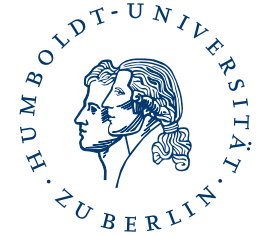
Kryptologie

Johannes Köbler



Institut für Informatik
Humboldt-Universität zu Berlin

WS 2020/21

Berechnung des diskreten Logarithmus

Zur Erinnerung:

- Sei $(G, \circ, 1)$ eine Gruppe und sei $\alpha \in G$
- Weiter bezeichne $[\alpha] = \{\alpha^i | i = 0, \ldots, n-1\}$ die von α in G erzeugte Untergruppe, wobei $n = \mathrm{ord}_G(\alpha) = \min\{e \geq 1 \mid \alpha^e = 1\}$ die Ordnung von α ist
- Dann heißt die eindeutig bestimmte Zahl $e \in \{0, \ldots, n-1\}$ mit $\beta = \alpha^e$ diskreter Logarithmus $\log_{G,\alpha}(\beta)$ von β zur Basis α in G

Das diskrete Logarithmusproblem (DLP):

Gegeben: (Beschreibung einer) Gruppe G, ein Element $\alpha \in G$ und die Ordnung $n = \mathrm{ord}_G(\alpha)$ von α in G sowie ein Element $\beta \in [\alpha]$

Gesucht: Der diskrete Logarithmus $e = \log_\alpha(\beta)$ von β zur Basis α

Berechnung des diskreten Logarithmus

- In vielen Gruppen ist die Funktion $e \mapsto \alpha^e$ effizient mittels wiederholtem Quadrieren und Multiplizieren berechenbar
- In bestimmten Fällen ist jedoch kein effizienter Algorithmus zur Berechnung der Umkehrfunktion, also von $\beta \mapsto \log_\alpha(\beta)$ bekannt, d.h. $e \mapsto \alpha^e$ ist ein Kandidat für eine Einwegfunktion

Beispiel

- Sei $G = \mathbb{Z}_p^*$, p prim, und sei α ein Erzeuger von $\mathbb{Z}_p^*$
- Dann ist $[\alpha] = \mathbb{Z}_p^*$ und α hat die Ordnung $n = p - 1$
- Ist p hinreichend groß und enthält $p - 1$ mindestens einen großen Primfaktor, so sind keine effizienten Algorithmen zur Berechnung von $\log_\alpha(\beta)$ in $\mathbb{Z}_p^*$ bekannt

Naive Berechnung des diskreten Logarithmus


```
1 γ := 1
2 for i := 0 to n − 1 do
3     if γ = β then output(i)
4     γ := αγ
```

- Dieser Algorithmus läuft in Zeit $\mathcal{O}(n)$ und benötigt nur logarithmischen Speicherplatz (wobei wir annehmen, dass elementare Gruppenoperationen in konstanter Zeit ausführbar sind)
- Falls wir im Vorfeld eine Tabelle mit den Logarithmen aller möglichen Werte für β erstellen, können wir danach für jedes β den diskreten Logarithmus durch Table-Lookup in konstanter Zeit bestimmen

DLP-Berechnung mittels Precomputation

1 *Precomputation*: Speichere die Exponenten $e = 0, \ldots, n-1$ unter der Adresse α^e in einer Tabelle T

2 *Computation*: Gib bei Eingabe β den Wert $T[\beta]$ aus

- Die Precomputation erfordert Zeit $\mathcal{O}(n)$ und Platz $\mathcal{O}(n \log n)$

Der Algorithmus von Shanks

- Der folgende Algorithmus von Shanks (auch baby-step giant-step Alg. genannt) berechnet ebenfalls im Vorfeld eine Tabelle von DLP-Werten
- Allerdings nur für die Potenzen α^{jm}, $j = 0, \ldots, m-1$ und $m = \lceil \sqrt{n} \rceil$
- Dadurch erhöht sich zwar die Laufzeit zur Bestimmung des diskreten Logarithmus für β von $O(1)$ auf $O(\sqrt{n})$
- Dafür wird der Speicherplatz von $\mathcal{O}(n \log n)$ auf $\mathcal{O}(\sqrt{n} \log n)$ reduziert

Algorithmus `Shanks`$(G, n, \alpha, \beta, m = \lceil \sqrt{n} \rceil)$

1 *Precomputation*: Sortiere die Paare (α^{im}, i), $i = 0, \ldots, m-1$, nach der ersten Komponente in eine Tabelle $T1$

