with $127194 \neq \pm 45335 \bmod N$. Computing $\gcd(127194 - 45335, 377753) = 751$ yields a non-trivial factor of N. ♢

Running time. We have omitted many details in our discussion of the algorithm above. It can be shown, however, that with appropriate optimizations the quadratic sieve algorithm runs in time $2^{\mathcal{O}(\sqrt{n \cdot \log n})}$ to factor a number N of length $\mathcal{O}(n)$. The important point is that this running time is sub-exponential in the length of N.

8.2 Algorithms for Computing Discrete Logarithms

Let $\mathbb{G}$ be a group for which the group operation can be carried out efficiently. By the results of Section B.2.3, this means that exponentiation in $\mathbb{G}$ can also be done efficiently. An *instance* of the discrete logarithm problem takes the following form (see Section 7.3.2): given $g \in \mathbb{G}$ and $y \in \langle g \rangle$, find x such that $g^x = y$.[4] This answer is denoted by $\log_g y$, and is uniquely defined modulo the order of g. We sometimes refer to g in an instance of the discrete logarithm problem as the *base*.

Algorithms for attacking the discrete logarithm problem fall into two categories: those that work for *arbitrary* groups (such algorithms are sometimes termed *generic*) and those that work for some *specific* group. For algorithms of the former type, we can often just as well take the group to be $\langle g \rangle$ itself (thus ignoring elements in $\mathbb{G} \setminus \langle g \rangle$ when g is not a generator of $\mathbb{G}$). When doing so, we will let q denote the order of $\langle g \rangle$ and assume that q is known. Note that brute-force search for the discrete logarithm can be done in time $\mathcal{O}(q)$, and so we will only be interested in algorithms whose running time is better than this.

We will discuss the following algorithms that work in arbitrary groups:

- The *baby-step/giant-step* method, due to Shanks, computes the discrete logarithm in a group of order q in time $\mathcal{O}(\sqrt{q} \cdot \mathsf{polylog}(q))$.
- The *Pohlig-Hellman* algorithm can be used when the factorization of the group order q is known. When q has small factors, this technique reduces the given discrete logarithm instance to multiple instances of the discrete logarithm problem in groups of smaller order. Solutions to each of the latter can be combined to give the desired solution to the original problem.

[4] Recall that $\langle g \rangle$, the cyclic subgroup generated by g, is the subgroup $\{g^0, g^1, \ldots\} \subseteq \mathbb{G}$. If $\langle g \rangle = \mathbb{G}$ then g is a *generator* of $\mathbb{G}$ and $\mathbb{G}$ is cyclic.

We next look at computing discrete logarithms in some specific groups. As an illustrative but simple example, we first look at the problem in the (additive) group $\mathbb{Z}_N$ and show that discrete logarithms can be computed in polynomial time in this case. The point of this exercise is to demonstrate that

> *even though any cyclic group of order q is isomorphic to $\mathbb{Z}_q$ (cf. Example 7.58 in Chapter 7), and hence all cyclic groups of the same order are, in some sense, "the same", the hardness of the discrete logarithm problem depends in a crucial way on the particular* **representation** *being used for the group.*

Indeed, the algorithm for computing discrete logarithms in the *additive* group $\mathbb{Z}_N$ will rely on the fact that *multiplication* modulo N is also defined. Such a statement makes no sense in some arbitrary group that is defined without reference to modular arithmetic.

Turning to groups with more cryptographic significance, we briefly discuss the computation of discrete logarithms in the cyclic group $\mathbb{Z}_p^*$ for p prime. We give a high-level overview of the *index calculus method* that solves the discrete logarithm problem in such groups in sub-exponential time. The full details of this approach are, unfortunately, beyond the scope of this book.

The baby-step/giant-step algorithm is known to be *optimal* (in terms of its asymptotic running time) as far as generic algorithms go. (We remark, however, that more space-efficient generic algorithms with the same running time are known.) The proven lower bound on the complexity of finding discrete logarithms when the group is treated generically, however, says nothing about the hardness of finding discrete logarithms in any particular group.

Currently, the best-known algorithm for computing discrete logarithms in $\mathbb{Z}_p^*$ (for p prime) is the *general number field sieve*.[5] Heuristically, this algorithm runs in time $2^{\mathcal{O}(n^{1/3}\cdot(\log n)^{2/3})}$ on average to compute discrete logarithms in $\mathbb{Z}_p^*$ when p has length $\|p\| = \mathcal{O}(n)$. Importantly, essentially no non-generic algorithms are currently known for computing discrete logarithms in certain specially-constructed elliptic curve groups (cf. Section 7.3.4). This means that for such groups, as long as the group order is prime (so as to preclude the Pohlig-Hellman algorithm), only exponential-time algorithms for computing discrete logarithms are known.

