10 CHAPTER Discrete Logarithms

In this chapter, we discuss the difficulty of the discrete logarithm problem (DL problem). The security of many public-key cryptosystems is based on the difficulty of this problem. An example is the ElGamal cryptosystem (see Section 8.6).

First, we describe generic algorithms that work in any cyclic group. Then we explain special algorithms that work in the group $(\mathbb{Z}/p\mathbb{Z})^*$ for a prime number p.

10.1 The DL Problem

In this chapter, G is a finite cyclic group of order n, γ is a generator of this group, and 1 is the neutral element in G. We assume that the group order n is known. Many algorithms for computing discrete logarithms, however,also work with an upper bound for the group order. Moreover, we let α be a group element. The goal is to find the smallest nonnegative integer x with

$$\alpha = \gamma^x. \tag{10.1}$$

It is called the *discrete logarithm* of α to the base γ. When we talk about the DL problem, we mean the problem of finding this integer x.

There is a more general version of the DL problem. In a group H, which is not necessarily cyclic, two elements α and γ are given. The problem is to decide whether there is an integer x such that (10.1) is satisfied, and if such an x exists to find the smallest nonnegative x. In cryptographic applications, the existence of x is typically guaranteed. The attacker's only problem is to find it. Therefore, our version of the DL problem is sufficient for the cryptographic context.

10.2 Enumeration

The simplest method for computing the discrete logarithm x from (10.1) is to test whether $x = 0, 1, 2, 3, \ldots$ satisfy (10.1). As soon as the answer is "yes", the discrete logarithm is found. This is called *enumeration*. Enumeration requires $x - 1$ multiplications and x comparisons in G. Only the elements α, γ and γ^x need to be stored. Hence, enumeration only requires space for three group elements.

Example 10.2.1
We determine the discrete logarithm of 3 to the base 5 in $(\mathbb{Z}/2017\mathbb{Z})^*$. Enumeration yields $x = 1030$ using 1029 multiplications modulo 2017.

In cryptographic applications, we have $x \geq 2^{160}$. Therefore, enumeration is infeasible because it would require at least $2^{160} - 1$ group operations.

10.3 Shanks Baby-Step Giant-Step Algorithm

A considerable improvement of the enumeration algorithm is the *baby-step giant-step algorithm* of D. Shanks. This algorithm requires

fewer group operations but more storage. We describe this algorithm as follows.

We set

$$m = \lceil \sqrt{n} \rceil$$

and write the unknown discrete logarithm x as

$$x = qm + r, \quad 0 \leq r < m.$$

Hence, r is the remainder and q is the quotient of the division of x by m. The baby-step giant-step algorithm computes q and r. This works as follows.

We have

$$\gamma^{qm+r} = \gamma^x = \alpha.$$

This implies

$$(\gamma^m)^q = \alpha\gamma^{-r}.$$

First, we compute the set of *baby-steps*

$$B = \{(\alpha\gamma^{-r}, r) : 0 \leq r < m\}.$$

If in this set we find a pair $(1, r)$, then $\alpha\gamma^{-r} = 1$ (i.e., $\alpha = \gamma^r$). Hence, we can set $x = r$ with the smallest such x. If we do not find such a pair, we determine

$$\delta = \gamma^m.$$

Then we test for $q = 1, 2, 3, \ldots$ whether the group element δ^q is the first component of an element in B (i.e., whether there is a pair (δ^q, r) in B). As soon as this is true, we have

$$\alpha\gamma^{-r} = \delta^q = \gamma^{qm}$$

which implies

$$\alpha = \gamma^{qm+r}.$$

Therefore, the discrete logarithm is

$$x = qm + r.$$

The elements δ^q, $q = 1, 2, 3 \ldots$ are called *giant-steps*. We must compare each δ^q with all first components of the baby-step set B. To make

this comparison efficient, the elements of B are stored in a hash table where the key is the first element (see [21], Chapter 12).