2 *Computation*: Sortiere bei Eingabe β die Paare $(\beta\alpha^{-j}, j)$, $j = 0, \ldots, m-1$, nach der ersten Komponente in eine Tabelle $T2$

3 ermittle durch parallele sequentielle Suche Paare (γ, i) in $T1$ und (γ, j) in $T2$ mit derselben ersten Komponente

4 **output** $im + j$ // *es gilt* $\beta\alpha^{-j} = \gamma = \alpha^{im}$

Der chinesische Restsatz

Satz (Chinesischer Restsatz)

Falls $n_1, \ldots, n_k$ paarweise teilerfremd sind, dann hat das System

$$\begin{array}{lll} x & \equiv_{n_1} & a_1 \\ & \vdots & \\ x & \equiv_{n_k} & a_k \end{array} \qquad (*)$$

für beliebige Zahlen $a_1, \ldots, a_k \in \mathbb{Z}$ genau eine Lösung modulo $n = \prod_{i=1}^{k} n_i$

Beweis.

- Zu jeder Zahl $b_i = {}^{n}/_{n_i}$ ex. wegen $\mathrm{ggT}(b_i, n_i) = 1$ Zahlen c_i und d_i mit
$$c_i b_i + d_i n_i = \mathrm{ggT}(b_i, n_i) = 1$$
- Diese sind mit dem erweiterten euklidschen Algorithmus effizient berechenbar und es gilt $c_i \equiv b_i^{-1} \bmod n_i$
- Für $j = 1, \ldots, k$ löst daher die Zahl $c_j b_j \bmod n_j$ das System

$$x \equiv_{n_i} \begin{cases} 0, & i \neq j \qquad (1) \\ 1, & i = j \qquad (2) \end{cases}$$

Beweis des chinesischen Restsatzes

Beweis (Fortsetzung).

- Folglich erfüllt $a = \sum_{j=1}^{k} a_j b_j c_j \bmod n$ für $i = 1, \ldots, k$ die Kongruenz

  $$a \overset{(1)}{\equiv}_{n_i} a_i b_i c_i \overset{(2)}{\equiv}_{n_i} a_i,$$

  d.h. a löst das System $(*)$
- Dies zeigt, dass die Funktion

  $$f : \mathbb{Z}_n \to \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_k} \text{ mit } f(x) = (x \bmod n_1, \ldots, x \bmod n_k)$$

  surjektiv ist
- Da der Definitions- und der Wertebereich von f gleich groß sind, muss f auch injektiv sein und $(*)$ ist eindeutig lösbar □

Der Pohlig-Hellman-Algorithmus

- Mit dem Pohlig-Hellman-Algorithmus lässt sich der diskrete Logarithmus in einer beliebigen Gruppe G berechnen
- Die Ordnung der Potenz α^i eines Elements $\alpha \in G$ der Ordnung n ist $\mathrm{ord}_G(\alpha^i) = n/\,\mathrm{ggT}(n, i)$
- Ist insbesondere q ein Teiler von n, so hat $\alpha^{n/q}$ die Ordnung q
- Sei $a = \log_{G,\alpha}(\beta)$ der diskrete Logarithmus von β zur Basis α
- Weiter sei $n = \prod_{i=1}^{k} p_i^{e_i}$ die Primfaktorzerlegung der Ordnung n von α
- Falls wir für $i = 1, \ldots, k$ die Werte $a_i = a \bmod p_i^{e_i}$ kennen, so lässt sich daraus a leicht mit dem Chinesischen Restsatz berechnen
- Schreiben wir a_i als Zahl zur Basis p_i, so erhalten wir Ziffern $d_0, \ldots, d_{e_i-1} \in \mathbb{Z}_{p_i}$ mit

$$a_i = \sum_{j=0}^{e_i-1} d_j p_i^j \quad \text{bzw.} \quad a_i = (d_{e_i-1} \cdots d_0)_{p_i}$$

- Weiter existieren für $i = 1, \ldots, k$ Zahlen $s_i \geq 0$ mit $a = a_i + s_i p_i^{e_i}$

Der Pohlig-Hellman-Algorithmus

- Um nun die Ziffer d_0 zu berechnen, betrachten wir die Gleichung

$$\beta^{n/p_i} = \alpha^{d_0 n/p_i} \quad \text{bzw.} \quad d_0 = \log_{G,\alpha^{n/p_i}}(\beta^{n/p_i})$$

