Computing Discrete Logarithms

Many cryptosystems could be broken if we could compute discrete logarithms quickly.

The first discrete logarithm algorithms below apply in any group. They are about the best one can do in elliptic curve groups.

The last algorithm depends on the idea of smoothness and solve the discrete logarithm problem only in the group R_p, the integers modulo a prime p, where one can define smooth numbers. The index calculus is much faster than the methods of Shanks and Pollard. Hence, the group R_p must be much larger than an elliptic curve group to achieve the same security.

A rough rule of thumb is that R_p with a 1024-bit prime p is about as safe as an elliptic curve modulo a 128-bit prime.

Consider first the congruence $a^x \equiv b \bmod p$. By analogy to ordinary logarithms, we may write $x = \mathrm{Log}_a b$ when p is understood from the context. These discrete logarithms enjoy many properties of ordinary logarithms, such as $\mathrm{Log}_a bc = \mathrm{Log}_a b + \mathrm{Log}_a c$, except that the arithmetic with logarithms must be done modulo $p-1$ because $a^{p-1} \equiv 1 \bmod p$.

Neglecting powers of $\log p$, the congruence may be solved in $O(p)$ time and $O(1)$ space by raising a to successive powers modulo p and comparing each with b. It may also be solved in $O(1)$ time and $O(p)$ space by looking up x in a precomputed table of pairs $(x, a^x \bmod p)$, sorted by the second coordinate. We next explain an intermediate method which takes essentially $O(\sqrt{p})$ time and $O(\sqrt{p})$ space.

Shanks' Baby-Step-Giant-Step Method

Shanks' baby-step-giant-step algorithm solves the congruence $a^x \equiv b \bmod p$ in $O(\sqrt{p} \log p)$ time and $O(\sqrt{p})$ space as follows.

Let $m = \lceil \sqrt{p-1} \rceil$.

Compute and sort the m ordered pairs $(j, a^{mj} \bmod p)$, for j from 0 to $m-1$, by the second coordinate.

Compute and sort the m ordered pairs $(i, ba^{-i} \bmod p)$, for i from 0 to $m-1$, by the second coordinate.

Find a pair (j, y) in the first list and a pair (i, y) in the second list.

This search will succeed because every integer between 0 and $p-1$ can be written as a two-digit number ji in base m.

Finally, $x = mj + i \bmod p - 1$.

Example Let $p = 23$, $a = 2$ and $h = 13$. Find $x = \mathrm{Log}_a h$.

We have $m = \lceil \sqrt{p-1} \rceil = \lceil \sqrt{22} \rceil = 5$. Also, $2^m \equiv 2^5 \equiv 32 \equiv 9 \pmod{23}$ and $a^{-1} \equiv 12 \pmod{23}$.

j	2^{5j}	i	$13 \cdot 12^i$
0	1	0	13
1	9	1	18
2	12	2	9
3	16	3	16
4	6	4	8

The value 9 appears for $j = 1$ and for $i = 2$, so $x = \mathrm{Log}_2 13 = mj + i = 5 \cdot 1 + 2 = 7 \pmod{22}$. Indeed, $2^7 \equiv 13 \pmod{23}$.

Also, the value 16 appears for $j = 3$ and for $i = 3$, so $x = \mathrm{Log}_2 13 = mj + i = 5 \cdot 3 + 3 = 18 \pmod{22}$. Indeed, $2^{18} \equiv 13 \pmod{23}$. Both 7 and 18 are correct.

Pollard's Methods

In 1978, Pollard invented two methods for finding discrete logarithms analogous to his rho method for factoring integers. Like Shanks' baby-step-giant-step algorithm, these algorithms work in any group and have complexity $O(\sqrt{p})$, where p is the group order. However, their space requirements are tiny.

We will describe the rho method for solving the congruence $a^x \equiv b \bmod p$, where p is prime, although it works in any group.

We are given a prime $p > 3$, a primitive root g modulo p and an element h of R_p, the group of nonzero integers modulo p. We seek the x modulo $p-1$ for which $g^x \equiv h \pmod{p}$. The answer x may be written $x = \mathrm{Log}_g h$.

Define three sequences $\{x_i\}, \{a_i\}, \{b_i\}$ by $x_0 = 1$, $a_0 = b_0 = 0$ and

if $0 < x_i < p/3$, then $x_{i+1} = hx_i \bmod p$,
$a_{i+1} = 1 + a_i \bmod p-1$ and $b_{i+1} = b_i \bmod p-1$,

if $p/3 < x_i < 2p/3$, then $x_{i+1} = x_i^2 \bmod p$,
$a_{i+1} = 2a_i \bmod p-1$ and $b_{i+1} = 2b_i \bmod p-1$,
and

if $2p/3 < x_i < p$, then $x_{i+1} = gx_i \bmod p$,
$a_{i+1} = a_i \bmod p-1$ and $b_{i+1} = 1 + b_i \bmod p-1$.