Example 10.3.1
We determine the discrete logarithm of 3 to the base 5 in $(\mathbb{Z}/2017\mathbb{Z})^*$. We have $\gamma = 5 + 2017\mathbb{Z}$, $\alpha = 3 + 2017\mathbb{Z}$, $m = \lceil\sqrt{2017}\rceil = 45$. The baby-step set is

$B = \{(3,0),(404,1),(1291,2),(1065,3),(213,4),(446,5),(896,6),$
$(986,7),(1004,8),(1411,9),(1089,10),(1428,11),(689,12),(1348,13),$
$(673,14),(538,15),(511,16),(909,17),(1392,18),(1892,19),$
$(1992,20),(2012,21),(2016,22),(1210,23),(242,24),(1662,25),$
$(1946,26),(1196,27),(1046,28),(1016,29),(1010,30),(202,31),$
$(1654,32),(1541,33),(1115,34),(223,35),(448,36),(493,37),(502,38),$
$(1714,39),(1553,40),(714,41),(1353,42),(674,43),(1345,44)\}.$

Here, the residue classes are represented by their least nonnegative representatives.

Next, we compute $\delta = \gamma^m = 45 + 2017\mathbb{Z}$. The giant-steps are

45, 8, 360, 64, 863, 512, 853, 62, 773, 496, 133, 1951,
1064, 1489, 444, 1827, 1535, 497, 178, 1959, 1424, 1553.

We find $(1553, 40)$ in the baby-step set. Therefore, $\alpha\gamma^{-40} = 1553 + 2017\mathbb{Z}$. Since 1553 has been found as the twenty-second giant-step, we obtain

$$\gamma^{22*45} = \alpha\gamma^{-40}.$$

Hence

$$\gamma^{22*45+40} = \alpha.$$

The solution of the DL problem is $x = 22 * 45 + 40 = 1030$. To compute the baby-step set, 45 multiplications mod 2017 were necessary. To compute the giant-steps, 21 multiplications mod 2017 were necessary. Enumeration requires many more multiplications, namely 1029. On the other hand, a baby-step set with 45 elements had to be stored, whereas enumeration only requires the storage of three elements.

If we use a hash table, then a constant number of comparisons are sufficient to check whether a group element computed as a giant-step is a first component of a baby-step. Therefore, the following result is easy to verify.

Theorem 10.3.2
The baby-step giant-step algorithm requires $O(\sqrt{|G|})$ multiplications and comparisons in G. It needs storage for $O(\sqrt{|G|})$ elements of G.

Time and space requirements of the baby-step giant-step algorithm are approximately $\sqrt{|G|}$. If $|G| > 2^{160}$, then computing discrete logarithms with the baby-step giant-step algorithm is still infeasible.

10.4 The Pollard ρ-Algorithm

The algorithm of Pollard described in this section has the same running time as the baby-step giant-step algorithm, namely $O(\sqrt{|G|})$. However, it only requires constant storage, while the baby-step giant-step algorithm needs to store roughly $\sqrt{|G|}$ group elements.

Again, we want to solve the DL problem (10.1). We need three pairwise disjoint subsets G_1, G_2, G_3 of G such that $G_1 \cup G_2 \cup G_3 = G$. Let $f : G \to G$ be defined by

$$f(\beta) = \begin{cases} \gamma\beta & \text{if } \beta \in G_1, \\ \beta^2 & \text{if } \beta \in G_2, \\ \alpha\beta & \text{if } \beta \in G_3. \end{cases}$$

We choose a random number x_0 in the set $\{1, \ldots, n\}$ and compute the group element $\beta_0 = \gamma^{x_0}$. Then, we compute the sequence (β_i) by the recursion

$$\beta_{i+1} = f(\beta_i).$$

The elements of this sequence can be written as

$$\beta_i = \gamma^{x_i}\alpha^{y_i}, \quad i \geq 0.$$

Here, x_0 is the initial random number, $y_0 = 0$, and we have

$$x_{i+1} = \begin{cases} x_i + 1 \mod n & \text{if } \beta_i \in G_1, \\ 2x_i \mod n & \text{if } \beta_i \in G_2, \\ x_i & \text{if } \beta_i \in G_3, \end{cases}$$

and

$$y_{i+1} = \begin{cases} y_i & \text{if } \beta_i \in G_1, \\ 2y_i \mod n & \text{if } \beta_i \in G_2, \\ y_i + 1 \mod n & \text{if } \beta_i \in G_3. \end{cases}$$

Since we are working in a finite group, two elements in the sequence (β_i) must be equal (i.e., there is $i \geq 0$ and $k \geq 1$ with $\beta_{i+k} = \beta_i$). This implies

$$\gamma^{x_i}\alpha^{y_i} = \gamma^{x_{i+k}}\alpha^{y_{i+k}}$$

and therefore

$$\gamma^{x_i - x_{i+k}} = \alpha^{y_{i+k} - y_i}.$$

Hence, by Corollary 2.9.3, the discrete logarithm x of α to the base γ satisfies

$$(x_i - x_{i+k}) \equiv x(y_{i+k} - y_i) \mod n.$$

We solve this congruence. The solution is unique mod n if $y_{i+k} - y_i$ is invertible mod n. If the solution is not unique, then the discrete logarithm can be found by testing the different possibilities mod n. If there are too many possibilities, then the algorithm is applied again with a different initial x_0.

