

1 The Discrete Logarithm Problem

1.1 The Problem

Let G be a cyclic group of order n and $g \in G$ a generator, i.e. $\langle g \rangle = G$. The Discrete Logarithm Problem is the following. Given $h \in G$, we want to find some $m \in \mathbb{Z}/n\mathbb{Z}$ such that $h = g^m$.

We currently have no good algorithm for solving the Discrete Log Problem (henceforth called the DLP), but we can reduce the number of steps it takes from the order n taken by the stupid algorithm of just computing every power of g until we find the one desired. For example, for a group of non-prime order n , if we can write $n = n_1 n_2$ with $(n_1, n_2) = 1$, then we know that $G \cong G_1 \times G_2$, where $|G_i| = n_i$. The group isomorphism is given by

$$\begin{aligned} G &\cong G_1 \times G_2 \\ x &\rightarrow (x^{n_2}, x^{n_1}) \\ y^a z^b &\leftarrow (y, z), \quad an_2 + bn_1 = 1. \end{aligned}$$

So the DLP in a group of order n reduces to the DLP on the groups of order n_1 and n_2 , respectively.

For a general group, the best algorithm for the DLP takes $O(\sqrt{n})$ steps. Notice that $\sqrt{n} = e^{\frac{\ln n}{2}}$, so the algorithm is exponential in $\ln n$. In \mathbb{F}_q^* , there is an algorithm taking $O(e^{(\ln n)^{1/3+\epsilon}})$ steps, known as the “index calculus algorithm”, which is probabilistic in nature. This algorithm performs better than exponential time, but still not as good as an ideal polynomial-time algorithm:

$$(\log n)^c < e^{(\log n)^\alpha} < n^c$$

There is currently no polynomial time algorithm for solving the DLP.

1.2 An Algorithm in $O(\sqrt{n})$ (Shank’s “Baby-step/Giant-step” Algorithm)

As above, let G be a cyclic group of order n with generator g . The following is a deterministic algorithm that, given $h \in G$, computes m such that $h = g^m$ in $O(\sqrt{n})$ time.

- (Baby Steps) Compute and store the values $1, g, g^2, g^3, \dots, g^{\lfloor \sqrt{n} \rfloor}$
- (Giant Steps) Given $h \in G$, compute $hg^{-i\lfloor \sqrt{n} \rfloor}$ for $i = 1, 2, \dots$, and compare it with the list $1, g, \dots, g^{\lfloor \sqrt{n} \rfloor}$, until you find a match. If a match is found, stop, else proceed to next value of i .

Once you find a match, you’ve just found i, j such that

$$hg^{-i\lfloor \sqrt{n} \rfloor} = g^j \implies h = g^{i\lfloor \sqrt{n} \rfloor + j}$$

Theorem 1.1. *There is a match with $0 < i \leq \lfloor \sqrt{n} \rfloor + 1$. Therefore, the algorithm above stops in $O(\sqrt{n})$ steps.*

Proof. Choose $h \in G$. Then since G is cyclic with generator g , $h = g^m$ for some m . Use the division algorithm to write

$$m = i\lfloor\sqrt{n}\rfloor + j, \quad 0 \leq j \leq \lfloor\sqrt{n}\rfloor - 1$$

Then we have

$$i = \frac{m-j}{\lfloor\sqrt{n}\rfloor} \leq \frac{m}{\lfloor\sqrt{n}\rfloor} \leq \frac{n-1}{\lfloor\sqrt{n}\rfloor} \leq \frac{n-1}{\sqrt{n}-1} = \sqrt{n} + 1$$

and it follows that $i \leq \lfloor\sqrt{n}\rfloor + 1$. □

1.3 The Index Calculus Algorithm “Framework”

Suppose that you have the following.

- G a cyclic group of order n .
- g a generator of G
- $B = \{g_1, g_2, \dots, g_r\} \subseteq G$ a “factor base” for G , and
- An algorithm (**) that, given $h \in G$, outputs with a certain probability $\epsilon > 0$ integers a_1, \dots, a_r with $h = g_1^{a_1} \cdots g_r^{a_r}$.

