Diskrete Logarithmen

10

In diesem Kapitel geht es um das Problem, diskrete Logarithmen zu berechnen (DL-Problem). Nur in Gruppen, in denen das DL-Problem schwierig zu lösen ist, können das ElGamal-Verschlüsselungsverfahren (siehe Abschn. 8.7) und viele andere Public-Key-Verfahren sicher sein. Daher ist das DL-Problem von großer Bedeutung in der Kryptographie.

Wir werden zuerst Algorithmen behandeln, die in allen Gruppen funktionieren. Dann werden spezielle Algorithmen für endliche Körper beschrieben. Eine Übersicht über Techniken und neuere Resultate findet man in [64] und in [42].

10.1 Das DL-Problem

In diesem Kapitel sei G eine endliche zyklische Gruppe der Ordnung n, γ sei ein Erzeuger dieser Gruppe und 1 sei das neutrale Element in G. Wir gehen davon aus, dass die Gruppenordnung n bekannt ist. Viele Algorithmen zur Lösung des DL-Problems funktionieren auch mit einer oberen Schranke für die Gruppenordnung. Ferner sei α ein Gruppenelement. Wir wollen die kleinste nicht negative ganze Zahl x finden, für die

$$\alpha = \gamma^x \tag{10.1}$$

gilt, d. h. wir wollen den *diskreten Logarithmus* von α zur Basis γ berechnen.

Man kann das DL-Problem auch allgemeiner formulieren: In einer Gruppe H, die nicht notwendig zyklisch ist, sind zwei Elemente α und γ gegeben. Gefragt ist, ob es einen Exponenten x gibt, der (10.1) erfüllt. Wenn ja, ist das kleinste nicht negative x gesucht. Man muss also erst entscheiden, ob es überhaupt einen diskreten Logarithmus gibt. Wenn ja, muss man ihn finden. In kryptographischen Anwendungen ist es aber fast immer klar, dass der diskrete Logarithmus existiert. Das Entscheidungsproblem ist also aus kryptographischer Sicht unbedeutend. Daher betrachten wir nur DL-Probleme in zyklischen Gruppen. Die Basis ist immer ein Erzeuger der Gruppe.

© Springer-Verlag Berlin Heidelberg 2016

J. Buchmann, *Einführung in die Kryptographie*, Springer-Lehrbuch,
DOI 10.1007/978-3-642-39775-2_10

10.2 Enumeration

Die einfachste Methode, den diskreten Logarithmus x aus (10.1) zu berechnen, besteht darin, für $x = 0, 1, 2, 3, \ldots$ zu prüfen, ob (10.1) erfüllt ist. Sobald die Antwort positiv ist, ist der diskrete Logarithmus gefunden. Dieses Verfahren bezeichnen wir als *Enumerationsverfahren*. Es erfordert $x - 1$ Multiplikationen und x Vergleiche in G. Man braucht dabei nur die Elemente α, γ und γ^x zu speichern. Also braucht man Speicherplatz für drei Gruppenelemente.

Beispiel 10.1 Wir bestimmen den diskreten Logarithmus von 3 zur Basis 5 in $(\mathbb{Z}/2017\mathbb{Z})^*$. Probieren ergibt $x = 1030$. Dafür braucht man 1029 Multiplikationen modulo 2017.

In kryptographischen Verfahren ist $x \geq 2^{160}$. Das Enumerationsverfahren ist dann nicht durchführbar. Man braucht dann nämlich wenigstens $2^{160} - 1$ Gruppenoperationen.

10.3 Shanks Babystep-Giantstep-Algorithmus

Eine erste Methode zur schnelleren Berechnung diskreter Logarithmen ist der Babystep-Giantstep-Algorithmus von Shanks. Bei diesem Verfahren muss man viel weniger Gruppenoperationen machen als bei der Enumeration, aber man braucht mehr Speicherplatz. Man setzt

$$m = \lceil \sqrt{n} \rceil$$

und macht den Ansatz

$$x = qm + r, \quad 0 \leq r < m.$$

Dabei ist r also der Rest und q ist der Quotient der Division von x durch m. Der Babystep-Giantstep-Algorithmus berechnet q und r. Dies geschieht folgendermaßen:

Es gilt

$$\gamma^{qm+r} = \gamma^x = \alpha.$$

Daraus folgt

$$(\gamma^m)^q = \alpha\gamma^{-r}.$$

Man berechnet nun zuerst die Menge der *Babysteps*

$$B = \{(\alpha\gamma^{-r}, r) : 0 \leq r < m\}.$$

Findet man ein Paar $(1, r)$, so kann man $x = r$ setzen und hat das DL-Problem gelöst. Andernfalls bestimmt man

$$\delta = \gamma^m$$

und prüft, ob für $q = 1, 2, 3, \ldots$ das Gruppenelement δ^q als erste Komponente eines Elementes von B vorkommt, ob also ein Paar (δ^q, r) zu B gehört. Sobald dies der Fall ist,

gilt

$$\alpha\gamma^{-r} = \delta^q = \gamma^{qm}$$

und man hat den diskreten Logarithmus

$$x = qm + r$$

gefunden. Die Berechnung der Elemente δ^q, $q = 1, 2, 3 \ldots$ nennt man *Giantsteps*. Um die Überprüfung, ob δ^q als erste Komponente eines Elementes der Babystep-Menge vorkommt, effizient zu gestalten, nimmt man die Elemente dieser Menge in eine Hashtabelle auf (siehe [22], Kapitel 12), wobei die erste Komponente eines jeden Elementes als Schlüssel dient.