We estimate the number of elements β_i that must be computed before a *match* is found (i.e., a pair $(i, i+k)$ of indices for which $\beta_{i+k} = \beta_i$). For this purpose, we use the birthday paradox (see Section 4.3). The possible birthdays are the group elements. We assume that the elements of the sequence $(\beta_i)_{i\geq 0}$ are random group elements. This is obviously not true, but the construction of the sequence makes it very similar to a random sequence. As we have shown in Section 4.3, $O(\sqrt{|G|})$ sequence elements are sufficient to make the probability for a match greater than $1/2$.

Thus far, our algorithm must store all triplets (β_i, x_i, y_i). As we have seen, the number of elements of the sequence is of the order of

magnitude $\sqrt{|G|}$, as in Shanks' baby-step giant-step algorithm. But we will now show that it suffices to store a single triplet. Therefore, the Pollard ρ-algorithm is much more space efficient than the baby-step giant-step algorithm.

Initially, (β_1, x_1, y_1) is stored. Now suppose that at a certain point in the algorithm (β_i, x_i, y_i) is stored. Then (β_j, x_j, y_j) is computed for $j = i+1, i+2, \ldots$ until either a match is found or $j = 2i$. In the latter case, we delete β_i and store β_{2i}. Hence, we only store the triplets (β_i, x_i, y_i) with $i = 2^k$. Before we show that in this way a match is found, we give an example.

Example 10.4.1
With the Pollard ρ-algorithm, we solve the discrete logarithm problem

$$5^x \equiv 3 \mod 2017.$$

All residue classes are represented by their smallest nonnegative representatives. We set

$$G_1 = \{1, \ldots, 672\}, G_2 = \{673, \ldots, 1344\}, G_3 = \{1345, \ldots, 2016\}.$$

As our starting value, we use $x_0 = 1023$.

Here are the stored triplets and the final triplet, which is a match and allows us to compute the discrete logarithm.

j	β_j	x_j	y_j
0	986	1023	0
1	2	30	0
2	10	31	0
4	250	33	0
8	1366	136	1
16	1490	277	8
32	613	447	155
64	1476	1766	1000
98	1476	966	1128

We see that

$$5^{800} \equiv 3^{128} \mod 2017.$$

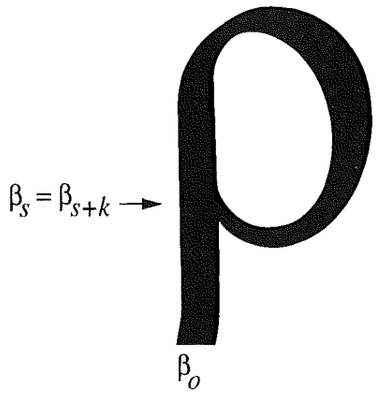

FIGURE 10.1 The Pollard ρ-algorithm.

To compute x, we must solve the congruence

$$128x \equiv 800 \mod 2016.$$

Since $\gcd(128, 2016) = 32$ divides 800, this congruence has a solution that is unique modulo 63. To find x, we solve the congruence

$$4z \equiv 25 \mod 63.$$

We obtain the solution $z = 22$. Therefore, the discrete logarithm is one of the values $x = 22 + k * 63$, $0 \leq k < 32$. For $k = 16$, we find the discrete logarithm $x = 1030$.

Now we prove that the preceding algorithm will eventually find a match.

First, we show that the sequence $(\beta_i)_{i\geq 0}$ is periodic after a match occurs. Let $(s, s+k)$ be the first match, which is not necessarily found in the algorithm because the wrong elements are stored. Then $k > 0$ and $\beta_{s+k} = \beta_s$. Moreover, $\beta_{s+k+l} = \beta_{s+l}$ for $l \geq 0$ since the construction of the next group element only depends on the previous group element in the sequence, so the sequence (β_i) is in fact periodic. We can draw it as the Greek letter ρ (see Figure 10.1). The *preperiod* is the sequence $\beta_0, \beta_1, \ldots, \beta_{s-1}$. It has length s. The *period* is $\beta_s, \beta_{s+1}, \ldots, \beta_{s+k-1}$ and has length k.