- Diese lässt sich leicht verifizieren:

$$\begin{aligned}
\beta^{n/p_i} &= (\alpha^a)^{n/p_i} = (\alpha^{d_0 + d_1 p_i + \cdots + d_{e_i-1} p_i^{e_i-1} + s_i p_i^{e_i}})^{n/p_i} \\
&= (\alpha^{d_0 + t p_i})^{n/p_i} \text{ für eine Zahl } t \geq 0 \\
&= \alpha^{d_0 n/p_i} \alpha^{tn} = \alpha^{d_0 n/p_i}
\end{aligned}$$

- Die obigen Gleichungen lassen sich verallgemeinern, indem wir $\beta_0 = \beta$ setzen und für $j = 1, \ldots, e_i - 1$ die folgenden Potenzen berechnen:

$$\begin{aligned}
\beta_j &= \beta\alpha^{-d_0 - d_1 p_i - d_2 p_i^2 \cdots - d_{j-1} p_i^{j-1}} \\
&= \alpha^{a - d_0 - d_1 p_i - d_2 p_i^2 \cdots - d_{j-1} p_i^{j-1}} \\
&= \alpha^{d_j p_i^j + d_{j+1} p_i^{j+1} + \cdots + d_{e_i-1} p_i^{e_i-1} + s_i p_i^{e_i}} \\
&= \alpha^{d_j p_i^j + t p_i^{j+1}} \text{ für eine Zahl } t \geq 0
\end{aligned}$$

Der Pohlig-Hellman-Algorithmus

- Um nun alle Ziffern $d_0, \ldots, d_{e_i-1}$ zu berechnen, betrachten wir für $j = 0, \ldots, e_i - 1$ und $\beta_j = \beta\alpha^{-d_0 - d_1 p_i - d_2 p_i^2 \cdots - d_{j-1} p_i^{j-1}}$ die Gleichung

$$\beta_j^{n/p_i^{j+1}} = \alpha^{d_j n/p_i} \quad \text{bzw.} \quad d_j = \log_{G,\alpha^{n/p_i}}(\beta_j^{n/p_i^{j+1}})$$

- Diese lässt sich ebenfalls leicht verifizieren:

$$\begin{aligned}
\beta_j^{n/p_i^{j+1}} &= (\alpha^{a - d_0 - d_1 p_i - d_2 p_i^2 \cdots - d_{j-1} p_i^{j-1}})^{n/p_i^{j+1}} \\
&= (\alpha^{d_j p_i^j + d_{j+1} p_i^{j+1} + \cdots + d_{e_i-1} p_i^{e_i-1} + s_i p_i^{e_i}})^{n/p_i^{j+1}} \\
&= (\alpha^{d_j p_i^j + t p_i^{j+1}})^{n/p_i^{j+1}} \text{ für eine Zahl } t \geq 0 \\
&= \alpha^{d_j n/p_i} \alpha^{tn} \\
&= \alpha^{d_j n/p_i}
\end{aligned}$$

- Der folgende Algorithmus berechnet sukzessive die Zahlen β_j und dazu die Ziffern $d_j = \log_{G,\alpha^{n/p_i}}(\beta_j^{n/p_i^{j+1}})$, die sich wegen $\text{ord}_G(\alpha^{n/p_i}) = p_i$ in Zeit $\mathcal{O}(\sqrt{p_i})$ (etwa mit dem Algorithmus von Shanks) ermitteln lassen

Der Pohlig-Hellman-Algorithmus

Algorithmus `Pohlig-Hellman`$(G, n, \alpha, \beta, p_1, \ldots, p_k, e_1, \ldots, e_k)$

1 $\beta_0 := \beta$
2 **for** $i := 1$ **to** k **do**
3 **for** $j := 0$ **to** $e_i - 1$ **do**
4 $d_j := \log_{G,\alpha^{n/p_i}}(\beta_j^{n/p_i^{j+1}})$ // *z.B. mit Shanks*
5 $\beta_{j+1} := \beta_j \alpha^{-d_j p_i^j}$
6 $a_i := (d_{e_i-1} \cdots d_0)_{p_i}$
7 $n_i := p_i^{e_i}$
8 $b_i := n/n_i$
9 $c_i := b_i^{-1} \bmod n_i$
10 **output** $a = \sum_{i=1}^{k} a_i b_i c_i \bmod n$