Then the Index Calculus Algorithm performs the following computations. First, we must find $\{x_1, \dots, x_r\}$ such that $g^{x_i} = g_i$. In order to do this, pick $k \in \mathbb{Z}/n\mathbb{Z}$ at random. Compute g^k and feed it to our algorithm (**) above. Repeat enough times to get a system of equalities

$$g^{k_j} = g_1^{a_{j1}} \cdots g_r^{a_{jr}}, \quad j = 1, \dots, R, \quad R > r.$$

This gives us the system of R linear equations in the x_i ’s:

$$k_j \equiv a_{j1}x_1 + \cdots + a_{jr}x_r \pmod{n}$$

If we have found r linearly independent equations, then we can solve for the x_i . We hope to succeed if R is a little bit bigger than r .

Now we are ready to compute the discrete log. Given $h \in G$, pick $k \in \mathbb{Z}/n\mathbb{Z}$ at random, compute hg^{-k} , and feed it to the algorithm (**) above, until you get $hg^{-k} = g_1^{a_1} \cdots g_r^{a_r}$. Then

$$h = g^m, \quad m = k + a_1x_1 + \cdots + a_rx_r.$$

Next time, we will work on finding the specific algorithm (**).

2 Diffie-Hellman Key Exchange

The Diffie-Hellman Key Exchange is an encoding scheme that relies on the difficulty of the DLP. If we could find a faster algorithm for computing the DLP, then this scheme would become drastically less effective.

2.1 The Algorithm

Let $G = \langle g \rangle$ be a cyclic group of order n . Suppose Alice and Bob want to communicate over an insecure channel. The Diffie-Hellman algorithm proceeds as follows:

- Alice chooses at random some $a \in \mathbb{Z}/n\mathbb{Z}$, keeps a a secret and sends Bob g^a .
- Bob chooses at random some $b \in \mathbb{Z}/n\mathbb{Z}$, keeps b a secret and sends Alice g^b .
- Alice can compute g^{ab} by computing $(g^b)^a$
- Bob can compute g^{ab} by computing $(g^a)^b$. Alice and Bob now have a shared secret key.

Notice that if the discrete logarithm problem is hard, then knowledge of g^a, g^b doesn't reveal the numbers a and b . Thus Alice and Bob's secret is safe.

2.2 Open Problem

One other way to crack the Diffie-Hellman code would be to find a way to compute g^{ab} knowing only g^a and g^b . i.e. without first computing a and b . An algorithm to compute this would be interesting even if it ran in $O(e^{(\log n)^\alpha})$ time, $\alpha < 1$.

3 More on the Index Calculus Algorithm

3.1 Running Time of the Index Calculus Algorithm

The average number of tries in the algorithm (***) before success is $\frac{1}{\epsilon}$. We need a total of R successes before we attempt to calculate the discrete logarithms of the g_i , so it follows that we must run (***) a total of $\frac{R}{\epsilon}$ times on average before success. This step is parallelizable, so we can improve the time it takes to run this algorithm by running on several machines simultaneously.

After finding our R linear equations, we must then attempt to solve them with linear algebra, however, which is not in general a parallelizable algorithm. The time it takes to perform this step is $O(R^3)$. For our purposes, we may assume that $R \sim r$ for studying the running time.

3.2 An Example

Let $G = \mathbb{F}_q^*$, $q = p^n$, p a prime (think p small and n large). [NOTE: We have changed notation - the order of our group is no longer n , but $p^n - 1$.] Then

$$\mathbb{F}_q = \mathbb{F}_p[x]/(f(x)), \quad \deg(f) = n.$$

Given an $h \in \mathbb{F}_q^*$, we can view h as the reduction of $h(x) \in \mathbb{F}_p[x]$, with $\deg(h(x)) \leq n$.

Let $B = \{ \text{monic irreducible polynomials of degree } \leq m \} \cup \{g_0\}$ where $\langle g_0 \rangle = \mathbb{F}_p^*$. Then

$$r = \#B \sim \frac{p^m}{m} + \frac{p^{m-1}}{m-1} + \dots \sim cp^m$$

and we have

$$\epsilon = \frac{\#\{h \mid \deg(h) < n \text{ and } h \text{ factors as a product of polynomials of degree } \leq m\}}{p^n = \#\{\text{polynomials of degree } < n\}}$$

We want to find a lower bound on ϵ to give us an upper bound on $\frac{1}{\epsilon}$.

