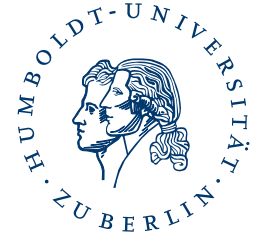
Kryptologie

Johannes Köbler



Institut für Informatik
Humboldt-Universität zu Berlin

WS 2022/23

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
- Bei geeigneter Wahl von G und α 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
```

- Der naive 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 in einer Precomputation eine Tabelle der Logarithmen aller Potenzen $\beta \in [\alpha]$ erstellen, können wir damit den diskreten Logarithmus durch Table-Lookup für jede Eingabe β in konstanter Zeit bestimmen

DLP-Berechnung mittels Precomputation

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

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^{jr}, $j = 0, \ldots, r-1$ und $r = \lceil \sqrt{n} \rceil$
- 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 $\texttt{Shanks}(G, n, \alpha, \beta, r = \lceil \sqrt{n} \rceil)$

Precomputation: Sortiere die Paare (α^{ir}, i), $0 \leq i \leq r-1$, nach der ersten Komponente in eine Tabelle $T1$

Computation: Sortiere die Paare $(\beta\alpha^{-j}, j)$, $0 \leq j \leq r-1$, nach der ersten Komponente in eine Tabelle $T2$ und ermittle durch parallele sequentielle Suche die zwei Paare (γ, i) in $T1$ und (γ, j) in $T2$ mit derselben ersten Komponente

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

Der Algorithmus von Shanks

Beispiel. Sei $G = \mathbb{Z}_{809}^*$, $\alpha = 3$ mit $ord(3) = 808$ und $\beta = 525$

- Dann ist $r = \lceil \sqrt{808} \rceil = 29$ und $\alpha^{29} \bmod 809 = 99$
- Sortieren wir die Paare (α^{ir}, i), $0 \leq i \leq 28$, so erhalten wir $T1$:

α^{ir}	1	15	26	81	93	99	147	207	211	268	275	295	308	329
i	0	23	12	28	2	1	13	8	6	9	19	27	3	5
464	496	528	559	564	575	586	632	**644**	654	664	676	727	781	800
17	21	20	4	22	26	25	18	10	11	7	24	15	16	14

- Sortieren der Paare $(\beta\alpha^{-j}, j)$, $0 \leq j \leq 28$, ergibt $T2$:

$\beta\alpha^{-j}$	44	132	133	163	175	256	259	314	328	355	356	379	388	396
j	6	5	17	28	1	13	24	18	2	14	25	3	15	4
399	440	489	511	521	525	554	**644**	658	686	724	754	768	713	777
16	10	27	9	21	0	7	19	26	11	8	20	12	22	23

- Suchen wir nun nach Einträgen (γ, i) in $T1$ und (γ, j) in $T2$ mit derselben ersten Komponente, so finden wir die beiden Paare $(644, 10)$ und $(644, 19)$, die auf $\log_3(525) = 29 \cdot 10 + 19 \bmod 808 = 309$ führen ◁

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 $b_j = n/n_j$ gibt es wegen $\mathrm{ggT}(b_j, n_j) = 1$ Zahlen c_j und d_j mit $c_j b_j + d_j n_j = \mathrm{ggT}(b_j, n_j) = 1$
- Diese sind mit dem erweiterten euklidschen Algorithmus effizient berechenbar und es gilt $c_j \equiv b_j^{-1} \bmod n_j$
- Daher erfüllt die Zahl $x_j = c_j b_j \bmod n_j$ für $i = 1, \ldots, k$ die Kongruenz

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

Beweis des chinesischen Restsatzes

Beweis (Fortsetzung).