- Somit erhalten wir eine Laufzeit von $\mathcal{O}(\sqrt{p_i} e_i)$ zur Bestimmung von a_i in den Zeilen 3 bis 5
- Dies führt auf eine Gesamtlaufzeit von $\mathcal{O}(\sqrt{\max_i p_i} \log n)$ zur Bestimmung von a

Der Pohlig-Hellman-Algorithmus

Beispiel

- Sei $G = \mathbb{Z}_m^*$ mit $m = 29^3 = 24389$
- Zudem sei $\alpha = 3$ und $\beta = 3344$
- Da wir die Ordnung von α in G nicht kennen, setzen wir $n = \|G\| = \varphi(29^3) = (29-1)29^2 = 23548 = 2^2 \cdot 7 \cdot 29^2$
- Der Algorithmus von Pohlig und Hellman berechnet nun die folgenden Werte für $a_i = a \bmod p_i^{e_i}$:

i	p_i	e_i	$n_i = p_i^{e_i}$	n/p_i	α^{n/p_i}	j	β_j	n/p_i^{j+1}	$\beta_j^{n/p_i^{j+1}}$	d_j	a_i
1	2	2	4	11774	24388	0	3344	11774	1	0	
						1	3344	5887	24388	1	2
2	7	1	7	3364	7302	0	3344	3364	4850	2	2
3	29	2	841	812	12616	0	3344	812	11775	28	
						1	6998	28	3365	8	260

Der Pohlig-Hellman-Algorithmus

Beispiel (Fortsetzung)

- Da für alle Primteiler p_i von n die Potenz $\alpha^{n/p_i} \neq 1$ ist, ist α ein Erzeuger von G und somit $\mathrm{ord}_G(\alpha) = n$
- Der gesuchte diskrete Logarithmus $a = \log_{G,\alpha}(\beta)$ muss nun noch mit dem Chinesischen Restsatz als Lösung der drei Kongruenzen $a \equiv_{n_i} a_i$ bestimmt werden:

i	a_i	$n_i = p_i^{e_i}$	$b_i = n/n_i$	$c_i = b_i^{-1} \bmod n_i$	$a_i b_i c_i \bmod n$
1	2	$4 = 2^2$	5887	3	11774
2	2	$7 = 7^1$	3364	2	13456
3	260	$841 = 29^2$	28	811	17080
$\sum$					18762

- Damit ist $a = 18762$ der diskrete Logarithmus $\log_{G,3}(3344)$ von $\beta = 3344$ in $G = \mathbb{Z}_m^*$ mit $m = 24389$ ◁

Der Rho-Faktorisierungsalgorithmus

- Von Pollard wurde eine heuristische Strategie entwickelt, die sich sowohl zur Lösung des DLP als auch des Faktorisierungsproblems eignet
- Wir betrachten zunächst die Faktorisierungsvariante des Rho-Algorithmus von Pollard
- Sei n eine Zahl mit mindestens zwei verschiedenen Primteilern $p < q$
- Falls n nur einen Primteiler hat, also eine Primzahlpotenz ist, lässt sich n leicht durch Berechnung der k-ten Wurzeln für $k = 2, \ldots, \log_2(n)$ faktorisieren
- Angenommen, wir wählen zufällig eine Menge $X \subseteq \mathbb{Z}_n$ der Größe $1{,}17\sqrt{p}$, so enthält X aufgrund des Geburtstagsparadoxons mit großer Wahrscheinlichkeit zwei Elemente $x \neq x'$ mit $x \equiv_p x'$
- Wegen $x \equiv_p x'$, aber $x \not\equiv_n x'$ führen diese auf einen nichttrivialen Faktor $d = \mathrm{ggT}(x - x', n)$ mit $p \leq d < n$ von n