Now we explain how a match is found if only one triplet is stored. Denote by i the index of the triplet that is currently stored. If $i = 2^j \geq s$, then β_i is in the period. In addition, if $2^j \geq k$, then the sequence

$$\beta_{2^j+1}, \beta_{2^j+2}, \ldots, \beta_{2^{j+1}}$$

is at least as long as the period. One of its elements is equal to β_{2^j}. But this is exactly the sequence that is computed after b_{2^j} has been stored. All of its elements are compared with β_{2^j}. Hence, one of these comparisons will reveal a match. Because the sum of the lengths of the preperiod and the period is $O(\sqrt{|G|})$, it follows that the number of sequence elements that must be computed before a match is found is $O(\sqrt{|G|})$. Therefore, the algorithm has running time $O(\sqrt{|G|})$ and needs space for $O(1)$ triplets. This is much more space efficient than the baby-step giant-step algorithm.

The algorithm is even more efficient if eight triplets are stored. This works as follows. Initially, all eight triplets are equal to (β_0, x_0, y_0). Then those triplets are successively replaced. Let i be the index of the last stored triplet. Initially, we have $i = 1$. For $j = 1, 2, \ldots$ we compute (β_j, x_j, y_j) and do the following:

1. If β_j is equal to one of the stored group elements, then a match is found and the computation of the sequence terminates.
2. If $j \geq 3i$, then the first of the eight triplets is deleted and (β_j, x_j, y_j) is the new last triplet.

This modification does not change the asymptotic time or space complexity.

10.5 The Pohlig-Hellman Algorithm

We now show that the problem of computing the discrete logarithm in our group G can be reduced to a discrete logarithm problem in a cyclic group of prime order if we know the factorization

$$n = |G| = \prod_{p|n} p^{e(p)}$$

of the group order $n = |G|$ of our cyclic group.

10.5.1 Reduction to prime powers

For each prime divisor p of n, we set

$$n_p = n/p^{e(p)}, \quad \gamma_p = \gamma^{n_p}, \quad \alpha_p = \alpha^{n_p}.$$

Then the order of γ_p is exactly $p^{e(p)}$ and

$$\gamma_p^x = \alpha_p.$$

The group element α_p belongs to the cyclic group generated by γ_p. Therefore, the discrete logarithm $x(p)$ of α_p to the base γ_p exists. The following theorem describes how the discrete logarithm x can be computed from all the $x(p)$.

Theorem 10.5.1
For a prime divisor p of n, let $x(p)$ be the discrete logarithm of α_p to the base γ_p. Moreover, let $x \in \{0, 1, \ldots, n-1\}$ be a solution of the simultaneous congruence $x \equiv x(p) \bmod p^{e(p)}$ for all prime divisors p of n. Then x is the discrete logarithm of α to the base γ.

Proof. We have

$$(\gamma^{-x}\alpha)^{n_p} = \gamma_p^{-x(p)}\alpha_p = 1$$

for all prime divisors p of n. Therefore, the order of the element $\gamma^{-x}\alpha$ is a divisor of n_p for all prime divisors p of n and therefore a divisor of the gcd of all n_p. But this gcd is 1. Hence, the order is 1 and this shows that $\alpha = \gamma^x$. □

We have seen that the discrete logarithm x can be computed by first determining all $x(p)$ and then applying the Chinese remainder theorem. The baby-step giant-step algorithm takes time $O(\sqrt{p^{e(p)}})$ for computing $x(p)$. If n has more than one prime divisor, then this modification is already considerably faster than the application of the baby-step giant-step algorithm in the full group. The computing time for the application of the Chinese remainder theorem is negligible.

Example 10.5.2
As in Example 10.3.1, let G be the multiplicative group of residues mod 2017. Its order is

$$2016 = 2^5 * 3^2 * 7.$$

We compute the discrete logarithm $x(2)$ in a subgroup of order $2^5 = 32$, $x(3)$ in a subgroup of order 9, and $x(7)$ in a subgroup of order 7. For those computations, we could use the baby-step giant-step algorithm. A more efficient variant is described in the next section.

10.5.2 Reduction to prime orders

In the previous section, we have seen that the computation of discrete logarithms in the cyclic group G can be reduced to the computation of discrete logarithms in subgroups of prime power order. Now we will show that the computation of discrete logarithms in cyclic groups of prime power order can be reduced to the computation of discrete logarithms in subgroups of prime order.