- Daher erfüllt die Zahl $x_j = c_j b_j \bmod n_j$ für $i = 1, \ldots, k$ die Kongruenz

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

- Folglich erfüllt die Zahl $a = \sum_{j=1}^{k} a_j x_j \bmod n$ die Kongruenzen

$$a \overset{(1)}{\equiv}_{n_i} a_i x_i \overset{(2)}{\equiv}_{n_i} a_i$$

  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^e eines Elements $\alpha \in G$ der Ordnung n ist $\text{ord}_G(\alpha^e) = n/g$ für $g = \text{ggT}(n, e)$ (wegen $g = \lambda n + \mu e$ ist $[\alpha^e] = [\alpha^g]$)
- Im Fall $q|n$ hat $\alpha^{n/q}$ also die Ordnung $n/\text{ggT}(n, n/q) = q$
- Sei $a = \log_{G,\alpha}(\beta)$ der diskrete Logarithmus von β zur Basis α
- Weiter sei $\prod_{i=1}^{k} p_i^{e_i}$ die Primfaktorzerlegung von $n = ord(\alpha)$
- Falls wir die Werte $a_i = a \bmod p_i^{e_i}$ für $i = 1, \ldots, k$ kennen, so können wir a mit dem Chinesischen Restsatz berechnen
- Schreiben wir a_i als Zahl zur Basis p_i, so erhalten wir Ziffern $z_{ij} \in \mathbb{Z}_{p_i}$ mit

$$a_i = (z_{i,\,e_i-1} \cdots z_{i0})_{p_i} = \sum_{j=0}^{e_i-1} z_{ij} p_i^j$$

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

Der Pohlig-Hellman-Algorithmus

- Um nun die Ziffer z_{i0} zu berechnen, betrachten wir die Gleichung

$$\beta^{n/p_i} = \alpha^{z_{i0}n/p_i} \quad \text{bzw.} \quad z_{i0} = \log_{G,\alpha^{n/p_i}}(\beta^{n/p_i})$$

- Diese lässt sich leicht verifizieren:

$$\begin{aligned}\beta^{n/p_i} &= (\alpha^a)^{n/p_i} = (\alpha^{z_{i0}+z_{i1}p_i+\cdots+z_{i,\,e_i-1}p_i^{e_i-1}+d_ip_i^{e_i}})^{n/p_i} \\ &= (\alpha^{z_{i0}+tp_i})^{n/p_i} \text{ für eine Zahl } t \geq 0 \\ &= \alpha^{z_{i0}n/p_i}\alpha^{tn} = \alpha^{z_{i0}n/p_i}\end{aligned}$$

- Dies lässt sich leicht verallgemeinern, indem wir für $j = 0, \ldots, e_i - 1$ $\beta_{ij} = \beta\alpha^{-z_{i0}-z_{i1}p_i-z_{i2}p_i^2\cdots-z_{i,\,j-1}p_i^{j-1}}$ setzen und die Gleichung

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

betrachten

Der Pohlig-Hellman-Algorithmus

- Dies lässt sich leicht verallgemeinern, indem wir für $j = 0, \ldots, e_i - 1$ $\beta_{ij} = \beta\alpha^{-z_{i0}-z_{i1}p_i-z_{i2}p_i^2\cdots-z_{i,j-1}p_i^{j-1}}$ setzen und die Gleichung

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

betrachten, welche sich ebenfalls leicht verifizieren lässt:

$$\begin{aligned}
\beta_{ij}^{n/p_i^{j+1}} &= (\beta\alpha^{-z_{i0}-z_{i1}p_i-z_{i2}p_i^2\cdots-z_{i,j-1}p_i^{j-1}})^{n/p_i^{j+1}} \\
&= (\alpha^{a-z_{i0}-z_{i1}p_i-z_{i2}p_i^2\cdots-z_{i,j-1}p_i^{j-1}})^{n/p_i^{j+1}} \\
&= (\alpha^{z_{ij}p_i^j+z_{i,j+1}p_i^{j+1}+\cdots+z_{i,e_i-1}p_i^{e_i-1}+d_ip_i^{e_i}})^{n/p_i^{j+1}} \\
&= (\alpha^{z_{ij}p_i^j+tp_i^{j+1}})^{n/p_i^{j+1}} \text{ für eine Zahl } t \geq 0 \\
&= \alpha^{z_{ij}n/p_i}\alpha^{tn} = \alpha^{z_{ij}n/p_i}
\end{aligned}$$

- Wegen $\text{ord}_G(\alpha^{n/p_i}) = p_i$ können wir also für $j = 0, \ldots, e_i - 1$ die Ziffer z_{ij} mit dem Algorithmus von Shanks in Zeit $\mathcal{O}(\sqrt{p_i})$ berechnen