Der Rho-Faktorisierungsalgorithmus

- Anstelle einer zufälligen Menge können wir auch eine pseudozufällige Menge der Form $X = \{x_1, x_2 = f(x_1), x_3 = f(x_2), \dots\}$ wählen
- Dabei ist $x_1 \in_R \mathbb{Z}_n$ ein zufällig gewählter Startwert
- Dann tritt bei geeigneter Wahl von $f : \mathbb{Z}_n \to \mathbb{Z}_n$ mit großer Wahrscheinlichkeit eine Kollision $x_j \equiv_p x_i$ für ein $i < j \leq 1{,}17\sqrt{p}$ auf
- Eine gute Wahl für f ist beispielsweise $f(x) = x^2 \pm 1 \bmod n$
- Werden zur Berechnung von f nur die Ringoperationen von $\mathbb{Z}_n$ verwendet, so ist $f(x)$ ein Polynom
- Wegen $x_i^k - x_j^k = (x_i - x_j)(x_i^{k-1} + x_j x_i^{k-2} + \cdots + x_i x_j^{k-2} + x_j^{k-1})$ erhalten wir daher die Implikation

$$x_i \equiv_p x_j \Rightarrow x_{i+1} = f(x_i) \equiv_p f(x_j) = x_{j+1}$$

- Dies impliziert wiederum für für alle $k \geq 0$ die Kongruenz $x_{i+k} \equiv_p x_{j+k}$ bzw. für alle $k \geq i$ die Kongruenz $x_k \equiv_p x_{j-i+k}$

Der Rho-Faktorisierungsalgorithmus

- Setzen wir $r = j - i$, so erhalten wir für alle $k \geq i$ die Kongruenz $x_k \equiv_p x_{k+r}$ und daraus $x_k \equiv_p x_{k+dr}$ für alle $k \geq i$ und $d \geq 0$
- Insbesondere folgt also $x_k \equiv_p x_{2k}$ für alle $k \geq i$ mit $k \equiv_r 0$
- Das kleinste solche k ist sicherlich kleiner als $i + r = j$
- Indem wir also sukzessive die Paare (x_k, y_k) mit $y_k = x_{2k}$ berechnen, können wir mit sehr geringem Platzverbrauch ein Kollisionspaar (x_k, y_k) mit $x_k \equiv_p y_k$ und $k < i + r = j$ finden (ohne p zu kennen):

Algorithmus `Pollard-Rho-Factorize`(n)

1 wähle zufällig $x \in \mathbb{Z}_n$
2 $y := x^2 + 1 \bmod n$
3 **while** $\mathrm{ggT}(x - y, n) = 1$ **do**
4 $\quad x := f(x)$
5 $\quad y := f(f(y))$
6 **if** $d = \mathrm{ggT}(x - y, n) < n$ **then output**(d)
7 **else output**$(?)$

Der Rho-DLP-Algorithmus

- Der Rho-DLP-Algorithmus berechnet eine pseudozufällige Folge von Paaren $(c_i, d_i) \in \mathbb{Z}_n \times \mathbb{Z}_n$
- Ziel ist es, zwei verschiedene Paare (c_i, d_i) und (c_j, d_j) mit $\alpha^{c_i}\beta^{d_i} = \alpha^{c_j}\beta^{d_j}$ zu finden
- Im Fall $\mathrm{ggT}(d_j - d_i, n) = 1$ lässt sich hieraus wegen

  $$\alpha^{c_i + a d_i} = \alpha^{c_i}\beta^{d_i} = \alpha^{c_j}\beta^{d_j} = \alpha^{c_j + a d_j}$$

  der diskrete Logarithmus $\log_{G,\alpha}(\beta) = (c_i - c_j)(d_j - d_i)^{-1} \bmod n$ leicht bestimmen
- Andernfalls erhalten wir $g = \mathrm{ggT}(d_j - d_i, n)$ Kandidaten $a_1, \ldots, a_g$, unter denen der richtige ebenfalls leicht zu ermitteln ist, sofern g nicht zu groß ist
- Zur Bildung der Pseudozufallsfolge kann bspw. die Funktion f in folgendem Algorithmus benutzt werden
- Aus Effizienzgründen berechnet sie auch gleich die Werte $x_i = \alpha^{c_i}\beta^{d_i}$

Der Rho-DLP-Algorithmus

function $f(x, c, d)$

1 case
2 $\quad x \in S_1$: **return**$(\beta x, c, d + 1 \bmod n)$
3 $\quad x \in S_2$: **return**$(x^2, 2c \bmod n, 2d \bmod n)$
4 $\quad x \in S_3$: **return**$(\alpha x, c + 1 \bmod n, d)$

Algorithmus `Pollard-Rho-DLP`(G, n, α, β)