Let $|G| = n = p^e$ for a prime number p and a positive integer e. We want to solve the congruence (10.1) in this group. We know that $x < p^e$. By Theorem 1.3.3, we can write

$$x = x_0 + x_1 p + \ldots + x_{e-1}p^{e-1}, \quad 0 \le x_i < p, \quad 0 \le i \le e-1. \quad (10.2)$$

We show that the coefficient x_i, $0 \le i \le e-1$ is a discrete logarithm in a group of order p.

Raise the equation $\gamma^x = \alpha$ to the power p^{e-1}. Then

$$\gamma^{p^{e-1}x} = \alpha^{p^{e-1}}. \quad (10.3)$$

Now we obtain from (10.2)

$$p^{e-1}x = x_0 p^{e-1} + p^e(x_1 + x_2 p + \ldots x_{e-1}p^{e-2}). \quad (10.4)$$

From Fermat's little theorem (see Theorem 2.11.1), (10.4), and (10.3) we obtain

$$(\gamma^{p^{e-1}})^{x_0} = \alpha^{p^{e-1}}. \quad (10.5)$$

By (10.5), the coefficient x_0 is a discrete logarithm in a group of order p because $\gamma^{p^{e-1}}$ is of order p. The other coefficients are determined recursively. Suppose that $x_0, x_1, \ldots, x_{i-1}$ have been determined. Then

$$\gamma^{x_i p^i + \ldots + x_{e-1}p^{e-1}} = \alpha\gamma^{-(x_0 + x_1 p + \ldots + x_{i-1}p^{i-1})}.$$

Denote the group element on the right-hand side by α_i. If we raise this equation to the power p^{e-i-1}, then we obtain

$$(\gamma^{p^{e-1}})^{x_i} = \alpha_i^{p^{e-i-1}}, \quad 0 \le i \le e-1. \tag{10.6}$$

This is a discrete logarithm problem with solution x_i. Hence, in order to compute $x(p)$ we must solve e DL problems in groups of order p.

Example 10.5.3
As in Example 10.3.1, we solve

$$5^x \equiv 3 \bmod 2017.$$

The order of the multiplicative group of residues mod 2017 is

$$n = 2016 = 2^5 * 3^2 * 7.$$

First, we determine $x(2) = x \bmod 2^5$. We obtain $x(2)$ as a solution of the congruence

$$(5^{3^2*7})^{x(2)} \equiv 3^{3^2*7} \bmod 2017.$$

This means that

$$500^{x(2)} \equiv 913 \bmod 2017.$$

To solve this congruence, we write

$$x(2) = x_0(2) + x_1(2) * 2 + x_2(2) * 2^2 + x_3(2) * 2^3 + x_4(2) * 2^4.$$

According to (10.6), the coefficient $x_0(2)$ is a solution of

$$2016^{x_0(2)} \equiv 1 \bmod 2017.$$

We obtain $x_0(2) = 0$ and $\alpha_1 = \alpha_0 = 913 + 2017\mathbb{Z}$. Hence, $x_1(2)$ is the solution of

$$2016^{x_1(2)} \equiv 2016 \bmod 2017.$$

We obtain $x_1(2) = 1$ and $\alpha_2 = 1579 + 2017\mathbb{Z}$. Hence, $x_2(2)$ is the solution of

$$2016^{x_2(2)} \equiv 2016 \bmod 2017.$$

We obtain $x_2(2) = 1$ and $\alpha_3 = 1 + 2017\mathbb{Z}$, so $x_3(2) = x_4(2) = 0$. Concluding those computations, we obtain

$$x(2) = 6.$$

Now we compute

$$x(3) = x_0(3) + x_1(3) * 3.$$

We obtain $x_0(5)$ as the solution of

$$294^{x_0(3)} \equiv 294 \bmod 2017,$$

so $x_0(3) = 1$ and $\alpha_1 = 294 + 2017\mathbb{Z}$. Hence, $x_1(3) = 1$ and

$$x(3) = 4.$$

Finally, we compute $x(7)$ as the solution of the congruence

$$1879^{x(7)} \equiv 1879 \bmod 2017,$$

so $x(7) = 1$. We obtain x as the solution of the simultaneous congruence

$$x \equiv 6 \bmod 32, \quad x \equiv 4 \bmod 9, \quad x \equiv 1 \bmod 7.$$

The solution is $x = 1030$.