Der Pohlig-Hellman-Algorithmus

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

1 **for** $i := 1$ **to** k **do**
2 $\beta_{i,0} := \beta$
3 **for** $j := 0$ **to** $e_i - 1$ **do**
4 $z_{ij} := \log_{G,\alpha^{n/p_i}}(\beta_{ij}^{n/p_i^{j+1}})$ // *mit Shanks*
5 $\beta_{i,j+1} := \beta_{ij}\alpha^{-z_{ij}p_i^j}$
6 $a_i := (z_{i,e_i-1} \cdots z_{i,0})_{p_i}$
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 $m = 29^3 = 24389$ sowie $\alpha = 3$ 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 folgende Werte für $\beta_{i,j+1} = \beta_{ij}\alpha^{-z_{ij}p_i^j}$, $z_{ij} = \log_{G,\alpha^{n/p_i}}(\beta_{ij}^{n/p_i^{j+1}})$ sowie a_i:

i	p_i	e_i	$n_i = p_i^{e_i}$	n/p_i	α^{n/p_i}	j	β_{ij}	n/p_i^{j+1}	$\beta_{ij}^{n/p_i^{j+1}}$	z_{ij}	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 $\mathrm{ord}_G(\alpha) = n = 23548$
- 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
- Falls n nur einen Primteiler hat, also eine Primzahlpotenz ist, lässt sich n durch Berechnung der k-ten Wurzel für ein $k \leq \log_2(n)$ faktorisieren
- Sei n also eine Zahl mit mindestens zwei Primteilern $p < q$
- 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 $x_i \neq x_j$ mit $x_i \equiv_p x_j$
- Wegen $x_i \equiv_p x_j$, aber $x_i \not\equiv_n x_j$ führen diese auf einen nichttrivialen Faktor $d_{ij} = \mathrm{ggT}(x_i - x_j, n)$ mit $p \leq d < n$ von n
- Da wir aber p nicht kennen, können wir ein solches Kollisionspaar (x_i, x_j) in $\binom{X}{2}$ nicht dadurch finden, dass wir die Werte $r_i = x_i \bmod p$ sortieren, sondern wir müssen schlimmstenfalls alle Teiler d_{ij} berechnen

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 $x_j \equiv_p x_i$ mit $i < j \leq 1{,}17\sqrt{p}$ 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 alle $k \geq 0$ die Kongruenz $x_{i+k} \equiv_p x_{j+k}$ bzw. für alle $k \geq i$ die Kongruenz $x_k \equiv_p x_{k+j-i}$
- Setzen wir $\ell := j - i$, so erhalten wir für alle $k \geq i$ die Kongruenz $x_k \equiv_p x_{k+\ell}$ und daraus $x_k \equiv_p x_{k+d\ell}$ für alle $k \geq i$ und $d \geq 0$

Der Rho-Faktorisierungsalgorithmus

- Setzen wir $\ell := j - i$, so erhalten wir für alle $k \geq i$ die Kongruenz $x_k \equiv_p x_{k+\ell}$ und daraus $x_k \equiv_p x_{k+d\ell}$ für alle $k \geq i$ und $d \geq 0$
- Insbesondere folgt also $x_k \equiv_p x_{2k}$ für alle $k \geq i$ mit $k \equiv_\ell 0$
- Sei k_{min} das kleinste k mit $x_k \equiv_p x_{2k}$, dann gilt $i \leq k_{min} < i + \ell = j$
- Indem wir also sukzessive die Paare (x_k, y_k) mit $y_k = x_{2k}$ berechnen, können wir mit sehr geringem Platzverbrauch ein Paar (x_k, y_k) mit $\mathrm{ggT}(x_k - y_k, n) > 1$ und $k < i + \ell = j$ finden (ohne p zu kennen):

Algorithmus `Rho-Factorize`(n)

```
1 wähle zufällig x ∈_R ℤ_n
2 y := x^2 + 1 mod n
3 while ggT(x − y, n) = 1 do
4    x := f(x)
5    y := f(f(y))
6 if  g = ggT(x − y, n) < n then output(g)
7 else output(?)