Beispiel 10.2 Wir bestimmen den diskreten Logarithmus von 3 zur Basis 5 in $(\mathbb{Z}/2017\mathbb{Z})^*$. Es ist $\gamma = 5 + 2017\mathbb{Z}$, $\alpha = 3 + 2017\mathbb{Z}$, $m = \lceil\sqrt{2016}\rceil = 45$. Die Babystep-Menge ist

$$\begin{aligned} B = \{&(3,0),(404,1),(1291,2),(1065,3),(213,4),(446,5),(896,6),\\ &(986,7),(1004,8),(1411,9),(1089,10),(1428,11),(689,12),(1348,13),\\ &(673,14),(538,15),(511,16),(909,17),(1392,18),(1892,19),(1992,20),\\ &(2012,21),(2016,22),(1210,23),(242,24),(1662,25),(1946,26),\\ &(1196,27),(1046,28),(1016,29),(1010,30),(202,31),(1654,32),\\ &(1541,33),(1115,34),(223,35),(448,36),(493,37),(502,38),(1714,39),\\ &(1553,40),(714,41),(1353,42),(674,43),(1345,44)\}. \end{aligned}$$

Hierbei wurden die Restklassen durch ihre kleinsten nicht negativen Vertreter dargestellt.

Man berechnet nun $\delta = \gamma^m = 45 + 2017\mathbb{Z}$. Die Giantsteps berechnen sich zu

$$\begin{aligned} &45, 8, 360, 64, 863, 512, 853, 62, 773, 496, 133, 1951,\\ &1064, 1489, 444, 1827, 1535, 497, 178, 1959, 1424, 1553. \end{aligned}$$

Man findet $(1553, 40)$ in der Babystep-Menge. Es ist also $\alpha\gamma^{-40} = 1553 + 2017\mathbb{Z}$. Andererseits wurde 1553 als zweiundzwanzigster Giantstep gefunden. Also gilt

$$\gamma^{22*45} = \alpha\gamma^{-40}.$$

Damit ist

$$\gamma^{22*45+40} = \alpha.$$

Als Lösung des DL-Problems findet man $x = 22 * 45 + 40 = 1030$. Um die Babystep-Menge zu berechnen, brauchte man 45 Multiplikationen. Um die Giantsteps zu berechnen, musste man zuerst δ berechnen und dann brauchte man 21 Multiplikationen in G. Die Anzahl der Multiplikationen ist also deutlich geringer als beim Enumerationsverfahren, aber

man muss viel mehr Elemente speichern. Außerdem muss man für 22 Gruppenelemente β prüfen, ob es ein Paar (β, r) in der Babystep-Menge gibt.

Wenn man unterstellt, dass es mittels einer Hashtabelle möglich ist, mit konstant vielen Vergleichen zu prüfen, ob ein gegebenes Gruppenelement erste Komponente eines Paares in der Babystep-Menge ist, dann kann man folgenden Satz leicht verifizieren.

Theorem 10.1 *Der Babystep-Giantstep-Algorithmus benötigt* $O(\sqrt{|G|})$ *Multiplikationen und Vergleiche in* G. *Er muss* $O(\sqrt{|G|})$ *viele Elemente in* G *speichern.*

Der Zeit- und Platzbedarf des Babystep-Giantstep-Algorithmus ist von der Größenordnung $\sqrt{|G|}$. Ist $|G| > 2^{160}$, so ist der Algorithmus in der Praxis nicht mehr einsetzbar.

10.4 Der Pollard-ρ-Algorithmus

Das Verfahren von Pollard, das in diesem Abschnitt beschrieben wird, benötigt wie der Babystep-Giantstep-Algorithmus $O(\sqrt{|G|})$ viele Gruppenoperationen, aber nur konstant viele Speicherplätze.

Wieder wollen wir das DL-Problem (10.1) lösen. Gebraucht werden drei paarweise disjunkte Teilmengen G_1, G_2, G_3 von G, deren Vereinigung die ganze Gruppe G ist. Sei die Funktion $f : G \to G$ definiert durch

$$f(\beta) = \begin{cases} \gamma\beta & \text{falls } \beta \in G_1, \\ \beta^2 & \text{falls } \beta \in G_2, \\ \alpha\beta & \text{falls } \beta \in G_3. \end{cases}$$

Wir wählen eine Zufallszahl x_0 in der Menge $\{1, \ldots, n\}$ und setzen $\beta_0 = \gamma^{x_0}$. Dann berechnen wir die Folge (β_i) nach der Rekursion

$$\beta_{i+1} = f(\beta_i).$$

Wir können die Glieder dieser Folge darstellen als

$$\beta_i = \gamma^{x_i}\alpha^{y_i}, \quad i \geq 0.$$

Dabei ist x_0 der zufällig gewählte Startwert, $y_0 = 0$, und es gilt

$$x_{i+1} = \begin{cases} x_i + 1 \mod n & \text{falls } \beta_i \in G_1, \\ 2x_i \mod n & \text{falls } \beta_i \in G_2, \\ x_i & \text{falls } \beta_i \in G_3 \end{cases}$$

und

$$y_{i+1} = \begin{cases} y_i & \text{falls } \beta_i \in G_1, \\ 2y_i \mod n & \text{falls } \beta_i \in G_2, \\ y_i + 1 \mod n & \text{falls } \beta_i \in G_3. \end{cases}$$