A simple induction argument shows that $x_i \equiv h^{a_i} g^{b_i} \pmod{p}$.

The mapping $x_i \to x_{i+1}$ is a random mapping from R_p to itself. By the birthday problem, after about $\sqrt{p}$ iterations of the mapping there will be a repeated value $x_i = x_j$.

We can use the Floyd cycle-finding algorithm to find two repeated values by computing two iterates of the mapping in the same loop, with one instance running twice as fast as the other. This gives us a subscript e with $x_{2e} = x_e$.

Now we have a congruence

$$h^{a_{2e}} g^{b_{2e}} \equiv h^{a_e} g^{b_e} \pmod{p}.$$

As we can easily find inverses modulo p, this leads at once to a congruence $h^m \equiv g^n \pmod{p}$, where $m \equiv a_e - a_{2e} \pmod{p-1}$ and

$$n \equiv b_{2e} - b_e \pmod{p-1}.$$

We can rewrite this as

$$mx \equiv m\mathrm{Log}_g h \equiv n \pmod{p-1}. \qquad (1)$$

Let $d = \gcd(m, p-1)$. We know that Congruence (1) must have a solution because g is a primitive root modulo p and p does not divide h. One of them is the answer x we seek. One can show that d is usually small, say, $d = 1$ or 2, so we can try all d solutions to Congruence (1) and find x.

Example Let $p = 999959$, $g = 7$ and $h = 3$. Find $x = \mathrm{Log}_g h$.

At $e = 1174$ we have $x_e = x_{2e} = 11400$, $m = 310686$ and $n = 764000$. Congruence (1) becomes $310686x \equiv 764000 \pmod{999958}$. The extended Euclidean algorithm gives

$$2 = \gcd(310686, 999958) =$$

$$= 148845 \cdot 310686 - 46246 \cdot 999958,$$

and we find that $3^2 \equiv 7^{356324} \pmod{p}$ and $3 \equiv \pm 7^{178162}$. Since 3 is a quadratic residue modulo p and -1 is not, the plus sign is correct and $x = \mathrm{Log}_g h = 178162$.

In the setting of an elliptic curve group E, we are given two points P and Q, are told that $Q = xP$ for some integer x, and must find x. The group is partitioned into three pieces of roughly equal size. The random mapping of $E \to E$ takes a point X into $X + P$, $X + X$ or $X + Q$, according to which piece of the group contains X. The initial value of the variable point X is the identity ∞. The a_i and b_i are defined just as above. A repeated point yields an equation $mQ = nP$, which means that $mx \equiv n \pmod{N}$, where N is the order of P in E. Since we know that $Q = xP$, this congruence must have a solution.

Pollard's Lambda Method

We describe Pollard's lambda method in the general setting of groups. This method is also called the kangaroo method, since it employs two kangaroos to hop around in the group.

Let G be a finite cyclic group with generator g and let h be an element of G. We seek the least positive integer x so that $h = g^x$. Suppose we know that x lies in the interval $a \leq x < b$.

Pollard defined two kangaroos, a tame one $\mathcal{T}$ starting at $t_0 = g^b$ (the upper end point of the interval) and a wild one $\mathcal{W}$ starting at $w_0 = h$ (an unknown point in the interval).

Define $d_0(\mathcal{T}) = b$, the initial distance of $\mathcal{T}$ from the origin.

Let $d_0(\mathcal{W}) = 0$, the initial distance of $\mathcal{W}$ from h.

Let $S = \{g^{s_1}, \ldots, g^{s_k}\}$ be a set of jumps.

Let G be partitioned into k pieces and for each $r \in G$, let $f(r)$, with $1 \leq f(r) \leq k$, be the number of the piece to which r belongs.

The exponents s_i should be positive and small compared to $b-a$. Pollard suggested that $s_i = 2^i$ might be good choices.

Think of the s_i as the lengths of the hops of the kangaroos.

Now let the two kangaroos hop around in the group G.

The tame one $\mathcal{T}$ hops from t_i to $t_{i+1} = t_i g^{s_{f(t_i)}}$ for $i \geq 0$. Keep track of $\mathcal{T}$'s distance from the origin by computing $d_{i+1}(\mathcal{T}) = d_i(\mathcal{T}) + s_{f(t_i)}$ for $i \geq 0$. It follows that $t_i = g^{d_i(\mathcal{T})}$ for $i \geq 0$.

After a while $\mathcal{T}$ stops and sets a trap at its final location, say t_m.

Then the wild kangaroo hops along the path from w_i to $w_{i+1} = w_i g^{s_{f(w_i)}}$ for $i \geq 0$.