```

Der Rho-Faktorisierungsalgorithmus

Beispiel. Sei $n = 7171 = 71 \cdot 101$, $f(x) = x^2 + 1 \bmod n$

- Dann führt der Startwert $x_1 = 1$ auf folgende Werte x_k, $y_k = x_{2k}$, $g_k = \mathrm{ggT}(x_k - y_k, n)$ sowie $r_k = x_k \bmod 71$:

k	1	2	3	4	5	6	7	8	9	10	**11**
x_k	1	2	5	26	677	6557	4105	6347	4903	2218	219
y_k	2	26	6557	6347	2218	4936	4560	375	4389	5471	574
g_k	1	1	1	1	1	1	1	1	1	1	71
r_k	1	2	5	26	38	25	**58**	28	4	17	6

k	12	13	14	15	16	17	18	19	20	21	22
x_k	4936	4210	4560	4872	375	4377	4389	2016	5471	88	574
r_k	37	21	16	44	20	46	**58**	28	4	17	6

- Es gilt $x_i \equiv_{71} x_j \equiv_{71} \mathbf{58}$ für $i = 7$ und $j = 18$, also ist $\ell = 18 - 7 = 11$
- Zudem gilt $x_k \equiv_{71} y_k$ für alle $k \geq 7$ mit $k \equiv_\ell 0$, d.h. $k_{min} = \mathbf{11}$ ◁

Der Pollard-Rho-DLP-Algorithmus

- Der Rho-DLP-Algorithmus berechnet eine pseudozufällige Folge von Paaren $(b_i, c_i) \in \mathbb{Z}_n \times \mathbb{Z}_n$
- Ziel ist es, zwei verschiedene Paare $(b_i, c_i) \neq (b_j, c_j)$ mit $\alpha^{b_i}\beta^{c_i} = \alpha^{b_j}\beta^{c_j}$ zu finden
- Dann erfüllt der diskrete Logarithmus $a = \log_{G,\alpha}(\beta)$ wegen

$$\alpha^{b_i + ac_i} = \alpha^{b_i}\beta^{c_i} = \alpha^{b_j}\beta^{c_j} = \alpha^{b_j + ac_j}$$

  die lineare Kongruenz $x(c_j - c_i) \equiv_n (b_i - b_j) \quad (*)$
- Im Fall $\mathrm{ggT}(c_j - c_i, n) = 1$ ist also $a = (b_i - b_j)(c_j - c_i)^{-1} \bmod n$
- Andernfalls erhalten wir $g = \mathrm{ggT}(c_j - c_i, n)$ Kandidaten $a_1, \ldots, a_g$, unter denen der richtige ebenfalls leicht zu ermitteln ist, sofern g nicht zu groß ist
- Der folgende Algorithmus benutzt zur Bildung der Pseudozufallsfolge eine Funktion $f : (x_i, b_i, c_i) \mapsto (x_{i+1}, b_{i+1}, c_{i+1})$, die aus Effizienzgründen auch gleich die Werte $x_i = \alpha^{b_i}\beta^{c_i}$ berechnet

Der Pollard-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**$(\alpha x, c + 1 \bmod n, d)$
4 $\quad x \in S_3$: **return**$(x^2, 2c \bmod n, 2d \bmod n)$

Algorithmus `Rho-DLP`(G, n, α, β)

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

Der Rho-DLP-Algorithmus

Beispiel. Sei $G = \mathbb{Z}_p^*$, $p = 809$, $\alpha = 89$, $n = ord(\alpha) = 101$ sowie $\beta = 618$

- Benutzen wir für $i = 1, 2, 3$ die Mengen $S_i = \{x \in G \mid x \equiv_3 i\}$, so führt das Tripel $(x_1, b_1, c_1) = (618, 0, 1)$ auf die Folgen (x_k, b_k, c_k) und $(y_k, d_k, e_k) = (x_{2k}, b_{2k}, c_{2k})$:

k	1	2	3	4	5	6	7	8	9	**10**
x_k	618	76	46	113	349	488	555	605	**451**	**422**
b_k	0	0	0	0	1	1	2	4	5	**5**
c_k	1	2	3	4	4	5	5	10	10	**11**
y_k	76	113	488	605	422	683	**451**	344	112	**422**
d_k	0	0	1	4	5	7	8	9	11	**11**
e_k	2	4	5	10	11	11	12	13	13	**15**

- Es gilt $x_i = x_j = \mathbf{451}$ für $i = 9$ und $j = 14$, d.h. $\ell = 14 - 9 = 5$; zudem gilt $x_k = y_k$ für alle $k \geq 9$ mit $k \equiv_\ell 0$, d.h. $k_{min} = \mathbf{10}$ und $x_{10} = y_{10} = \mathbf{422}$