To get a sense for the practical importance of this latter remark, we can compare the group sizes needed for each type of group in order to make the discrete logarithm problem equally hard. (This will be a rough comparison only, as a more careful comparison would, for starters, need to take into account the constants implicit in the big-$\mathcal{O}$ notation of the running times given above.) For a 512-bit prime p, the general number field sieve computes discrete

[5] It is no accident that the name of this algorithm and its running time are the same as for that of the currently best-known algorithm for factoring: they share many of the same underlying steps.

logarithms in $\mathbb{Z}_p^*$ in roughly $2^{512^{1/3} \cdot 9^{2/3}} \approx 2^{8 \cdot 4} = 2^{32}$ steps. This matches the time needed to compute discrete logarithms using the best generic algorithm in an elliptic curve group of order q, where q is a 64-bit prime, since then $\sqrt{q} \approx 2^{64/2} = 2^{32}$. We see that a significantly smaller elliptic curve group, with concomitantly faster group operations, can be used without reducing the difficulty of the discrete logarithm problem (at least with respect to the best currently-known techniques). Roughly speaking, then, by using elliptic curve groups in place of $\mathbb{Z}_p^*$ we obtain cryptographic schemes that are more efficient for the honest players, but that are equally hard for an adversary to break.

8.2.1 The Baby-Step/Giant-Step Algorithm

The baby-step/giant-step algorithm, due to Shanks, computes discrete logarithms in a group of order q in time $\mathcal{O}(\sqrt{q} \cdot \mathsf{polylog}(q))$. The idea is simple. Given as input g and $y \in \langle g \rangle$, we can imagine the elements of $\langle g \rangle$ laid out in a circle as

$$1 = g^0, g^1, g^2, \ldots, g^{q-2}, g^{q-1}, g^q = 1,$$

and we know that y must lie somewhere on this circle. Computing and writing down all the points on this circle would take at least $\Omega(q)$ time. Instead, we "mark off" the circle at intervals of size $t \stackrel{\text{def}}{=} \lfloor \sqrt{q} \rfloor$; that is, we compute and record the $\lfloor q/t \rfloor + 1 = \mathcal{O}(\sqrt{q})$ elements

$$g^0, g^t, g^{2t}, \ldots, g^{\lfloor q/t \rfloor \cdot t}.$$

(These are the "giant steps".) Note that the "gap" between any consecutive "marks" on the circle is at most t. Furthermore, we know that $y = g^x$ lies in one of these gaps. We are thus guaranteed that one of the t elements

$$y \cdot g^0 = g^x, \;\; y \cdot g^1 = g^{x+1}, \;\; \ldots, \;\; y \cdot g^t = g^{x+t},$$

will be equal to one of the points we have marked off. (These are the "baby steps".) Say $y \cdot g^i = g^{k \cdot t}$. We can easily solve this to obtain $y = g^{kt-i}$ or $\log_g y = [kt - i \bmod q]$. Psuedocode for this algorithm is given next.

The algorithm requires $\mathcal{O}(\sqrt{q})$ exponentiations and multiplications in $\mathbb{G}$, and each exponentiation can be done in time $\mathcal{O}(\mathsf{polylog}(q))$ using an efficient exponentiation algorithm. (Actually, other than the first value $g_1 = g^t$, each value g_i can be computed using a single multiplication as $g_i = g_{i-1} \cdot g_1$.) Sorting the $\mathcal{O}(\sqrt{q})$ pairs (i, g_i) can be done in time $\mathcal{O}(\sqrt{q} \cdot \log q)$, and we can then use binary search to check whether y_i is equal to some g_k in time $\mathcal{O}(\log q)$. The overall algorithm thus runs in time $\mathcal{O}(\sqrt{q} \cdot \mathsf{polylog}(q))$.