1 wähle zufällig $c, d \in \mathbb{Z}_n$
2 $x := \alpha^c \beta^d$
3 $(y, e, f) := f(x, c, d)$
4 **while** $x \neq y$ **do**
5 $\quad (x, c, d) := f(x, c, d)$
6 $\quad (y, e, f) := f(f(y, e, f))$
7 $g := \mathrm{ggT}(f - d, n)$
8 bestimme alle Lösungen $a_1, \ldots, a_g$ von $(f - d)a \equiv_n (c - e)$
9 **output** a_i mit $\alpha^{a_i} = \beta$

Der Rho-DLP-Algorithmus

- Die Mengen S_1, S_2, S_3 bilden eine Partition von G in drei etwa gleich große Mengen, wobei das neutrale Element 1 von G nicht in S_2 enthalten sein sollte
- Ähnlich wie beim Rho-Faktorisierungsalgorithmus lässt sich argumentieren, dass die while-Schleife nach ca. $1{,}17\sqrt{n}$ Iterationen abbricht

Die Index-Calculus-Methode

- Bei der Index-Calculus-Methode handelt es sich nicht um einen generischen DLP-Algorithmus, da sie nur für spezielle Gruppen anwendbar ist
- Wir betrachten den wichtigen Spezialfall $G = \mathbb{Z}_p^*$, p prim, und $\text{ord}(\alpha) = p - 1$
- Der Algorithmus benutzt eine Faktorbasis $B = \{p_1, \ldots, p_b\}$, wobei wir annehmen, dass B die ersten b Primzahlen enthält

Algorithmus `Index-Calculus`(p, α, β, b)

Precomputation: Bestimme $\ell_i = \log_\alpha p_i$ für $i = 1, \ldots, b$
Computation:
  wähle zufällig eine Zahl $s \in \{0, \ldots, p-2\}$
  $\gamma := \beta\alpha^s \bmod p$
  if γ ist über B faktorisierbar **then**
    berechne Exponenten $c_1, \ldots, c_b$ mit $\gamma = p_1^{c_1} \cdots p_b^{c_b}$
    output $(c_1\ell_1 + \cdots + c_b\ell_b - s \bmod p - 1)$

Die Index-Calculus-Methode

- Zur Bestimmung der Zahlen ℓ_i kann man wie folgt vorgehen
- Wähle c etwas größer als b (z.B. $c = b + 10$) und generiere c Kongruenzen der Form

$$\alpha^{x_j} \equiv_p p_1^{a_{1j}} \cdots p_b^{a_{bj}},\ j = 1, \ldots, c$$

- Hierzu kann man x_j zufällig wählen und testen, ob $y_j = \alpha^{x_j} \bmod p$ über B faktorisierbar ist
- Die Wahrscheinlichkeit hierfür hängt natürlich von der Größe von B ab
- Aus den Kongruenzen lässt sich ein lineares Kongruenzgleichungssystem der Form

$$\underbrace{\begin{pmatrix} a_{11} & \cdots & a_{b1} \\ & \ddots & \\ a_{1c} & \cdots & a_{bc} \end{pmatrix}}_{A} \begin{pmatrix} \ell_1 \\ \vdots \\ \ell_b \end{pmatrix} \equiv_{p-1} \begin{pmatrix} x_1 \\ \vdots \\ x_c \end{pmatrix}$$

für die Unbekannten $\ell_1, \ldots, \ell_b$ gewinnen

Die Index-Calculus-Methode

- Diese liefert die gewünschten Werte, falls A durch Streichen von $c - b$ Zeilen in eine $(b \times b)$-Matrix A' mit $\mathrm{ggT}(\det A', p-1) = 1$ transformiert werden kann
- Durch eine heuristische Komplexitätsanalyse lässt sich zeigen, dass
  - die Precomputation-Phase in Zeit $\mathcal{O}(e^{(1+o(1))\sqrt{\ln p \ln\ln p}})$ und
  - die Computation-Phase in Zeit $\mathcal{O}(e^{(1/2+o(1))\sqrt{\ln p \ln\ln p}})$

  ausführbar ist

Die Index-Calculus-Methode

Beispiel

- Sei $p = 10007$ und $\alpha = 5$
- Als Faktorbasis B wählen wir $B = \{2, 3, 5, 7\}$
- Zudem wählen wir $x_1 = 4063$, $x_2 = 5136$ und $x_3 = 9865$
- Damit erhalten wir wegen

$$5^{4063} \bmod p = 42 = 2^1 3^1 7^1$$
$$5^{5136} \bmod p = 54 = 2^1 3^3 7^0$$
$$5^{9865} \bmod p = 189 = 2^0 3^3 7^1$$

das Kongruenzgleichungssystem

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 3 & 0 \\ 0 & 3 & 1 \end{pmatrix} \begin{pmatrix} \ell_1 \\ \ell_2 \\ \ell_4 \end{pmatrix} \equiv_{p-1} \begin{pmatrix} 4063 \\ 5136 \\ 9865 \end{pmatrix}$$

für die Unbekannten ℓ_1, ℓ_2, ℓ_4

Die Index-Calculus-Methode

Beispiel (Fortsetzung)