10.5.3 Complete algorithm and analysis

We describe the complete Pohlig-Hellman algorithm and analyze it. First, the group elements $\gamma_p = \gamma^{n_p}$ and $\alpha_p = \alpha^{n_p}$ are computed for all prime divisors p of n. Then the coefficients $x_i(p)$ are computed for all prime divisors p of n and $0 \le i \le e(p) - 1$ using the Pollard ρ-algorithm or Shanks' baby-step giant-step algorithm. Finally, the Chinese remainder theorem is used to compute the discrete logarithm. The complexity of the algorithm is estimated in the following theorem.

Theorem 10.5.4
The Pohlig-Hellman algorithm finds discrete logarithms in the cyclic group G using $O(\sum_{p \| |G|}(e(p)(\log |G| + \sqrt{p})))$ group operations.

Proof. We use the notation introduced in the previous section. The computation of the powers γ_p and α_p for a prime divisor p of $n = |G|$ requires $O(\log n)$ group operations. The computation of each digit in $x(p)$ for a prime divisor p of n requires $O(\log n)$ group operations for

powers and $O(\sqrt{p})$ group operations for the baby-step giant-step algorithm. The number of digits is $e(p)$. For the Chinese remaindering step no group operations are necessary. □

Note that by Theorem 2.15.3 the time for the Chinese remaindering step is $O((\log|G|)^2)$.

Theorem 10.5.3 shows that the time for computing discrete logarithms with the Pohlig-Hellman algorithm is dominated by the square root of the largest prime divisor of $|G|$. If this prime divisor is small, then it is easy to compute discrete logarithms in G.

Example 10.5.5
The integer $p = 2 * 3 * 5^{278} + 1$ is a prime number. Its binary length is 649. The order of the multiplicative group of residues mod p is $p - 1 = 2 * 3 * 5^{278}$. The computation of discrete logarithms in this group is very easy because the largest prime divisor of the group order is 5. Therefore, this prime cannot be used in the ElGamal cryptosystem.

10.6 Index Calculus

For multiplicative groups of residues modulo prime numbers or, more generally, for the unit group of a finite field, there are more efficient DL algorithms, the *index calculus algorithms*. They are closely related to integer factoring algorithms such as the quadratic sieve and the number field sieve. In this section, we describe a simple index calculus algorithm.

10.6.1 Idea

Let p be a prime number, g a primitive root mod p, and $a \in \{1, \ldots, p-1\}$. We want to solve the discrete logarithm problem

$$g^x \equiv a \bmod p. \tag{10.7}$$

We choose a bound B and determine the set

$$F(B) = \{q \in \mathbb{P} : q \leq B\}.$$

This is the *factor base*. An integer b is called B-smooth if it has only prime factors in $F(B)$.

Example 10.6.1
Let $B = 15$. Then $F(B) = \{2, 3, 5, 7, 11, 13\}$. The number 990 is 15-smooth. Its prime factorization is $990 = 2 * 3^2 * 5 * 11$.

We proceed in two steps. First, we compute the discrete logarithms of the factor base elements; that is, we solve

$$g^{x(q)} \equiv q \bmod p \tag{10.8}$$

for all $q \in F(B)$. Then we determine an exponent $y \in \{1, 2, \ldots, p-1\}$ such that $ag^y \bmod p$ is B-smooth. We obtain

$$ag^y \equiv \prod_{q \in F(B)} q^{e(q)} \bmod p \tag{10.9}$$

with nonnegative exponents $e(q)$, $q \in F(B)$. Equations (10.8) and (10.9) imply

$$ag^y \equiv \prod_{q \in F(B)} q^{e(q)} \equiv \prod_{q \in F(B)} g^{x(q)e(q)} \equiv g^{\sum_{q \in F(B)} x(q)e(q)} \bmod p,$$

and hence

$$a \equiv g^{\sum_{q \in F(B)} x(q)e(q) - y} \bmod p.$$

Therefore,

$$x = \left(\sum_{q \in F(B)} x(q)e(q) - y \right) \bmod (p-1) \tag{10.10}$$

is the discrete logarithm for which we were looking.

10.6.2 Discrete logarithms of the factor base elements

To compute the discrete logarithms of the factor base elements, we choose random numbers $z \in \{1, \ldots, p-1\}$ and compute $g^z \bmod p$. We check whether those numbers are B-smooth. If they are, we compute

the decomposition

$$g^z \bmod p = \prod_{q \in F(B)} q^{f(q,z)}.$$

Each exponent vector $(f(q,z))_{q \in F(B)}$ is called a *relation*.

