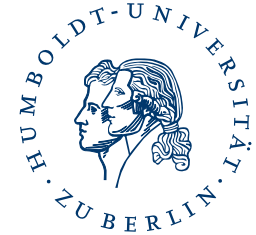
Kryptologie

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 $\beta_j = \beta\alpha^{-d_0-d_1p_i-d_2p_i^2\cdots-d_{j-1}p_i^{j-1}}$ 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:

$$\begin{aligned}
\beta_j^{n/p_i^{j+1}} &= (\alpha^{a-d_0-d_1p_i-d_2p_i^2\cdots-d_{j-1}p_i^{j-1}})^{n/p_i^{j+1}} \\
&= (\alpha^{d_jp_i^j+d_{j+1}p_i^{j+1}+\cdots+d_{e_i-1}p_i^{e_i-1}+s_ip_i^{e_i}})^{n/p_i^{j+1}} \\
&= (\alpha^{d_jp_i^j+tp_i^{j+1}})^{n/p_i^{j+1}} \text{ für eine Zahl } t \geq 0 \\
&= \alpha^{d_jn/p_i}\alpha^{tn} \\
&= \alpha^{d_jn/p_i}
\end{aligned}$$

- 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)

- 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
- Es ergeben sich die folgenden Werte:

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 $m = 24389$ ◁