4.6 Computing Discrete Logarithms

The classical algorithm for computing discrete logarithms is the **index calculus** by Adleman—"index" was Gauss' denotation of the discrete logarithm.

Let $p \geq 3$ be a prime and a be a primitive element for p.

The naive algorithm for computing $\log_a y$ for $y \in \mathbb{F}_p^\times$ is punished by exponentially growing costs, as usual. It computes $a, a^2, a^3, \ldots$ in order until x with $a^x = y$ is found. In the mean it needs $\frac{p}{2} - 1$ trials, in the worst case, $p - 2$ (omitting the trivial value $y = 1$).

Preliminary Steps

For given p and a we need to execute this precomputation only once.

Let $p_1 = 2$, $p_2 = 3$, …, p_k be the first k primes.

If we randomly choose an exponent r, then it could happen that $a^r \bmod p$—considered as integer $\in \mathbb{Z}$—has only prime divisors in $\{p_1, \ldots, p_k\}$. After h strokes of luck we have a system of h equations:

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

in $\mathbb{Z}$ and a forteriori in $\mathbb{F}_p$. Taking logarithms results in a system of linear equations over the ring $\mathbb{Z}/(p-1)\mathbb{Z}$ for the k unknowns $\log_a p_i$:

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

From Chapter I we know efficient algorithms for solving it. If h is sufficiently large—at least $h \geq k$—, then we can compute $\log_a p_1$, …, $\log_a p_k$.

The random search for "strokes of luck" makes the precomputation probabilistic.

Computation

Let $y \in \mathbb{F}_p^\times$ be given. We want to compute $\log_a y$.

For a randomly chosen exponent s it could happen that

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

in $\mathbb{Z}$. Then we easily compute

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

This observation reduces the computation of the discrete logarithm of any element to the computation for the elements of the **factor basis** $(p_1, \ldots, p_k)$. This reduction is also probabilistic.

Variants

The presented approach has several variants that result in different running times. They vary in the choice of the factor basis—that might be adapted to y and need not consist of the first primes without gap—and in the strategy of choosing the exponents r and s.

The fastest known variant uses a number field sieve such as applied for factoring large integers and has expenses of

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

the same order of magnitude as is needed for factoring an integer of the same size. By the state of the art 1024-bit primes are insecure, and 2048-bit primes secure only for short-term cryptographic applications.

As an oddity we mention that the "Secure NFS" protocol deployed by SUN used a 192-bit prime (58 decimal places) even in the 1990s.

Special Primes

There are reasons to choose p as a special prime of the form $p = 2p' + 1$ with p' prime:

1. Some algorithms are very fast if $p - 1$ has only small prime divisors. This argument is no longer considered as solid since the advantage of special algorithms over the current versions of the number field sieve is only small. Moreover the probability of choosing such a "bad" prime by accident is extremely small.

2. Finding a primitive element is easy, see Section A.9 in the appendix.