18.783 Elliptic Curves
Lecture 9

Andrew Sutherland

October 2, 2025

The discrete logarithm problem

Definition

Let G be a finite group. For $\alpha \in G$ and $\beta \in \langle\alpha\rangle$ we define

$$\log_\alpha \beta := \min\{x \in \mathbb{Z}_{\geq 1} : \alpha^x = \beta\}.$$

The discrete logarithm problem (DLP) is to compute $\log_\alpha \beta$ given $\alpha, \beta \in \langle\alpha\rangle$.
More generally, we can ask, given $\alpha, \beta \in G$, compute $\log_\alpha \beta$ or determine $\beta \notin \langle\alpha\rangle$.

DLP-based cryptography relies on the assumption that this is a hard problem.

Example

In $G = \mathbb{F}_{101}^\times$ we have $\log_3 37 = 24$ since $3^{24} \equiv 37 \bmod 101$.
In $G = \mathbb{F}_{101}^+$ we have $\log_3 37 = 46$ since $46 \cdot 3 \equiv 37 \bmod 101$.

One of these problems can be solved in quasilinear time!
The other one can be solved in subexponential (in some cases quasipolynomial) time!

Some generalizations

Definition

For $\alpha, \beta \in G$ we define

$$\operatorname{ord}_\alpha \beta := \min\{y \in \mathbb{Z}_{\geq 1} : \beta^y \in \langle \alpha \rangle\}.$$

The extended discrete logarithm problem is to compute the pair (x, y) where $x = \log_\alpha \beta^y$ and $y = \operatorname{ord}_\alpha \beta$, given $\alpha, \beta \in G$.

Definition

Given $\alpha_1, \ldots, \alpha_r \in G$ and $n_1, \ldots, n_r \in \mathbb{Z}$ such that every $\beta \in G$ has a unique representation as

$$\beta = \alpha_1^{e_1} \cdots \alpha_r^{e_r} \qquad (1 \leq e_i \leq n_i),$$

the vector discrete logarithm problem is to compute $(e_1, \ldots, e_r)$ for a given $\beta \in G$.

The discrete logarithm problem in a cyclic group

We will focus on the original DLP, with $\beta \in \langle \alpha \rangle$. WLOG we may assume $G = \langle \alpha \rangle$.

Let $N = \#G$ be the order of the cyclic group G. We have an isomorphism

$$\begin{aligned} G &\xrightarrow{\sim} \mathbb{Z}/N\mathbb{Z} \\ \beta &\to \log_\alpha \beta \\ \alpha^x &\leftarrow x \end{aligned}$$

The Euclidean algorithm solves the DLP in $\mathbb{Z}/N\mathbb{Z}$ in quasilinear time.

DLP-based cryptography is based on the assumption that there is no way to compute the isomorphism $G \simeq \mathbb{Z}/N\mathbb{Z}$ without solving the DLP in G (note that the Euclidean algorithm does not work in the additive group $\mathbb{Z}/N\mathbb{Z}$, it uses ring operations in $\mathbb{Z}$).

To analyze the difficulty of the DLP in general (and the efficiency of solutions), we will use a computational model that forces algorithms to work entirely in G.

Generic group algorithms

Definition

A generic group algorithm (or just a generic algorithm) is one that interacts with an abstract group G solely through a black box (sometimes called an oracle).

Group elements are opaquely encoded as bit-strings via a map $\mathrm{id}\colon G \to \{0,1\}^m$ chosen by the black box. The black box supports the following operations.

- `identity`: output $\mathrm{id}(1_G)$.
- `inverse`: given input $\mathrm{id}(\alpha)$, output $\mathrm{id}(\alpha^{-1})$.
- `compose`: given inputs $\mathrm{id}(\alpha)$ and $\mathrm{id}(\beta)$, output $\mathrm{id}(\alpha\beta)$.
- `random`: output $\mathrm{id}(\alpha)$ for a uniformly distributed random element $\alpha \in G$.