Example 10.6.2
We choose $p = 2027$, $g = 2$ and determine relations for the factor base $\{2, 3, 5, 7, 11\}$. We obtain

$$\begin{aligned} 3 * 11 &= 33 \equiv 2^{1593} \bmod 2027 \\ 5 * 7 * 11 &= 385 \equiv 2^{983} \bmod 2027 \\ 2^7 * 11 &= 1408 \equiv 2^{1318} \bmod 2027 \\ 3^2 * 7 &= 63 \equiv 2^{293} \bmod 2027 \\ 2^6 * 5^2 &= 1600 \equiv 2^{1918} \bmod 2027. \end{aligned}$$

If we have found as many relations as there are factor base elements, then we try to find the discrete logarithms by solving a linear system. Using (10.8), we obtain

$$g^z \equiv \prod_{q \in F(B)} q^{f(q,z)} \equiv \prod_{q \in F(B)} g^{x(q)f(q,z)} \equiv g^{\sum_{q \in F(B)} x(q)f(q,z)} \bmod p.$$

This implies

$$z \equiv \sum_{q \in F(B)} x(q)f(q,z) \bmod (p-1) \tag{10.11}$$

for all z, so each relation yields one linear congruence. We can solve this linear system by applying the Gauss algorithm modulo each prime power l^e of $p-1$. If $e = 1$, then the standard Gauss algorithm over a field can be applied. If $e > 1$, then the linear algebra is slightly more complicated. Finally, the $x(q)$ are computed using the Chinese remainder theorem.

Example 10.6.3
We continue Example 10.6.2. If we write

$$q \equiv g^{x(q)} \bmod 2027, \quad q = 2, 3, 5, 7, 11$$

and use the relations from Example 10.6.2, then we obtain the linear system

$$x(3) + x(11) \equiv 1593 \bmod 2026$$

$$\begin{aligned} x(5) + x(7) + x(11) &\equiv 983 \bmod 2026 \\ 7x(2) + x(11) &\equiv 1318 \bmod 2026 \\ 2x(3) + x(7) &\equiv 293 \bmod 2026 \\ 6x(2) + 2x(5) &\equiv 1918 \bmod 2026. \end{aligned} \tag{10.12}$$

Because $2026 = 2 * 1013$ and 1013 is prime, we solve this system mod 2 and mod 1013. We obtain

$$\begin{aligned} x(3) + x(11) &\equiv 1 \bmod 2 \\ x(5) + x(7) + x(11) &\equiv 1 \bmod 2 \\ x(2) + x(11) &\equiv 0 \bmod 2 \\ x(7) &\equiv 1 \bmod 2. \end{aligned} \tag{10.13}$$

We know that $x(2) = 1$ because the primitive root $g = 2$ is used, so we find

$$x(2) \equiv x(5) \equiv x(7) \equiv x(11) \equiv 1 \bmod 2, \quad x(3) \equiv 0 \bmod 2. \tag{10.14}$$

Next, we compute the discrete logarithms of the factor base elements mod 1013. Again, we have $x(2) = 1$. From (10.12), we get

$$\begin{aligned} x(3) + x(11) &\equiv 580 \bmod 1013 \\ x(5) + x(7) + x(11) &\equiv 983 \bmod 1013 \\ x(11) &\equiv 298 \bmod 1013 \\ 2x(3) + x(7) &\equiv 293 \bmod 1013 \\ 2x(5) &\equiv 899 \bmod 1013. \end{aligned} \tag{10.15}$$

This implies $x(11) \equiv 298 \bmod 1013$. To compute $x(5)$, we invert 2 mod 1013. The result is $2 * 507 \equiv 1 \bmod 1013$. Hence, $x(5) \equiv 956 \bmod 1013$. From the second congruence, we obtain $x(7) \equiv 742 \bmod 1013$. From the first congruence, we obtain $x(3) \equiv 282 \bmod 1013$. Using (10.14), we finally obtain

$$x(2) = 1, x(3) = 282, x(5) = 1969, x(7) = 1755, x(11) = 1311.$$

It is easy to verify that this result is correct.

10.6.3 Individual logarithms

If the discrete logarithms of the factor base elements are computed, then the discrete logarithm of a to the base g is determined. We choose a random $y \in \{1, \ldots, p-1\}$. If $ag^y \bmod p$ is B-smooth, then (10.10) is applied. Otherwise, we choose a new y.

