4.6 Berechnung des diskreten Logarithmus

Der klassische Algorithmus zur Berechnung des diskreten Logarithmus ist der **Index-Calculus** von ADLEMAN – „Index“ war übrigens die Bezeichnung, die GAUSS für den diskreten Logarithmus verwendet hatte.

Sie $p \geq 3$ eine Primzahl und a eine Primitivwurzel für p.

Vorbereitung

Dieser Schritt muss zu gegebenen p und a nur einmal ausgeführt werden.

Seien $p_1 = 2$, $p_2 = 3$, …, p_k die ersten k Primzahlen – wie k gewählt wird, wird noch festgelegt.

Für einen zufällig gewählten Exponenten r könnte es sein, dass $a^r \bmod p$ – als ganze Zahl $\in \mathbb{Z}$ betrachtet – nur Primfaktoren in $\{p_1, \ldots, p_k\}$ hat. Nach h solchen Glücksfällen hat man in $\mathbb{Z}$, also erst recht in $\mathbb{F}_p$, ein System von h Gleichungen

$$\begin{aligned} a^{r_1} \bmod p &= p_1^{\alpha_{11}} \cdots p_k^{\alpha_{1k}}, \\ &\vdots \\ a^{r_h} \bmod p &= p_1^{\alpha_{h1}} \cdots p_k^{\alpha_{hk}}. \end{aligned}$$

Daraus entsteht ein lineares Gleichungssystem für die k unbekannten Größen $\log_a p_i$ über dem Ring $\mathbb{Z}/(p-1)\mathbb{Z}$:

$$\begin{aligned} r_1 &= \alpha_{11} \cdot \log_a p_1 + \cdots + \alpha_{1k} \cdot \log_a p_k, \\ &\vdots \\ r_h &= \alpha_{h1} \cdot \log_a p_1 + \cdots + \alpha_{hk} \cdot \log_a p_k. \end{aligned}$$

Zu dessen Auflösung gibt es, wie wir aus Kapitel I wissen, effiziente Algorithmen; ist h groß genug – mindestens $h \geq k$ –, lassen sich $\log_a p_1$, …, $\log_a p_k$ bestimmen.

Die Suche nach den „Glücksfällen“ ergibt also einen probabilistischen Algorithmus.

Berechnung

Sei $y \in \mathbb{F}_p^\times$ gegeben; gesucht ist $\log_a y$.

Für einen zufällig gewählten Exponenten s könnte in $\mathbb{Z}$

$$y \cdot a^s \bmod p = p_1^{\beta_1} \cdots p_k^{\beta_k}$$

sein. Dann ist

$$\log_a y = \beta_1 \cdot \log_a p_1 + \cdots + \beta_k \cdot \log_a p_k - s$$

ganz leicht berechnet.

Die Berechnung des diskreten Logarithmus für beliebige Elemente ist daher auf die Berechnung für die Elemente der **Faktorbasis** $(p_1, \ldots, p_k)$ *reduziert.* Auch diese Reduktion ist probabilistisch.

Varianten

Aus dem vorgestellten Ansatz werden verschiedene Algorithmen konstruiert, die unterschiedlich schnell sind. Sie unterscheiden sich in der Wahl der Faktorbasis – diese kann an y adaptiert sein und muss nicht aus der lückenlosen Folge der ersten Primzahlen bestehen – sowie in der Strategie für die Wahl der Exponenten r und s.

Die schnellste Variante verwendet ein Zahlkörpersieb, wie es auch bei der Faktorisierung großer Zahlen eingesetzt wird, und kommt auf einen Aufwand

$$\approx e^{c\cdot \sqrt[3]{\log p \cdot (\log\log p)^2}},$$

wie er auch bei der Faktorisierung einer gleichlangen zusammengesetzten Zahl benötigt wird. Nach dem aktuellen Stand sind also 1024-Bit-Primzahlmoduln nur ganz kurzfristig als sicher für kryptographische Anwendungen anzusehen, mittelfristig braucht man mindestens 2048 Bit.

Als Kuriosum sei angemerkt, dass das „Secure NFS“ von SUN noch in den 90er Jahren eine Primzahl der Länge 192 Bits (58 Dezimalstellen) verwendete.

Spezielle Primzahlen

Es gibt Argumente, p speziell, also als die größere Zahl eines Germain-Paares zu wählen – also $p = 2q + 1$ mit q prim:

1. Es gibt Algorithmen, die besonders schnell sind, wenn $p-1$ nur kleine Primfaktoren hat. Dieses Argument wird heute nicht mehr für stichhaltig angesehen, da die Chance, aus Versehen eine soche „schlechte“ Primzahl zu erwischen, extrem gering ist; außerdem ist das Zahlkörpersieb so schnell, dass die speziellen Algorithmen kaum noch einen Vorteil bringen.

2. Die Bestimmung eines primitiven Elements ist sehr viel leichter, siehe Abschnitt A.10 im Anhang.