Da es nur endlich viele verschiedene Gruppenelemente gibt, müssen in dieser Folge zwei gleiche Gruppenelemente vorkommen. Es muss also $i \geq 0$ und $k \geq 1$ geben mit $\beta_{i+k} = \beta_i$. Das bedeutet, dass

$$\gamma^{x_i} \alpha^{y_i} = \gamma^{x_{i+k}} \alpha^{y_{i+k}}.$$

Daraus folgt

$$\gamma^{x_i - x_{i+k}} = \alpha^{y_{i+k} - y_i}.$$

Für den diskreten Logarithmus x von α zur Basis γ gilt also

$$(x_i - x_{i+k}) \equiv x(y_{i+k} - y_i) \mod n.$$

Diese Kongruenz muss man lösen. Ist die Lösung nicht eindeutig mod n, so muss man die richtige Lösung durch Ausprobieren ermitteln. Wenn das nicht effizient genug geht, weil zu viele Möglichkeiten bestehen, wiederholt man die gesamte Berechnung mit einem neuen Startwert x_0.

Wir schätzen die Anzahl der Folgenglieder β_i ab, die berechnet werden müssen, bis ein *Match* gefunden ist, also ein Paar $(i, i+k)$ von Indizes, für das $\beta_{i+k} = \beta_i$ gilt. Dazu verwenden wir das Geburtstagsparadox. Die Geburtstage sind die Gruppenelemente. Wir nehmen an, dass die Elemente der Folge $(\beta_i)_{i \geq 0}$ unabhängig und gleichverteilt zufällig gewählt sind. Das stimmt zwar nicht, aber sie ist so konstruiert, dass sie einer Zufallsfolge sehr ähnlich ist. Wie in Abschn. 1.3.1 gezeigt, werden $O(\sqrt{|G|})$ Folgenelemente benötigt, damit die Wahrscheinlichkeit für ein Match größer als $1/2$ ist.

So, wie der Algorithmus bis jetzt beschrieben wurde, muss man alle Tripel (β_i, x_i, y_i) speichern. Der Speicherplatzbedarf ist dann $O(\sqrt{|G|})$, wie beim Algorithmus von Shanks. Tatsächlich genügt es aber, nur ein Tripel zu speichern. Der Pollard-ρ-Algorithmus ist also viel Speicher-effizienter als der Algorithmus von Shanks. Am Anfang speichert man (β_1, x_1, y_1). Hat man gerade (β_i, x_i, y_i) gespeichert, so berechnet man (β_j, x_j, y_j) für $j = i+1, i+2, \ldots$ bis man ein Match (i, j) findet oder bis $j = 2i$ ist. Im letzteren Fall löscht man β_i und speichert statt dessen β_{2i}. Gespeichert werden also nur die Tripel (β_i, x_i, y_i) mit $i = 2^k$. Bevor gezeigt wird, dass im modifizierten Algorithmus wirklich ein Match gefunden wird, geben wir ein Beispiel:

Beispiel 10.3 Wir lösen mit dem Pollard-ρ-Algorithmus die Kongruenz

$$5^x \equiv 3 \mod 2017.$$

Alle Restklassen werden durch ihre kleinsten nicht negativen Vertreter dargestellt. Wir setzen

$$G_1 = \{1, \ldots, 672\}, G_2 = \{673, \ldots, 1344\}, G_3 = \{1345, \ldots, 2016\}.$$

Als Startwert nehmen wir $x_0 = 1023$. Wir geben nur die Werte für die gespeicherten Tripel an und zusätzlich das letzte Tripel, das die Berechnung des diskreten Logarithmus ermöglicht.

j	β_j	x_j	y_j
0	986	1023	0
1	2	29	0
2	10	31	0
4	250	33	0
8	1366	136	1
16	1490	277	8
32	613	447	155
64	1476	1766	1000
98	1476	966	1128

Man erkennt, dass

$$5^{800} \equiv 3^{128} \mod 2017$$

ist. Zur Berechnung von x müssen wir die Kongruenz

$$128x \equiv 800 \mod 2016$$

lösen. Da $\gcd(128, 2016) = 32$ ein Teiler von 800 ist, existiert eine Lösung, die mod 63 eindeutig ist. Um x zu finden, löst man erst

$$4z \equiv 25 \bmod 63.$$

Wir erhalten $z = 22$. Der gesuchte diskrete Logarithmus ist einer der Werte $x = 22 + k * 63$, $0 \leq k < 32$. Für $k = 16$ findet man den diskreten Logarithmus $x = 1030$.

Auf die oben beschriebene Weise wird tatsächlich immer ein Match gefunden. Das wird in Abb. 10.1 illustriert und jetzt bewiesen.

Zuerst zeigen wir, dass die Folge $(\beta_i)_{i \geq 0}$ periodisch wird. Sei $(s, s + k)$ das erste Match. Dann ist $k > 0$ und $\beta_{s+k} = \beta_s$. Weiter gilt $\beta_{s+k+l} = \beta_{s+l}$ für alle $l \geq 0$. Das liegt an der Konstruktion der Folge $(\beta_i)_{i \geq 0}$. Die Folge wird also tatsächlich periodisch. Man kann sie zeichnen wie den griechischen Buchstaben ρ. Die *Vorperiode* ist die Folge $\beta_0, \beta_1, \ldots, \beta_{s-1}$. Sie hat die Länge s. Die *Periode* ist die Folge $\beta_s, \beta_{s+1}, \ldots, \beta_{s+k-1}$. Sie hat die Länge k.