In the description above we used multiplicative notation; in additive notation the outputs would be $\mathrm{id}(0_G)$, $\mathrm{id}(-\alpha)$, $\mathrm{id}(\alpha+\beta)$, respectively.

Generic algorithms for DLP

Example (Linear search)

Compute $\alpha, 2\alpha, 3\alpha, \ldots, x\alpha = \beta$. This uses $O(N)$ group operations.

Example (Baby-steps giant-steps)

Pick $r, s \in \mathbb{Z}_{\geq 1}$ with $rs > N$ and compute

$$\textbf{baby-steps}: 0, \alpha, 2\alpha, \ldots, i\alpha, \ldots (r-1)\alpha,$$
$$\textbf{giant-steps}: \beta, \beta - r\alpha, \beta - 2r\alpha, \ldots, \beta - jr\alpha, \ldots \beta - (s-1)r\alpha.$$

A collision between the ith baby-step and the jth giant-step yields the relation

$$i\alpha = \beta - jr\alpha$$

with $0 \leq i < r$ and $0 \leq j < s$. If $i = j = 0$ then $\log_\alpha \beta = N$, else $\log_\alpha \beta = i + jr$.

For $r \approx s$ this uses $O(\sqrt{N})$ group operations.

The Pohlig-Hellman algorithm

Suppose $N = N_1 N_2$ with $N_1 \perp N_2$. Then $\mathbb{Z}/N\mathbb{Z} \simeq \mathbb{Z}/N_1\mathbb{Z} \oplus \mathbb{Z}/N_2\mathbb{Z}$ and we have

$$x \mapsto (x \bmod N_1, x \bmod N_2)$$
$$(M_1 x_1 + M_2 x_2) \bmod N \leftarrow (x_1, x_2)$$

where

$$M_1 = N_2(N_2^{-1} \bmod N_1) \equiv \begin{cases} 1 \bmod N_1, \\ 0 \bmod N_2, \end{cases}$$

$$M_2 = N_1(N_1^{-1} \bmod N_2) \equiv \begin{cases} 0 \bmod N_1, \\ 1 \bmod N_2. \end{cases}$$

Note that computing M_1 and M_2 involves no group operations.
It thus costs nothing in our computational model which only counts group operations, but its bit complexity is quasilinear in any case (so it is indeed negligible).

The Pohlig-Hellman algorithm

Let $N = N_1 N_2$ with $N_1 \perp N_2$, define M_1, M_2 as above, and let

$$x_1 := x \bmod N_1 \qquad \text{and} \qquad x_2 := x \bmod N_2,$$

so that $x = M_1 x_1 + M_2 x_2$, and $\beta = (M_1 x_1 + M_2 x_2)\alpha$. We then have

$$N_2 \beta = M_1 x_1 N_2 \alpha + M_2 x_2 N_2 \alpha.$$

The order of $N_2\alpha$ is N_1 (since $N_1 \perp N_2$), and $M_1 \equiv 1 \bmod N_1$, $M_2 \equiv 0 \bmod N_1$ yield

$$N_2 \beta = x_1 N_2 \alpha.$$

We similarly find that $N_1 \beta = x_2 N_1 \alpha$, and therefore

$$x_1 = \log_{N_2\alpha} N_2 \beta, \qquad x_2 = \log_{N_1\alpha} N_1 \beta.$$

If we know x_1 and x_2 then we can compute $x = (M_1 x_1 + M_2 x_2) \bmod N$.

The Pohlig-Hellman algorithm

Applying $N = N_1 N_2$ with $N_1 \perp N_2$ recursively reduces to the case where $N = p^e$ is a prime power using $O(n \log n)$ group operations, where $n = \log N$.

