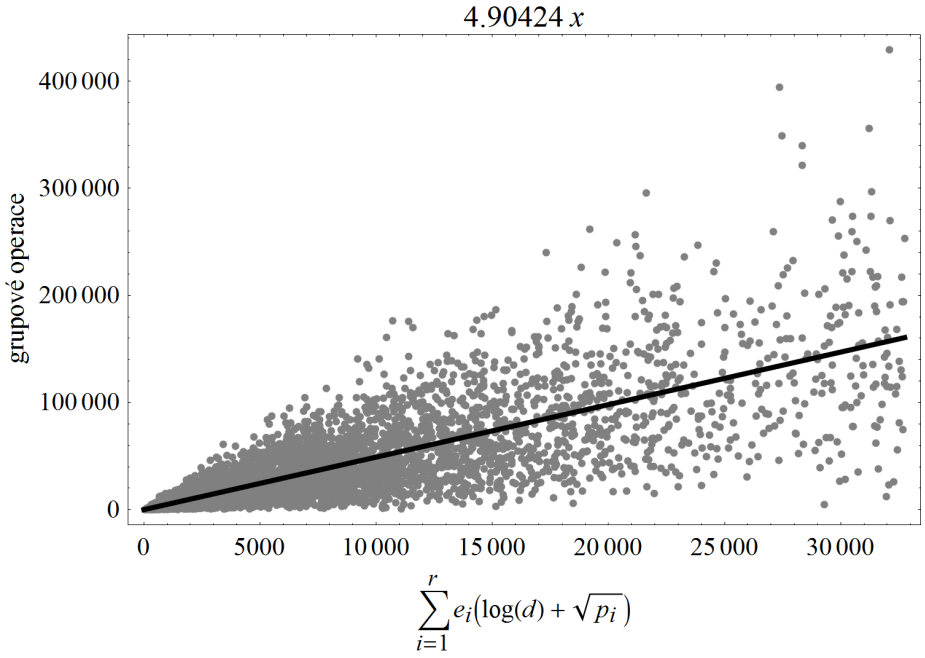
Obrázek 4.12: Pohlig–Hellman (Baby-step giant-step) nad $E(\mathbb{Z}_p)$ – čas CPU



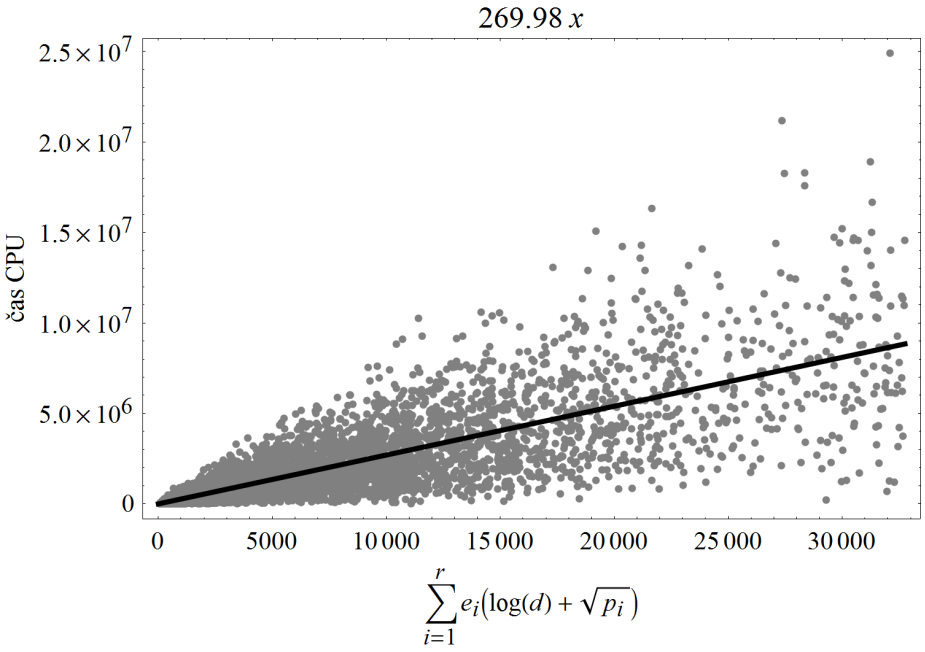
Obrázek 4.13: Pohlig–Hellman (Pollardovo ró) nad $\mathbb{Z}_n^*$ – grupové operace



Obrázek 4.14: Pohlig–Hellman (Pollardovo ró) nad $\mathbb{Z}_n^*$ – čas CPU



Obrázek 4.15: Pohlig–Hellman (Pollardovo ró) nad $E(\mathbb{Z}_p)$ – grupové operace



Obrázek 4.16: Pohlig–Hellman (Pollardovo ró) nad $E(\mathbb{Z}_p)$ – čas CPU