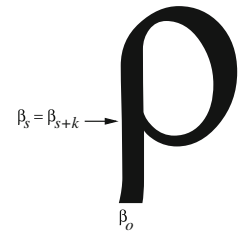

Abb. 10.1 Der Pollard-ρ-Algorithmus

Nun erläutern wir, wann ein Match gefunden wird. Ist $i = 2^j \geq s$, so liegt das gespeicherte Element β_i in der Periode. Ist zusätzlich $2^j \geq k$, so ist die Folge

$$\beta_{2^j+1}, \beta_{2^j+2}, \ldots, \beta_{2^{j+1}}$$

wenigstens so lang wie die Periode. Eins ihrer Glieder stimmt also mit β_{2^j} überein. Diese Folge wird aber berechnet, nachdem b_{2^j} gespeichert wurde, und alle ihre Elemente werden mit β_{2^j} verglichen. Bei diesen Vergleichen wird also ein Match gefunden. Da die Summe aus Vorperioden- und Periodenlänge $O(\sqrt{|G|})$ ist, wird also nach Berechnung von $O(\sqrt{|G|})$ Folgengliedern ein Match gefunden. Der Algorithmus braucht also $O(\sqrt{|G|})$ Gruppenoperationen und muss $O(1)$ Tripel speichern.

Der Algorithmus wird effizienter, wenn man nicht nur ein, sondern acht Tripel speichert. Das macht man so: Zuerst sind diese 8 Tripel alle gleich (β_0, x_0, y_0). Später werden die Tripel nach und nach durch andere ersetzt. Sei i der Index des letzten gespeicherten Tripels. Am Anfang ist $i = 1$.

Für $j = 1, 2, \ldots$ berechnet man nun (β_j, x_j, y_j) und macht folgendes:

1. Wenn β_j mit einem gespeicherten Gruppenelement übereinstimmt, bricht man die Berechnung der β_j ab und versucht, den diskreten Logarithmus zu bestimmen.
2. Wenn $j \geq 3i$ ist, löscht man das erste gespeicherte Tripel und nimmt (β_j, x_j, y_j) als neues letztes gespeichertes Tripel.

Diese Modifikation ändert aber nichts an der Laufzeit oder dem Speicherbedarf des Algorithmus.

10.5 Der Pohlig-Hellman-Algorithmus

Wir zeigen nun, wie man das Problem der Berechnung diskreter Logarithmen in G auf dasselbe Problem in Gruppen von Primzahlordnung reduzieren kann, wenn man die Faktorisierung der Gruppenordnung $|G|$ kennt. Es sei also

$$n = |G| = \prod_{p|n} p^{e(p)}$$

die Primfaktorzerlegung von $n = |G|$.

10.5.1 Reduktion auf Primzahlpotenzordnung

Für jeden Primteiler p von n setzen wir

$$n_p = n/p^{e(p)}, \quad \gamma_p = \gamma^{n_p}, \quad \alpha_p = \alpha^{n_p}.$$

Dann ist die Ordnung von γ_p genau $p^{e(p)}$ und es gilt

$$\gamma_p^x = \alpha_p.$$

Das Element α_p liegt also in der von γ_p erzeugten zyklischen Untergruppe von G. Daher existiert der diskrete Logarithmus von α_p zur Basis γ_p. Der folgende Satz beschreibt die Berechnung von x aus den diskreten Logarithmen der α_p zur Basis γ_p.

Theorem 10.2 *Für alle Primteiler p von n sei $x(p)$ der diskrete Logarithmus von α_p zur Basis γ_p. Außerdem sei $x \in \{0, 1, \ldots, n-1\}$ Lösung der simultanen Kongruenz $x \equiv x(p) \bmod p^{e(p)}$ für alle Primteiler p von n. Dann ist x der diskrete Logarithmus von α zur Basis γ.*

Beweis Es gilt

$$(\gamma^{-x}\alpha)^{n_p} = \gamma_p^{-x(p)}\alpha_p = 1$$

für alle Primteiler p von n. Daher ist die Ordnung des Elementes $\gamma^{-x}\alpha$ ein Teiler von n_p für alle Primteiler p von n und damit ein Teiler des größten gemeinsamen Teilers aller n_p. Dieser größte gemeinsame Teiler ist aber 1. Die Ordnung ist also 1 und damit gilt $\alpha = \gamma^x$. □

Man kann also x berechnen, indem man zuerst alle $x(p)$ bestimmt und dann den chinesischen Restsatz anwendet. Zur Berechnung eines $x(p)$ braucht der Babystep-Giantstep-Algorithmus oder der Algorithmus von Pollard nur noch $O(\sqrt{p^{e(p)}})$ viele Gruppenoperationen. Wenn n mehr als einen Primfaktor hat, ist das bereits deutlich schneller als wenn einer dieser Algorithmen in der gesamten Gruppe G angewendet wird. Die Rechenzeit für den chinesischen Restsatz kann man vernachlässigen.

Beispiel 10.4 Wie in Beispiel 10.2 sei G die prime Restklassengruppe mod 2017. Ihre Ordnung ist

$$2016 = 2^5 * 3^2 * 7.$$

Nach obiger Reduktion muss man $x(2)$ in einer Untergruppe der Ordnung $2^5 = 32$ bestimmen, $x(3)$ wird in einer Untergruppe der Ordnung 9 berechnet und $x(7)$ in einer Untergruppe der Ordnung 7. Dies wird im nächsten Abschnitt noch weiter vereinfacht.