Let $e_0 = \lceil e/2 \rceil$, $e_1 = \lfloor e/2 \rfloor$, and write $x = \log_\alpha \beta$ as $x = x_0 + p^{e_0} x_1$, where we have $0 \le x_0 < p^{e_0}$ and $0 \le x_1 < p^{e_1}$. Then

$$\begin{aligned} \beta &= (x_0 + p^{e_0} x_1)\alpha \\ p^{e_1}\beta &= x_0 p^{e_1}\alpha + x_1 p^e \alpha \\ x_0 &= \log_{p^{e_1}\alpha} p^{e_1}\beta. \end{aligned}$$

We also have $\beta - x_0\alpha = p^{e_0} x_1 \alpha$, and therefore

$$x_1 = \log_{p^{e_0}\alpha}(\beta - x_0\alpha).$$

If N is not prime, this again reduces the computation of $\log_\alpha \beta$ to the computation of two smaller discrete logarithms (of roughly equal size) using $O(n)$ group operations.

The Pohlig-Hellman algorithm

If we use the baby-steps giant-steps algorithm to solve the prime order cases we obtain a total complexity of

$$O\Big(n \log n + \sum e_i \sqrt{p_i}\Big)$$

group operations, where $N = p_1^{e_1} \cdots p_r^{e_r}$ and $n = \log N$.

If p is the largest prime factor of N this simplifies to

$$O\Big(n \log n + n\sqrt{p}\Big)$$

group operations. If $p = O(n^k)$ for some k, this is a polynomial-time generic algorithm.

The Pollard-rho algorithm

Let us view $G = \langle \alpha \rangle$ as the vertex set V of a connected graph Γ with edges $e_{ij} = (\gamma_i, \gamma_j)$ labeled by $\delta_{ij} = \gamma_j - \gamma_i$ so that $\gamma_i + \delta_{ij} = \gamma_j$ (this is the Cayley graph).

If we can write each δ_{ij} as a linear combination of α and β then any cycle in this graph gives a linear relation between α and β that we can use to compute $\log_\alpha \beta$ (provided the coefficients of this relation are invertible modulo N).

Consider a random walk in Γ starting at $v_0 \in V$ defined by a function $f\colon V \to V$:

$$
\begin{aligned}
v_1 &= f(v_0) \\
v_2 &= f(v_1) \\
v_3 &= f(v_2) \\
&\vdots
\end{aligned}
$$

Eventually we will repeat a vertex $v_\rho = v_\lambda$ with $\rho > \lambda$ and then enter an infinite cycle.

The Pollard-rho algorithm

Theorem

Let V be a finite set. For any $v_0 \in V$, the expected value of ρ for a walk from v_0 defined by a random function $f : V \to V$ is

$$\mathrm{E}[\rho] \sim \sqrt{\pi N/2},$$

as $\#V = N \to \infty$. We also have $\mathrm{E}[\lambda] = \mathrm{E}[\sigma] = \frac{1}{2}\,\mathrm{E}[\rho] = \sqrt{\pi N/8}$, where $\sigma = \rho - \lambda$.

The Pollard-rho algorithm

Fix $r \approx 20$. Let $h\colon G \to \{1, \ldots, r\}$ be a random function (a hash function), pick r random pairs $(c_i, d_i) \in \mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z}$, define $\delta_i := c_i\alpha + d_i\beta$, and define

$$
\begin{aligned}
f\colon \mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z} \times G &\longrightarrow \mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z} \times G \\
(a, b, \gamma) &\mapsto (a + c_i, b + d_i, \gamma + \delta_i) \qquad \text{(where } i = h(\gamma)\text{)}.
\end{aligned}
$$

In practice we don't pick h at random, we could use $\mathrm{id}(\gamma) \bmod r$, for example.

We can now use f to define an r-adding walk, starting from some $\gamma_0 = a_0\alpha + b_0\beta$ with $a_0, b_0 \in \mathbb{Z}/N\mathbb{Z}$ chosen at random.