- Wegen $\mathrm{ggT}(e_{10} - c_{10}, n) = \mathrm{ggT}(15 - 11, n) = 1$ führt dies auf den Wert $\log_{G,\alpha}(\beta) = (\mathbf{5} - \mathbf{11})(\mathbf{15} - \mathbf{11})^{-1} \bmod n = -6 \cdot 4^{-1} \bmod 101 = 49$ ◁

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_3 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 $\mathrm{ord}(\alpha) = p - 1$
- 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:
  repeat
    wähle zufällig eine Zahl $s \in \{0, \ldots, p-2\}$
    $\gamma := \beta\alpha^s \bmod p$
  until γ ist über B faktorisierbar
  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

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

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

$$\ell_1 = 5136 - 3\ell_2 \bmod p - 1 = 6578,$$
$$\ell_4 = 9865 - 3\ell_2 \bmod p - 1 = 1301 \text{ und}$$
$$\ell_3 = \log_\alpha p_3 = \log_5 5 = 1$$

- 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 $y_i = x_i^2 \bmod n$ für $i \in I = \{1, 2, 3\}$:

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

- Für $J = I = \{1, 2, 3\}$ 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
  - $b = 2 \cdot 3 \cdot 7 \cdot 13 \bmod n = 546$
- Die beiden 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 $(-1)^{c_{i0}} p_1^{c_{i1}} \cdots p_b^{c_{ib}}$ 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 $y_i' = n - y_i$ einen kleinen Wert (bzw. für $y_i = x_i^2 \bmod n$ einen 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 $(x_1, \ldots, x_8) = (42, 43, 60, 61, 74, 75, 85, 86)$ und erhalten für $I = \{1, 2, 4, 5, 7, 8\}$ folgende Faktorisierungen der Zahlen x_i, $i \in I$:

$$x_1^2 = 42^2 \equiv_n -65 = (-1)5^1 13^1 \qquad x_2^2 = 43^2 \equiv_n 20 = 2^2 5^1$$
$$x_4^2 = 61^2 \equiv_n 63 = 3^2 7^1 \qquad x_5^2 = 74^2 \equiv_n -11 = (-1)11^1$$
$$x_7^2 = 85^2 \equiv_n -91 = (-1)7^1 13^1 \qquad x_8^2 = 86^2 \equiv_n 80 = 2^4 5^1$$

- Für $J = \{2, 8\}$ erhalten wir das Produkt $x_2^2 x_8^2 \equiv_n 2^6 5^2$
- Dieses führt auf die beiden Quadrate $a^2 \equiv_n b^2$ mit
  - $a = x_2 x_8 \bmod n = 43 \cdot 86 \bmod n = 3698 \bmod n = 40$ 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 $J = \{1, 2, 4, 7\}$ das Produkt

$$x_1^2 x_2^2 x_4^2 x_7^2 \equiv_n (-5^1 13^1)(2^2 5^1)(3^2 7^1)(-7^1 13^1) = (-1)^2 2^2 3^2 5^2 7^2 13^2$$

- Dieses führt auf die beiden Quadrate $a^2 \equiv_n b^2$ mit
  - $a = x_1 x_2 x_4 x_7 \bmod n = 42 \cdot 43 \cdot 61 \cdot 85 \bmod n = 1459$ und
  - $b = -\underbrace{2 \cdot 3 \cdot 5 \cdot 7 \cdot 13}_{901} \bmod n = 928$
- Wegen $a \not\equiv_n \pm b$ liefern die Zahlen $a = 1459$ und $b = 928$ den nichttrivialen Faktor $\mathrm{ggT}(531, n) = 59$ von n ◁

Eine untere Schranke für generische DLP-Algorithmen