- Subtrahieren wir die erste Zeile von der Summe der 2. und 3. Zeile, so erhalten wir die Gleichung

$$5\ell_2 \equiv_{p-1} 5136 + 9865 - 4063 = 10938 \equiv_{p-1} 932$$

und somit $\ell_2 = 6190$

- Zudem ist

$$\ell_1 = 5136 - 3\ell_2 \bmod p - 1 = 6578,$$
$$\ell_4 = 9865 - 3\ell_2 \bmod p - 1 = 1301$$

und

$$\ell_3 = \log_\alpha p_3 = \log_5 5 = 1$$

- Wollen wir nun den diskreten Logarithmus für $\beta = 9451$ bestimmen, so wählen wir eine Zufallszahl s (z.B. $s = 7736$) und berechnen

$$\gamma = \beta\alpha^s = 9451 \cdot 5^{7736} \equiv_p 8400 = 2^4 3^1 5^2 7^1$$

- Daraus erhalten wir $\log_\alpha \beta = 4\ell_1 + \ell_2 + 2\ell_3 + \ell_4 - s \bmod p - 1 = 4 \cdot 6578 + 6190 + 2 \cdot 1 + 1301 - 7736 \bmod 10006 = 6057$ ◁

Die Methode der zufälligen Quadrate

- Mit einer ähnlichen Methode lässt sich übrigens auch eine zusammengestzte Zahl n faktorisieren (so genannte Methode der zufälligen Quadrate)
- Hierzu sucht man nach Zahlen $x_i \in \mathbb{Z}_n^*$, $i \in I$, mit der Eigenschaft, dass $y_i = x_i^2 \bmod n$ über $B = \{p_1, \ldots, p_b\}$ faktorisierbar ist: $y_i = p_1^{c_{i1}} \cdots p_b^{c_{ib}}$
- Danach bestimmt man eine Teilmenge $J \subseteq I$, so dass die Primfaktorzerlegung $p_1^{e_1} \cdots p_b^{e_b}$ von $y = \prod_{i \in J} y_i$ nur gerade Exponenten $e_j = \sum_{i \in J} c_{ij}$ hat
- Setzen wir nun $a = \prod_{i \in J} x_i \bmod n$ und $b = p_1^{e_1/2} \cdots p_b^{e_b/2} \bmod n$, so gilt offenbar $a^2 \equiv_n y \equiv_n b^2$
- Dies impliziert $n|(a^2 - b^2)$, also $n|(a+b)(a-b)$
- Daher können wir im Fall $a \not\equiv_n \pm b$ (d.h. n teilt weder $a+b$ noch $a-b$) einen nichttrivialen Faktor $\mathrm{ggT}(a-b, n)$ von n bestimmen

Die Methode der zufälligen Quadrate

Beispiel

- Sei $n = 15770708441$ und $B = \{2, 3, 5, 7, 11, 13\}$
- Wählen wir $x_1 = 8340934156$, $x_2 = 12044942944$ und $x_3 = 2773700011$, so erhalten wir folgende über B faktorisierbaren quadratischen Reste:

$$y_1 = x_1^2 \bmod n = 21 = 3^1 7^1$$
$$y_2 = x_2^2 \bmod n = 182 = 2^1 7^1 13^1$$
$$y_3 = x_3^2 \bmod n = 78 = 2^1 3^1 13^1$$

- Für $J = \{1, 2, 3\}$ erhalten wir das Produkt $x_1^2 x_2^2 x_3^2 \equiv_n 2^2 3^2 7^2 13^2$
- Dieses führt auf die beiden Quadrate $a^2 \equiv_n b^2$ mit
  - $a = x_1 x_2 x_3 \bmod n = 9503435785$ und
  - $b = 2^{2/2} 3^{2/2} 7^{2/2} 13^{2/2} \bmod n = 2 \cdot 3 \cdot 7 \cdot 13 \bmod n = 546$
- Die Zahlen $a = 9503435785$ und $b = 546$ führen wegen $a \not\equiv_n \pm b$ auf den Faktor $\mathrm{ggT}(a - b, n) = 115759$ von n ◁