Example 10.6.4
We solve

$$2^x \equiv 13 \bmod 2027.$$

We choose a random $y \in \{1, \ldots, 2026\}$ until all prime factors of $13 * 2^y \bmod 2027$ are in the factor base $\{2, 3, 5, 7, 11\}$. We find

$$2 * 5 * 11 = 110 \equiv 13 * 2^{1397} \bmod 2027.$$

Using (10.10), we obtain $x = (1 + 1969 + 1311 - 1397) \bmod 2026 = 1884$.

10.6.4 Analysis

It can be shown that the index calculus algorithm that was described in the previous sections has subexponential running time $L_p[1/2, c + o(1)]$, where the constant c depends on the technical realization of the algorithm; for example, on the complexity of the algorithm for solving the linear system. The analysis is similar to the analysis of the quadratic sieve in Section 9.4. Since all of the generic algorithms described earlier have exponential running time, index calculus algorithms are asymptotically much more efficient and also much faster in practice.

10.7 Other Algorithms

There are much more efficient variants of the index calculus algorithm. Currently, the fastest index calculus algorithm is the number field sieve. It has running time $L_p[1/3, (64/9)^{1/3}]$ and was invented

shortly after the discovery of the number field sieve factoring algorithm.

DL records can be found in [24]. In September 2001 the solution $x = 262112280685811387636008622038191827370390768520656974243035380382193478767436018681449804940840373741641452864730765082$ of the discrete logarithm problem $y = g^x \bmod p$ with $p = \lfloor 10^{119}\pi \rfloor + 207819 = 3141592653589793238462643383279502884197169399375105820974944592307816406286208998628034825342117067982148086513284384483$, $g = 2$ and $y = \lfloor 10^{119}e \rfloor = 2718281828459045235360287471352662497757247093699959574966967627724076630353547594571382178525166427427466391932003059921$ was found in a 10-week computation.

Other efficient integer factoring algorithms also have DL variants. This shows that the integer factoring problem and the DL problem in finite fields are closely related. Therefore, cryptosystems based on the discrete logarithm problem in finite fields cannot really be considered to be an alternative to systems that are based on the difficulty of factoring integers. Real alternatives are the DL problem on elliptic curves or in algebraic number fields.

All DL problems that are relevant in the context of cryptography can be solved in polynomial time on quantum computers (see [67]). Again, it is unclear whether sufficiently large quantum computers can ever be built.

10.8 Generalization of the Index Calculus Algorithm

Although the baby-step giant-step algorithm and the Pollard ρ-algorithm work in any cyclic group, we have explained the index calculus algorithm only in multiplicative groups of residues modulo a prime number. But in principle, the index calculus algorithm also works in any group. Some factor base of group elements is fixed. Relations for this factor base are computed. The discrete logarithms are computed by linear algebra techniques. However, the factor base must be chosen such that relations can be found efficiently. Unfortu-

nately, for some groups, such as for elliptic curves over finite fields, it is not known how to choose the factor base and how to compute relations. Therefore, the index calculus algorithm is not applicable in those groups.

10.9 Exercises

Exercise 10.9.1
Solve the DL problem $3^x \equiv 693 \bmod 1823$ using the baby-step giant-step algorithm.

Exercise 10.9.2
Use the baby-step giant-step algorithm to compute the discrete logarithm of 15 to the base 2 mod 239.

Exercise 10.9.3
Solve the DL problem $a^x \equiv 507 \bmod 1117$ for the smallest primitive root a mod 1117 with the Pohlig-Hellman algorithm.

Exercise 10.9.4
Use the Pohlig-Hellman algorithm to compute the discrete logarithm of 2 to the base 3 mod 65537.

Exercise 10.9.5
Use the Pollard ρ-algorithm to solve the DL problem $g^x \equiv 15 \bmod 3167$ for the smallest primitive root g mod 3167.

Exercise 10.9.6
Use the variant of the Pollard ρ-algorithm that stores eight triplet (β, x, y) to solve the DL problem $g^x \equiv 15 \bmod 3167$ for the smallest primitive root g mod 3167. Compare the efficiency of this computation with the efficiency of the simple Pollard ρ-algorithm (Exercise 10.9.5).

Exercise 10.9.7
Use the index calculus algorithm with the factor base $\{2, 3, 5, 7, 11\}$ to solve $7^x \equiv 13 \bmod 2039$.

Exercise 10.9.8
Determine the smallest factor base that can be used in the index calculus algorithm to solve $7^x \equiv 13 \bmod 2039$.