- Wir gehen nun der Frage nach, wie effizient der diskrete Logarithmus $\log_{\alpha,G} \beta$ berechenbar ist, wenn über die Gruppe G nichts bekannt ist, außer dass $\alpha \in G$ die Ordnung n hat und $\beta \in [\alpha]$ ist
- Ein Algorithmus, der das DLP unter dieser Voraussetzung löst, heißt generisch
- Dabei nehmen wir an, dass sich die Gruppenoperation und auch die Potenzierung von Elementen in $[\alpha]$ effizient ausführen lassen
- Um zu verhindern, dass der DLP-Algorithmus spezielle Eigenschaften von G ausnützen kann (bspw. lässt sich das DLP in der additiven Gruppe $(\mathbb{Z}_n, +)$ sehr effizient lösen), gehen wir davon aus, dass die Elemente von $[\alpha]$ durch beliebige Binärstrings kodiert sind
- Formal verwenden wir hierzu eine injektive Kodierungsfunktion $\sigma : \mathbb{Z}_n \to \{0,1\}^l$, die jedem Exponenten $i \in \mathbb{Z}_n$ eine (zufällig gewählte) Kodierung $\sigma(i) \in \{0,1\}^l$ des Elements α^i zuweist

Eine untere Schranke für generische DLP-Algorithmen

- Da ein generischer DLP-Algorithmus A zu Beginn der Rechnung nur die (Kodierungen der) beiden Elemente α und β kennt, kann er durch wiederholte Ausführung von Gruppen- und Potenzoperationen nur Elemente der Form $\alpha^c \beta^d$ berechnen
- Der Einfachheit halber nehmen wir an, dass A die Möglichkeit hat, für beliebige Paare $(c, d) \in \mathbb{Z}_n \times \mathbb{Z}_n$ die Kodierung von $\alpha^c \beta^d$ in einem Rechenschritt zu erfragen
- Weiterhin nehmen wir an, dass n eine Primzahl ist
- Sei also $\mathcal{C} = \{(c_1, d_1), \ldots, (c_m, d_m)\}$ die Menge aller erfragten Paare, wobei die Antworten α und β auf die beiden Paare $(c_1, d_1) = (1, 0)$ und $(c_2, d_2) = (0, 1)$ Teil der Eingabe sind
- Dann kann A den gesuchten Wert $a = \log_{\alpha, G} \beta$ berechnen, wenn zwei Paare $(c_i, d_i) \neq (c_j, d_j)$ auf die gleiche Antwort $\alpha^{c_i} \beta^{d_i} = \alpha^{c_j} \beta^{d_j}$ führen
- Wegen $\alpha^{c_i - c_j} = \beta^{d_j - d_i}$ gilt dann nämlich $a = (c_i - c_j)(d_j - d_i)^{-1} \bmod n$

Eine untere Schranke für generische DLP-Algorithmen

- Genauer folgt im Fall $d_j \neq d_i$ aus der Gleichheit $\alpha^{c_i}\beta^{d_i} = \alpha^{c_j}\beta^{d_j}$, dass $a = (c_i - c_j)(d_j - d_i)^{-1} \bmod n$, und aus $\alpha^{c_i}\beta^{d_i} \neq \alpha^{c_j}\beta^{d_j}$, dass $a \neq (c_i - c_j)(d_j - d_i)^{-1} \bmod n$ ist
- Wir betrachten zuerst den Fall, dass A nur nichtadaptive Fragen stellt
- Beispielsweise ist der Shanks-Algorithmus ein nichtadaptiver generischer DLP-Algorithmus

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^{jr}, $j = 0, \ldots, r-1$ und $r = \lceil \sqrt{n} \rceil$
- 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 $\texttt{Shanks}(G, n, \alpha, \beta, r = \lceil \sqrt{n} \rceil)$

Precomputation: Sortiere die Paare (α^{ir}, i), $0 \leq i \leq r-1$, nach der ersten Komponente in eine Tabelle $T1$