Die Methode der zufälligen Quadrate

- Die Zahlen x_i können hierbei entweder zufällig aus $\mathbb{Z}_n$ gewählt werden, oder besser von der Form $x_i = \lceil \sqrt{kn} \rceil + l$ für kleine Zahlen $k, l \geq 0$
- Da dies bewirkt, dass $y_i = x_i^2 \bmod n$ relativ klein ist, erhöht dies die Wahrscheinlichkeit, dass y_i über B faktorisierbar ist
- Wir können auch testen, ob die Zahl $y_i' = n - y_i$ über B faktorisierbar ist, da sich in diesem Fall y_i in ein Produkt $y_i = (-1)^{c_{i0}} p_1^{c_{i1}} \cdots p_b^{c_{ib}}$ von Zahlen in der erweiterten Basis $B' = B \cup \{-1\} = \{-1, p_1, \ldots, p_b\}$ zerlegen lässt
- Wir müssen dann nur darauf achten, dass wir $J \subseteq I$ so bestimmen, dass auch die Summe $e_0 = \sum_{i \in J} c_{i0}$ der Exponenten von -1 gerade ist
- Um für $y_i = x_i^2 \bmod n$ einen möglichst großen Wert zu erhalten, kann man x_i beispielsweise von der Form $\lfloor \sqrt{kn} \rfloor - l$ für kleine Zahlen $k, l \geq 0$ wählen

Die Methode der zufälligen Quadrate

Beispiel

- Sei $n = 1829$ und $B' = \{-1, 2, 3, 5, 7, 11, 13\}$
- Wegen $\sqrt{n} = 42{,}8$, $\sqrt{2n} = 60{,}5$, $\sqrt{3n} = 74{,}1$ und $\sqrt{4n} = 85{,}5$ testen wir die Zahlen 42, 43, 60, 61, 74, 75, 85, 86 und erhalten folgende Faktorisierungen über B':

$$
\begin{aligned}
x_1^2 &= 42^2 \equiv_n -65 = (-1)5^1 13^1 & x_2^2 &= 43^2 \equiv_n 20 = 2^2 5^1 \\
x_3^2 &= 61^2 \equiv_n 63 = 3^2 7^1 & x_4^2 &= 74^2 \equiv_n -11 = (-1)11^1 \\
x_5^2 &= 85^2 \equiv_n -91 = (-1)7^1 13^1 & x_6^2 &= 86^2 \equiv_n 80 = 2^4 5^1
\end{aligned}
$$

- Für $J = \{2, 6\}$ erhalten wir das Produkt $x_2^2 x_6^2 \equiv_n 2^6 5^2$
- Dieses führt auf die beiden Quadrate $a^2 \equiv_n b^2$ mit
  - $a = x_2 x_6 \bmod n = 43 \cdot 86 \bmod n = 40$ und
  - $b = 2^{6/2} 5^{2/2} \bmod n = 2^3 5^1 \bmod n = 40$
- Wegen $a = b = 40$ liefern diese Zahlen aber keinen nichttrivialen Faktor von n

Die Methode der zufälligen Quadrate

Beispiel (Fortsetzung)

- Dagegen erhalten wir für $J = \{1, 2, 3, 5\}$ das Produkt

$$x_1^2 x_2^2 x_3^2 x_5^2 \equiv_n (-1)^2 2^2 3^2 5^2 7^2 13^2$$

- Dieses führt auf die beiden Quadrate $a^2 \equiv_n b^2$ mit
  - $a = x_1 x_2 x_3 x_5 \bmod n = 42 \cdot 43 \cdot 61 \cdot 85 \bmod n = 1459$ und
  - $b = (-1)^{e_0/2} p_1^{e_1/2} \cdots p_6^{e_6/2} \bmod n = -2 \cdot 3 \cdot 5 \cdot 7 \cdot 13 \bmod n = 901$
- Die Zahlen $a = 1459$ und $b = 901$ führen wegen $a \not\equiv_n \pm b$ auf den nichttrivialen Faktor $\mathrm{ggT}(a - b, n) = 31$ von n ◁