10.5.2 Reduktion auf Primzahlordnung

Im vorigen Abschnitt haben wir gesehen, dass man die Berechnung diskreter Logarithmen in einer zyklischen Gruppe, für die man die Faktorisierung der Gruppenordnung kennt, auf DL-Berechnungen in Gruppen von Primzahlpotenzordnung zurückführen kann. Jetzt vereinfachen wir das DL-Problem noch weiter. Wir zeigen, wie die DL-Berechnung in einer zyklischen Gruppe von Primzahlpotenzordnung auf DL-Berechnungen in Gruppen von Primzahlordnung reduziert werden kann.

Sei also $|G| = n = p^e$ für eine Primzahl p. Wir wollen (10.1) in dieser Gruppe lösen. Wir wissen, dass $x < p^e$ gelten muss. Gemäß Theorem 1.3 kann man x in der Form

$$x = x_0 + x_1 p + \ldots + x_{e-1} p^{e-1}, \quad 0 \leq x_i < p, \quad 0 \leq i \leq e-1 \tag{10.2}$$

schreiben. Wir zeigen, dass sich jeder Koeffizient x_i, $0 \leq i \leq e-1$, als Lösung eines DL-Problems in einer Gruppe der Ordnung p bestimmen lässt.

Wir potenzieren die Gleichung $\gamma^x = \alpha$ mit p^{e-1}. Dann ergibt sich

$$\gamma^{p^{e-1}x} = \alpha^{p^{e-1}}. \tag{10.3}$$

Nun gilt nach (10.2)

$$p^{e-1}x = x_0 p^{e-1} + p^e(x_1 + x_2 p + \ldots x_{e-1} p^{e-2}). \tag{10.4}$$

Aus dem kleinen Satz von Fermat (siehe Theorem 2.13), (10.4) und (10.3) erhält man

$$(\gamma^{p^{e-1}})^{x_0} = \alpha^{p^{e-1}}. \tag{10.5}$$

Gemäß (10.5) ist der Koeffizient x_0 Lösung eines DL-Problems in einer Gruppe der Ordnung p, weil $\gamma^{p^{e-1}}$ die Ordnung p hat. Die anderen Koeffizienten bestimmt man rekursiv. Angenommen, $x_0, x_1, \ldots, x_{i-1}$ sind schon bestimmt. Dann gilt

$$\gamma^{x_i p^i + \ldots + x_{e-1} p^{e-1}} = \alpha \gamma^{-(x_0 + x_1 p + \ldots + x_{i-1} p^{i-1})}.$$

Bezeichne das Gruppenelement auf der rechten Seite mit α_i. Potenzieren dieser Gleichung mit p^{e-i-1} liefert

$$(\gamma^{p^{e-1}})^{x_i} = \alpha_i^{p^{e-i-1}}, \quad 0 \leq i \leq e-1. \tag{10.6}$$

Zur Berechnung der Koeffizienten x_i hat man also e DL-Probleme in Gruppen der Ordnung p zu lösen.

Beispiel 10.5 Wie in Beispiel 10.2 lösen wir

$$5^x \equiv 3 \bmod 2017.$$

Die Gruppenordnung der primen Restklassengruppe mod 2017 ist

$$n = 2016 = 2^5 * 3^2 * 7.$$

Zuerst bestimmen wir $x(2) = x \bmod 2^5$. Wir erhalten $x(2)$ als Lösung der Kongruenz

$$(5^{3^2*7})^{x(2)} \equiv 3^{3^2*7} \bmod 2017.$$

Dies ergibt die Kongruenz

$$500^{x(2)} \equiv 913 \bmod 2017.$$

Um diese Kongruenz zu lösen, schreiben wir

$$x(2) = x_0(2) + x_1(2) * 2 + x_2(2) * 2^2 + x_3(2) * 2^3 + x_4(2) * 2^4.$$

Gemäß (10.6) ist $x_0(2)$ die Lösung von

$$2016^{x_0(2)} \equiv 1 \bmod 2017.$$

Man erhält $x_0(2) = 0$ und $\alpha_1 = \alpha_0 = 913 + 2017\mathbb{Z}$. Damit ist $x_1(2)$ Lösung von

$$2016^{x_1(2)} \equiv 2016 \bmod 2017.$$

Dies ergibt $x_1(2) = 1$ und $\alpha_2 = 1579 + 2017\mathbb{Z}$. Damit ist $x_2(2)$ Lösung von

$$2016^{x_2(2)} \equiv 2016 \bmod 2017.$$

Dies ergibt $x_2(2) = 1$ und $\alpha_3 = 1 + 2017\mathbb{Z}$. Damit ist $x_3(2) = x_4(2) = 0$. Insgesamt ergibt sich

$$x(2) = 6.$$

Nun berechnen wir

$$x(3) = x_0(3) + x_1(3) * 3.$$

Wir erhalten $x_0(3)$ als Lösung von

$$294^{x_0(3)} \equiv 294 \bmod 2017.$$

Dies ergibt $x_0(3) = 1$ und $\alpha_1 = 294 + 2017\mathbb{Z}$. Damit ist $x_1(3) = 1$ und

$$x(3) = 4.$$

Schließlich berechnen wir $x(7)$ als Lösung der Kongruenz

$$1879^{x(7)} \equiv 1879 \bmod 2017.$$

Also ist $x(7) = 1$. Wir erhalten dann x als Lösung der simultanen Kongruenz

$$x \equiv 6 \bmod 32, \quad x \equiv 4 \bmod 9, \quad x \equiv 1 \bmod 7.$$

Die Lösung ist $x = 1030$.