Keep track of $\mathcal{W}$'s distance from the unknown starting position (the discrete logarithm of h) by computing $d_{i+1}(\mathcal{W}) = d_i(\mathcal{W}) + s_{f(w_i)}$ for $i \geq 0$. Then $w_i = g^{x+d_i(\mathcal{W})}$ for $i \geq 0$.

After each hop, we check to see whether $\mathcal{W}$ has fallen into the trap by testing whether $w_i = t_m$. With a good choice of the parameters s_i, it is highly likely that eventually $w_n = t_m$ for some n.

Then we have $x = d_m(\mathcal{T}) - d_n(\mathcal{W})$.

If we find that $d_n(\mathcal{W}) > d_m(\mathcal{T})$, then $\mathcal{W}$ has passed the trap. In this case, we start a new wild kangaroo at $w_0 = hg^z$ for some small integer $z > 0$ and hope it falls into the trap.

If the two kangaroos ever land on the same spot ($w_i = t_j$), then their paths will coincide from that point on and $\mathcal{W}$ will be trapped. If you draw their paths going upwards, the paths will form the Greek letter lambda: λ. This is the reason for the name.

The most important property of the jumps sizes s_i is their average. Van Oorschot and Wiener have shown that if the mean value of the s_i is about $\frac{1}{2}\sqrt{b-a}$ and if $\mathcal{T}$ makes about $0.7\sqrt{b-a}$ hops before setting the trap, the running time will be minimal.

With these choices, $\mathcal{W}$ will hop about $2.7\sqrt{b-a}$ times before getting trapped, which happens three-fourths of the time, or passing the trap. The space requirement is about $O(\log(b-a))$.

Example Let $p = 31$, $g = 3$ and $h = 13$. Find $x = \mathrm{Log}_g h$.

Let $k = 3$, $s_1 = 2$, $s_2 = 3$, $s_3 = 4$, so $\mathcal{S} = \{9, 27, 19\}$. Start the tame kangaroo at $t_0 = 1$ and let it set a trap after six hops:

i	$f(i)$	$d_i(\mathcal{T})$	t_i
0	1	0	1
1	1	2	9
2	2	4	19
3	2	7	17
4	3	10	25
5	1	14	10
6	3	16	28

The wild kangaroo starts at $w_0 = 13$ and hops:

i	$f(i)$	$d_i(\mathcal{W})$	w_i
0	2	0	13
1	1	3	10
2	2	5	28

The wild kangaroo $\mathcal{W}$ is trapped and $x = \mathrm{Log}_g h = d_6(\mathcal{T}) - d_2(\mathcal{W}) = 16 - 5 = 11$.

Discrete Logarithms via Index Calculus

There is a faster way to solve $a^x \equiv b \pmod{p}$ using a method similar to the integer factoring algorithm QS. It is called the **index calculus method**.

If $a^x \equiv b \pmod{p}$, then we write $x = \mathrm{Log}_a(b)$. Note that $\mathrm{Log}_a(b)$ is an integer determined modulo $p-1$ because of Fermat's theorem: $a^{p-1} \equiv 1 \pmod{p}$.

$\mathrm{Log}_a(b)$ is called the discrete logarithm of b to base a. (The modulus p is usually supressed.)

Choose a finite set of primes $p_1, \ldots, p_k$, usually all primes $\leq B$. Perform the following precomputation which depends on a and p but not on b. For many random values of x, try to factor $a^x \bmod p$ using the primes in the factor base.

Save at least $k + 20$ of the factored residues:

$$a^{x_j} \equiv \prod_{i=1}^{k} p_i^{e_{ij}} \pmod{p} \text{ for } 1 \leq j \leq k + 20,$$

or equivalently

$$x_j \equiv \sum_{i=1}^{k} e_{ij} \mathrm{Log}_a p_i \pmod{p-1} \text{ for } 1 \leq j \leq k+20.$$

Use linear algebra to solve for the $\text{Log}_a p_i$.

When b is given, perform the following main computation to find $\text{Log}_a b$. Try many random values for s until one is found for which $ba^s \bmod p$ can be factored using only the primes in the factor base.

Write it as

$$ba^s \equiv \prod_{i=1}^{k} p_i^{c_i} \pmod{p}$$

or

$$(\text{Log}_a b) + s \equiv \sum_{i=1}^{k} c_i \text{Log}_a p_i \pmod{p-1}.$$

Substitute the values of $\text{Log}_a p_i$ found in the precomputation to get $\text{Log}_a b$.

Using arguments like those for the running time of the quadratic sieve factoring algorithms, one can prove that the precomputation takes time

$$\exp\left(\sqrt{2\log p \log\log p}\right),$$

while the main computation takes time

$$\exp\left(\sqrt{\log p \log\log p}\right).$$