Computation: Sortiere die Paare $(\beta\alpha^{-j}, j)$, $0 \leq j \leq r-1$, nach der ersten Komponente in eine Tabelle $T2$ und ermittle durch parallele sequentielle Suche die zwei Paare (γ, i) in $T1$ und (γ, j) in $T2$ mit derselben ersten Komponente

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

Eine untere Schranke für generische DLP-Algorithmen

- Genauer folgt im Fall $d_j \neq d_i$ aus der Gleichheit $\alpha^{c_i}\beta^{d_i} = \alpha^{c_j}\beta^{d_j}$, dass $a = (c_i - c_j)(d_j - d_i)^{-1} \bmod n$, und aus $\alpha^{c_i}\beta^{d_i} \neq \alpha^{c_j}\beta^{d_j}$, dass $a \neq (c_i - c_j)(d_j - d_i)^{-1} \bmod n$ ist
- Wir betrachten zuerst den Fall, dass A nur nichtadaptive Fragen stellt
- Beispielsweise ist der Shanks-Algorithmus ein nichtadaptiver generischer DLP-Algorithmus
- Weiterhin nehmen wir an, dass die Eingabe β und damit der Wert von $a = \log_{\alpha,G}(\beta)$ zufällig gewählt wird
- Sei $Good(\mathcal{C}) = \{(c_i - c_j)(d_j - d_i)^{-1} \bmod n \mid 1 \leq i < j \leq m, d_j \neq d_i\}$
- Dann kann A den Wert a im Fall $a \in Good(\mathcal{C})$ mit Sicherheit korrekt bestimmen
- Setzen wir $g = |Good(\mathcal{C})|$, so tritt dieser Fall mit Wahrscheinlichkeit $g/n \leq \binom{m}{2}/n$ ein

Eine untere Schranke für generische DLP-Algorithmen

- Dagegen gelingt dies im Fall $a \notin \mathit{Good}(\mathcal{C})$ nur mit Wahrscheinlichkeit $1/(n-g)$, und zwar unabhängig davon, nach welcher Strategie A den Ausgabewert aus der Menge $\mathbb{Z}_n - \mathit{Good}(\mathcal{C})$ auswählt
- Wegen $\Pr[a \in \mathit{Good}(\mathcal{C})] = g/n$ erhalten wir folgende Abschätzung für die Erfolgswahrscheinlichkeit γ von A:

$$\begin{aligned}\gamma \;&\leq\; \underbrace{\Pr[a \in \mathit{Good}(\mathcal{C})]}_{g/n} \cdot \underbrace{\Pr[A \text{ gibt } a \text{ aus} \mid a \in \mathit{Good}(\mathcal{C})]}_{1} \\ &\quad + \underbrace{\Pr[a \notin \mathit{Good}(\mathcal{C})]}_{(1-g/n)} \cdot \underbrace{\Pr[A \text{ gibt } a \text{ aus} \mid a \notin \mathit{Good}(\mathcal{C})]}_{1/(n-g)} \\ &=\; (g+1)/n \leq \tfrac{\binom{m}{2}+1}{n}\end{aligned}$$

- Um eine Erfolgswahrscheinlichkeit $\gamma = \Omega(1)$ zu erreichen, muss A also mindestens $m = \Omega(\sqrt{n})$ Fragen stellen

Eine untere Schranke für generische DLP-Algorithmen

- Abschließend betrachten wir den Fall, dass A adaptive Fragen stellt
- Da die Antworten zufällig gewählte Binärstrings sind, können sie offensichtlich nicht bei der Suche nach nachfolgenden Fragen von Nutzen sein
- Der einzige Vorteil, den ein adaptiver generischer DLP-Algorithmus A hat, besteht darin, dass er sofort die Rechnung beenden kann, sobald er auf zwei verschiedene Fragen die gleiche Antwort erhält
- Legen wir aber von vornherein eine Obergrenze für die Anzahl der Fragen fest, so ergibt sich genau die gleiche Erfolgswahrscheinlichkeit wie im nichtadaptiven Fall