10.5.3 Gesamtalgorithmus und Analyse

Um die Gleichung (10.1) zu lösen, geht man im Algorithmus von Pohlig-Hellman folgendermaßen vor. Man berechnet die Werte $\gamma_p = \gamma^{n_p}$ und $\alpha_p = \alpha^{n_p}$ für alle Primteiler p von n. Anschließend werden die Koeffizienten $x_i(p)$ ermittelt für alle Primteiler p von n und $0 \leq i \leq e(p) - 1$. Dazu kann man den Algorithmus von Pollard oder den Babystep-Giantstep-Algorithmus von Shanks benutzen. Zuletzt wird der chinesische Restsatz benutzt, um den diskreten Logarithmus x zu konstruieren.

Der Aufwand, den der Pohlig-Hellman-Algorithmus zur Berechnung diskreter Logarithmen benötigt, kann folgendermaßen abgeschätzt werden.

Theorem 10.3 *Der Pohlig-Hellman-Algorithmus berechnet diskrete Logarithmen der Gruppe G unter Verwendung von $\mathrm{O}(\sum_{p\|G|}(e(p)(\log|G| + \sqrt{p})) + (\log|G|)^2)$ Gruppenoperationen.*

Beweis Wir benutzen dieselben Bezeichnungen wie im vorigen Abschnitt. Die Berechnung einer Ziffer von $x(p)$ für einen Primteiler p von $n = |G|$ erfordert $O(\log n)$ für die Potenzen und $O(\sqrt{p})$ für den Babystep-Giantstep-Algorithmus. Die Anzahl der Ziffern ist höchstens $e(p)$. Daraus folgt die Behauptung. □

Theorem 10.5 zeigt, dass die Zeit, die der Pohlig-Hellman-Algorithmus zur Berechnung diskreter Logarithmen braucht, von der Quadratwurzel des größten Primteilers der Gruppenordnung dominiert wird. Wenn also der größte Primteiler der Gruppenordnung zu klein ist, kann man in der Gruppe leicht diskrete Logarithmen berechnen.

Beispiel 10.6 Die Zahl $p = 2 * 3 * 5^{278} + 1$ ist eine Primzahl. Ihre binäre Länge ist 649. Die Ordnung der primen Restklassengruppe mod p ist $p - 1 = 2 * 3 * 5^{278}$. Die Berechnung diskreter Logarithmen in dieser Gruppe ist sehr einfach, wenn man den Pohlig-Hellman-Algorithmus verwendet, weil der größte Primteiler der Gruppenordnung 5 ist. Darum kann man diese Primzahl p nicht im ElGamal-Verfahren benutzen.

10.6 Index-Calculus

Für die prime Restklassengruppe modulo einer Primzahl und genereller für die multiplikative Gruppe eines endlichen Körpers gibt es effizientere Methoden zur Berechnung diskreter Logarithmen, nämlich sogenannte Index-Calculus-Methoden. Sie sind eng verwandt mit Faktorisierungsverfahren wie dem Quadratischen Sieb und dem Zahlkörpersieb. Wir beschreiben hier die einfachste Version eines solchen Algorithmus.

10.6.1 Idee

Sei p eine Primzahl, g eine Primitivwurzel mod p und $a \in \{1, \ldots, p-1\}$. Wir wollen die Kongruenz

$$g^x \equiv a \bmod p \tag{10.7}$$

lösen. Dazu wählen wir eine Schranke B, bestimmen die Menge

$$F(B) = \{q \in \mathbb{P} : q \leq B\}.$$

Diese Menge ist die *Faktorbasis*. Eine ganze Zahl b heißt B*-glatt*, wenn in der Primfaktorzerlegung von b nur Primzahlen $q \leq B$ vorkommen.

Beispiel 10.7 Sei $B = 15$. Dann ist $F(B) = \{2, 3, 5, 7, 11, 13\}$. Die Zahl 990 ist 15-glatt. Ihre Primfaktorzerlegung ist nämlich $990 = 2 * 3^2 * 5 * 11$.

Wir gehen in zwei Schritten vor. Zuerst berechnen wir die diskreten Logarithmen für alle Faktorbasiselemente. Wir lösen also

$$g^{x(q)} \equiv q \bmod p \tag{10.8}$$

für alle $q \in F(B)$. Dann bestimmen wir einen Exponenten $y \in \{1, 2, \ldots, p-1\}$, für den $ag^y \bmod p$ B-glatt ist. Dann gilt also

$$ag^y \equiv \prod_{q \in F(B)} q^{e(q)} \bmod p \tag{10.9}$$

mit nicht negativen ganzen Exponenten $e(q)$, $q \in F(B)$. Aus (10.8) und (10.9) folgt

$$ag^y \equiv \prod_{q \in F(B)} q^{e(q)} \equiv \prod_{q \in F(B)} g^{x(q)e(q)} \equiv g^{\sum_{q \in F(B)} x(q)e(q)} \bmod p,$$

also

$$a \equiv g^{\sum_{q \in F(B)} x(q)e(q) - y} \bmod p.$$

Daher ist

$$x = \left(\sum_{q \in F(B)} x(q)e(q) - y \right) \bmod (p-1) \tag{10.10}$$

der gesuchte diskrete Logarithmus.