Note that if $(a_{j+1}, b_{j+1}, \gamma_{j+1}) = f(a_j, b_j, \gamma_j)$, the value of γ_{j+1} depends only on γ_j, not on a_j or b_j, so this defines a random walk on V.

The Pollard-rho algorithm

Algorithm (Pollard-ρ)

Given α, $N = |\alpha|$, $\beta \in \langle\alpha\rangle$, compute $\log_\alpha \beta$ as follows:

1. Compute $\delta_i = c_i\alpha + d_i\beta$ for $r \approx 20$ randomly chosen pairs $c_i, d_i \in \mathbb{Z}/N\mathbb{Z}$.
2. Compute $\gamma_0 = a_0\alpha + b_0\beta$ for randomly chosen $a_0, b_0 \in \mathbb{Z}/N\mathbb{Z}$.
3. Compute $(a_j, b_j, \gamma_j) = f(a_{j-1}, b_{j-1}, \gamma_{j-1})$ for $j \geq 1$ until $\gamma_k = \gamma_j$ with $k > j$.
4. $\gamma_k = \gamma_j$ implies $a_j\alpha + b_j\beta = a_k\alpha + b_k\beta$. Provided that $b_k - b_j$ is invertible in $\mathbb{Z}/N\mathbb{Z}$, we return $\log_\alpha \beta = \frac{a_j - a_k}{b_k - b_j} \in \mathbb{Z}/N\mathbb{Z}$; otherwise start over at step 1.

This algorithm terminates with probability 1 and its output is always correct. It is a Las Vegas algorithm with expected running time $O(\sqrt{N})$.

As written it uses $O(\sqrt{N})$ space, because we have to store all the γ_j to detect $\gamma_k = \gamma_j$.

Floyd's cycle detection method (aka the tortoise and the hare)

We now modify Step 3 of the algorithm to compute

$$(a_j, b_j, \gamma_j) = f(a_{j-1}, b_{j-1}, \gamma_{j-1})$$
$$(a_k, b_k, \gamma_k) = f(f(a_{k-1}, b_{k-1}, \gamma_{k-1})).$$

The triple (a_j, b_j, γ_j) is the tortoise, and the triple (a_k, b_k, γ_k) is the hare.

Once the tortoise enters the cycle, the hare (already in the cycle) will collide with the tortoise within $\sigma = \rho - \lambda$ iterations.

The expected number of iterations is $\mathrm{E}[\lambda + \sigma/2] = 3/4\ \mathrm{E}[\rho]$, but each iteration uses 3 group operations, making the algorithm slower by a factor of $9/4$. Still, this achieves a time complexity of $O(\sqrt{N})$ group operations while storing just $O(1)$ group elements.

The $9/4$ can be reduced to $1 + o(1)$ using distinguished points (see notes for details).

A generic lower bound

Theorem (Shoup 1997)

Let $G = \langle \alpha \rangle$ be a group of order N. Let $\mathcal{B}$ be a black box for G using a random identification map id: $G \hookrightarrow \{0,1\}^m$. Let $\mathcal{A}\colon \{0,1\}^m \times \{0,1\}^m \to \mathbb{Z}/N\mathbb{Z}$ be a randomized generic group algorithm that makes at most $s - 4\lceil \lg N \rceil$ calls to $\mathcal{B}$, for some integer s, and let x denote a random element of $\mathbb{Z}/N\mathbb{Z}$. Then

$$\Pr_{x,\mathit{id},\tau}[\mathcal{A}(\mathit{id}(\alpha), \mathit{id}(x\alpha)) = x] < \frac{s^2}{2p},$$

where τ denotes random coin-flips made by $\mathcal{A}$ and p is the largest prime factor of N.

Corollary

Let G be a cyclic group of prime order N. Every generic Las Vegas algorithm for the discrete logarithm problem in G uses an expected $\Omega(\sqrt{N})$ group operations.