Let $v = \lfloor \frac{n}{m} \rfloor$. Assume that m does not divide n so that $vm < n$. Take v irreducible polynomials of degree $\leq m$ and multiply them out to get a polynomial of degree $< n$ which factors the way we want. There are at least $\binom{r}{v}$ choices for how to do this. For our purposes, $\binom{r}{v} \sim \frac{r^v}{v!}$. Thus we get an (approximate) lower bound on ϵ :

$$\epsilon \geq \frac{r^v/v!}{p^n}.$$

We can then attempt to figure out the running time of the index calculus method in this case. We have

$$\begin{aligned} \frac{R}{\epsilon} &\sim \frac{r}{\epsilon} \ll p^m \cdot \frac{p^n}{p^{mv}} \cdot v! \quad (r \ll p^m) \\ &= \frac{p^m p^{mv+a} v!}{p^{mv}} \ll p^{2m} v! \quad (n = mv + a, 0 \leq a < m) \\ &\ll p^{2m} e^{v \log v} \quad (v! \ll e^{v \log v}) \\ &\ll p^{2m} e^{\frac{n}{m} \log n} \end{aligned}$$

Here we must balance the p^{2m} term (becomes large if m is large) with the $e^{\frac{n}{m}}$ term (which becomes small if m is large). It turns out that the best way to do this is to set $m = \sqrt{n}$. In this case, we have

$$\frac{R}{\epsilon} \ll p^{2\sqrt{n}} e^{\sqrt{n} \log n} \quad (1)$$

$$\ll e^{\sqrt{n} \log n + 2\sqrt{n} \log p} \quad (2)$$

$$\ll e^{(n \log p)^{\frac{1/2+\delta}{1+\delta}}} \quad (3)$$

$$\sim e^{(\log |\mathbb{F}_q^*|)^\alpha} \quad (4)$$

Where we get from (2) to (3) by assuming that p is small in the sense that $\log p < n^\delta$. It follows that if p is small, then we can take $\log p = n^\delta$, and hence $\sqrt{n} \log p = n^{1/2+\delta} = (n \log p)^\alpha = n^{(1+\delta)\alpha} \implies \alpha = \frac{1/2+\delta}{1+\delta}$. We get from step (3) to step (4) by assuming that $\log p \leq n^\delta$.

In practice, the performance of the linear algebra computation will always dominate the running time.

3.3 Another Example

Let $G = \mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$. Then for $h \in \mathbb{F}_p^*$, we can lift h to \mathbb{Z} , with $1 \leq h \leq p-1$. Our factor base B is the set of primes $l \leq x$. In other words, we're hoping that h factors as a product of small primes. Then, by the Prime Number Theorem, we have

$$\#B = r \sim \frac{x}{\log x}$$

Let $v = \lfloor \frac{\log p}{\log x} \rfloor$. Then as before we have at least $\binom{r}{v}$ choices that factor as a product of small primes less than x (i.e. factor as we want them to). Thus we have

$$\#\{h < p \mid h \text{ is a product of primes } \leq x\} \geq \binom{r}{v}$$

So we must have as before that (approximately)

$$\epsilon \geq \frac{r^v/v!}{p}.$$

For comparison to the previous example, $\log p = n$ and $\log x = m$. So now we must choose $\log x = \sqrt{\log p}$, or $x = e^{\sqrt{\log p}}$.

The running time of this algorithm is $e^{(\log p)^\alpha}$, $\alpha < 1$ (in fact, α is close to $\frac{1}{2}$).

For p^n where p is big and n is small, take an extension K/\mathbb{Q} of degree n where p is inert and unramified so that

$$\mathcal{O}_k/(p) \cong \mathbb{F}_{p^n}$$

and proceed along similar lines. (This is in fact never done in practice for $n > 3$.)

3.4 Other examples

The function field sieve is another way of approaching the index calculus algorithm for \mathbb{F}_q . Instead of working in $\mathbb{F}_p[x]$, work in a ring of integers on an extension of $\mathbb{F}_p(x)$, and use the fact that there may be more prime divisors of low degree and use them as the factor base.

Suppose C/\mathbb{F}_q is an algebraic curve of genus $g \geq 1$. Then the Jacobian $J_C(\mathbb{F}_q)$ is an abelian group. Thus we can do index calculus with the factor base

$$B = \{\text{positive divisors of small degree}\}$$

which ends up working very well when g is very big.

If $G = \mathbb{F}_q^*$, $q = r^m$, then $\{x \in \mathbb{F}_q^* \mid N_{\mathbb{F}_q/\mathbb{F}_r} = 1\} = G$, and

$$|G| = \frac{q-1}{r-1}, \quad (\text{i.e. we've made } G \text{ smaller})$$

It turns out however that it is currently not known whether the DLP is any easier on this smaller group than it is on the larger one!