10.6.2 Diskrete Logarithmen der Faktorbasiselemente

Um die diskreten Logarithmen der Faktorbasiselemente zu berechnen, wählt man zufällige Elemente $z \in \{1, \ldots, p-1\}$ und berechnet $g^z \bmod p$. Man prüft, ob diese Zahlen B-glatt

sind. Wenn ja, berechnet man die Zerlegung

$$g^z \bmod p = \prod_{q \in F(B)} q^{f(q,z)}.$$

Jeder Exponentenvektor $(f(q,z))_{q \in F(B)}$ heißt *Relation*.

Beispiel 10.8 Wir wählen $p = 2027$, $g = 2$ und bestimmen Relationen für die Faktorbasis $\{2, 3, 5, 7, 11\}$. Wir erhalten

$$\begin{aligned} 3 * 11 = 33 &\equiv 2^{1593} \bmod 2027 \\ 5 * 7 * 11 = 385 &\equiv 2^{983} \bmod 2027 \\ 2^7 * 11 = 1408 &\equiv 2^{1318} \bmod 2027 \\ 3^2 * 7 = 63 &\equiv 2^{293} \bmod 2027 \\ 2^6 * 5^2 = 1600 &\equiv 2^{1918} \bmod 2027. \end{aligned}$$

Wenn man viele Relationen gefunden hat, kann man für die diskreten Logarithmen folgendermaßen ein lineares Kongruenzensystem aufstellen. Unter Verwendung von (10.8) erhält man

$$g^z \equiv \prod_{q \in F(B)} q^{f(q,z)} \equiv \prod_{q \in F(B)} g^{x(q) f(q,z)} \equiv g^{\sum_{q \in F(B)} x(q) f(q,z)} \bmod p.$$

Daher ist

$$z \equiv \sum_{q \in F(B)} x(q) f(q,z) \bmod (p-1) \tag{10.11}$$

für alle z. Für jede Relation hat man also eine Kongruenz gefunden. Wenn man $n = |F(B)|$ viele Relationen gefunden hat, versucht man die diskreten Logarithmen $x(q)$ durch Verwendung des Gaußalgorithmus zu berechnen und zwar modulo jedes Primteilers l von $p-1$. Teilt ein Primteiler l die Ordnung $p-1$ in höherer Potenz, dann berechnet man die $x(q)$ modulo dieser Potenz. Dadurch wird die lineare Algebra etwas schwieriger. Danach berechnet man die $x(q)$ mit dem chinesischen Restsatz.

Beispiel 10.9 Wir setzen Beispiel 10.8 fort. Der Ansatz

$$q \equiv g^{x(q)} \bmod 2027, \quad q = 2, 3, 5, 7, 11$$

führt mit den Relationen aus Beispiel 10.8 zu dem Kongruenzensystem

$$\begin{aligned} x(3) + x(11) &\equiv 1593 \bmod 2026 \\ x(5) + x(7) + x(11) &\equiv 983 \bmod 2026 \\ 7x(2) + x(11) &\equiv 1318 \bmod 2026 \end{aligned} \tag{10.12}$$

$$2x(3) + x(7) \equiv 293 \bmod 2026$$
$$6x(2) + 2x(5) \equiv 1918 \bmod 2026.$$

Da $2026 = 2 * 1013$ ist, und 1013 eine Primzahl ist, lösen wir jetzt das Kongruenzensystem mod 2 und mod 1013. Es ergibt sich

$$\begin{aligned} x(3) + x(11) &\equiv 1 \bmod 2 \\ x(5) + x(7) + x(11) &\equiv 1 \bmod 2 \\ x(2) + x(11) &\equiv 0 \bmod 2 \\ x(7) &\equiv 1 \bmod 2. \end{aligned} \tag{10.13}$$

Wir wissen bereits, dass $x(2) = 1$ ist, weil als Primitivwurzel $g = 2$ gewählt wurde. Daraus gewinnt man

$$x(2) \equiv x(5) \equiv x(7) \equiv x(11) \equiv 1 \bmod 2, \quad x(3) \equiv 0 \bmod 2. \tag{10.14}$$

Als nächstes berechnen wir die diskreten Logarithmen der Faktorbasiselemente mod 1013. Wieder ist $x(2) = 1$. Aus (10.12) erhalten wir

$$\begin{aligned} x(3) + x(11) &\equiv 580 \bmod 1013 \\ x(5) + x(7) + x(11) &\equiv 983 \bmod 1013 \\ x(11) &\equiv 298 \bmod 1013 \\ 2x(3) + x(7) &\equiv 293 \bmod 1013 \\ 2x(5) &\equiv 899 \bmod 1013. \end{aligned} \tag{10.15}$$

Wir erhalten $x(11) \equiv 298 \bmod 1013$. Um $x(5)$ auszurechnen, müssen wir 2 mod 1013 invertieren. Wir erhalten $2 * 507 \equiv 1 \bmod 1013$. Daraus ergibt sich $x(5) \equiv 956 \bmod 1013$. Aus der zweiten Kongruenz erhalten wir $x(7) \equiv 742 \bmod 1013$. Aus der ersten Kongruenz erhalten wir $x(3) \equiv 282 \bmod 1013$. Unter Berücksichtigung von (10.14) erhalten wir schließlich

$$x(2) = 1, x(3) = 282, x(5) = 1969, x(7) = 1755, x(11) = 1311.$$

Man verifiziert leicht, dass dies korrekt ist.

10.6.3 Individuelle Logarithmen

Sind die diskreten Logarithmen der Faktorbasiselemente berechnet, bestimmt man den diskreten Logarithmus von a zur Basis g, indem man ein $y \in \{1, \ldots, p-1\}$ zufällig bestimmt. Wenn $ag^y \bmod p$ B-glatt ist, wendet man (10.10) an. Andernfalls wählt man ein neues y.

Beispiel 10.10 Wir lösen

$$2^x \equiv 13 \bmod 2027.$$

Wir wählen $y \in \{1, \ldots, 2026\}$ zufällig, bis $13 * 2^y \bmod 2027$ nur noch Primfaktoren aus der Menge $\{2, 3, 5, 7, 11\}$ hat. Wir finden

$$2 * 5 * 11 = 110 \equiv 13 * 2^{1397} \bmod 2027.$$

Unter Verwendung von (10.10) ergibt sich $x = (1 + 1969 + 1311 - 1397) \bmod 2026 = 1884$.

10.6.4 Analyse

Man kann zeigen, dass der beschriebene Index-Calculus-Algorithmus die subexponentielle Laufzeit $L_p[1/2, c + o(1)]$ hat, wobei c eine Konstante ist, die von der technischen Umsetzung des Algorithmus abhängt. Die Analyse wird ähnlich durchgeführt wie die Analyse des quadratischen Siebs in Abschn. 9.4.

10.7 Andere Algorithmen

Es gibt eine Reihe effizienterer Varianten des Index-Calculus-Algorithmus. Der zur Zeit effizienteste Algorithmus ist das Zahlkörpersieb. Es hat die Laufzeit $L_p[1/3, (64/9)^{1/3}]$. Das Zahlkörpersieb zur DL-Berechnung wurde kurz nach der Erfindung des Zahlkörpersiebs zur Faktorisierung entdeckt. Rekordberechnungen von diskreten Logarithmen in endlichen Körpern findet man unter [26]. So wurde am 11. Juni 2014 ein diskreter Logarithmus modulo einer 180-stelligen Primzahl berechnet.

Alle DL-Probleme, die im Kontext der Kryptographie von Bedeutung sind, sind auf Quantencomputern leicht lösbar. Das wurde von Shor in [69] gezeigt. Es ist nur noch nicht klar, ob und wann es Quantencomputer geben wird.

10.8 Verallgemeinerung des Index-Calculus-Verfahrens

Das Index-Calculus-Verfahren ist nur für die prime Restklassengruppe modulo einer Primzahl erklärt worden. Die Verfahren von Shanks, Pollard oder Pohlig-Hellman funktionieren aber in beliebigen endlichen Gruppen. Auch das Index-Calculus-Verfahren kann verallgemeinert werden. In beliebigen Gruppen braucht man eine Faktorbasis von Gruppenelementen. Man muss zwischen den Gruppenelementen genügend Relationen finden, also Potenzprodukte, deren Wert die Eins in der Gruppe ist. Hat man die Relationen gefunden, kann man mit linearer Algebra die diskreten Logarithmen genauso berechnen,

wie das oben beschrieben wurde. Die entscheidende Schwierigkeit ist die Bestimmung der Relationen. In primen Restklassengruppen kann man Relationen finden, weil man die Gruppenelemente in den Ring der ganzen Zahlen liften kann und dort die eindeutige Primfaktorzerlegung gilt. Für andere Gruppen wie z. B. die Punktgruppe elliptischer Kurven ist nicht bekannt, wie die Relationen gefunden werden können und daher ist in diesen Gruppen auch das Index-Calculus-Verfahren bis jetzt nicht anwendbar.

10.9 Übungen

Übung 10.1 Lösen Sie $3^x \equiv 693 \bmod 1823$ mit dem Babystep-Giantstep-Algorithmus.

Übung 10.2 Verwenden Sie den Babystep-Giantstep-Algorithmus, um den diskreten Logarithmus von 15 zur Basis 2 mod 239 zu berechnen.

Übung 10.3 Lösen Sie $a^x \equiv 507 \bmod 1117$ für die kleinste Primitivwurzel a mod 1117 mit dem Pohlig-Hellman-Algorithmus.

Übung 10.4 Verwenden Sie den Pohlig-Hellman-Algorithmus, um den diskreten Logarithmus von 2 zur Basis 3 mod 65537 zu berechnen.

Übung 10.5 Berechnen Sie mit dem Pollard-ρ-Algorithmus die Lösung von $g^x \equiv 15 \bmod 3167$ für die kleinste Primitivwurzel g mod 3167.

Übung 10.6 Verwenden Sie die Variante des Pollard-ρ-Algorithmus, die acht Tripel (β, x, y) speichert, um das DL-Problem $g^x \equiv 15 \bmod 3167$ für die kleinste Primitivwurzel g mod 3167 zu berechnen. Vergleichen Sie die Effizienz dieser Berechnung mit dem Ergebnis von Übung 10.5.

Übung 10.7 Berechnen Sie mit dem Index-Calculus-Algorithmus unter Verwendung der Faktorbasis $\{2, 3, 5, 7, 11\}$ die Lösung von $7^x \equiv 13 \bmod 2039$.