Example 8.6

We show an application of the algorithm in the cyclic group $\mathbb{Z}_{29}^*$ of order $q = 29 - 1 = 28$. Take $g = 2$ and $y = 17$. We set $t = 5$ and compute

$$2^0 = 1, \;\; 2^5 = 3, \;\; 2^{10} = 9, \;\; 2^{15} = 27, \;\; 2^{20} = 23, \;\; 2^{25} = 11.$$

ALGORITHM 8.5
The baby-step/giant-step algorithm

Input: Elements $g \in \mathbb{G}$ and $y \in \langle g \rangle$; the order q of g
Output: $\log_g y$

$t := \lfloor \sqrt{q} \rfloor$
for $i = 0$ to $\lfloor q/t \rfloor$:
  compute $g_i := g^{i \cdot t}$
sort the pairs (i, g_i) by their second component
for $i = 0$ to t:
  compute $y_i := y \cdot g^i$
  if $y_i = g_k$ for some k, **return** $[kt - i \bmod q]$

(We omit the "mod 29" since it is understood that operations are in the group $\mathbb{Z}_{29}^*$.) Then compute

$$17{\cdot}2^0 = 17, \quad 17{\cdot}2^1 = 5, \quad 17{\cdot}2^2 = 10, \quad 17{\cdot}2^3 = 20, \quad 17{\cdot}2^4 = 11, \quad 17{\cdot}2^5 = 22,$$

and notice that $2^{25} = 11 = 17 \cdot 2^4$. We thus have $\log_2 17 = 25 - 4 = 21$. $\diamondsuit$

8.2.2 The Pohlig-Hellman Algorithm

The Pohlig-Hellman algorithm can be used to speed up the computation of discrete logarithms when any non-trivial factors of the group order q are known. Recall that the order of an element g, which we denote here by $\mathsf{ord}(g)$, is the smallest positive i for which $g^i = 1$. We will need the following lemma:

LEMMA 8.7 *Let* $\mathsf{ord}(g) = q$*, and say* $p \mid q$*. Then* $\mathsf{ord}(g^p) = q/p$*.*

PROOF Since $(g^p)^{q/p} = g^q = 1$, the order of g^p is certainly at most q/p. Let $i > 0$ be such that $(g^p)^i = 1$. Then $g^{pi} = 1$ and, since q is the order of g, it must be the case that $pi \geq q$ or $i \geq q/p$. The order of g^p is therefore exactly q/p. ■

We will also use a generalization of the Chinese remainder theorem: if $q = \prod_{i=1}^{k} q_i$ and the $\{q_i\}$ are pairwise relatively prime (i.e., $\gcd(q_i, q_j) = 1$ for all $i \neq j$), then

$$\mathbb{Z}_q \simeq \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_k} \quad \text{and} \quad \mathbb{Z}_q^* \simeq \mathbb{Z}_{q_1}^* \times \cdots \times \mathbb{Z}_{q_k}^*.$$

(This can be proved by induction on k, using the basic Chinese remainder theorem as the base case.) Moreover, by an extension of the algorithm in Section 7.1.5 it is possible to convert efficiently between the representation of an element as an element of $\mathbb{Z}_q$ and its representation as an element of $\mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_k}$.

We now describe the Pohlig-Hellman approach. We are given g and y and are interested in finding an x such that $g^x = y$. Let $\mathsf{ord}(g) = q$, and say a factorization

$$q = \prod_{i=1}^{k} q_i$$

is known with the $\{q_i\}$ pairwise relatively prime. (Note that this need not be the complete prime factorization of q.) We know that

$$\left(g^{q/q_i}\right)^x = (g^x)^{q/q_i} = y^{q/q_i} \text{ for } i = 1, \ldots, k. \tag{8.4}$$

Letting $g_i \stackrel{\text{def}}{=} g^{q/q_i}$, we thus have k instances of a discrete logarithm problem in k *smaller* groups, each of size $\mathsf{ord}(g_i) = q_i$ (by Lemma 8.7).

We can solve each of the k resulting instances using any other algorithm for solving the discrete logarithm problem; for concreteness, let us assume that the baby-step/giant-step algorithm of the previous section is used. Solving these instances gives a set of answers $\{x_i\}_{i=1}^k$ for which $g_i^{x_i} = y^{q/q_i} = g_i^x$. (The second equality follows from Equation (8.4).) Proposition 7.50 implies that $x = x_i \bmod q_i$ for all i. By the generalized Chinese remainder theorem discussed earlier, the constraints

$$\begin{aligned} x &= x_1 \bmod q_1 \\ &\vdots \\ x &= x_k \bmod q_k \end{aligned}$$

uniquely determine x modulo q. (This is of course the best we can hope for, since the equation $g^x = y$ only uniquely determines x modulo q.) The answer x itself can be efficiently reconstructed from $x_1, \ldots, x_k$.

Example 8.8

We again apply the ideas introduced here to compute a discrete logarithm in $\mathbb{Z}_p^*$. Here, take $p = 31$ with the order of $\mathbb{Z}_{31}^*$ being $q = 31 - 1 = 30 = 5 \cdot 3 \cdot 2$. Say $g = 3$ and $y = 26 = g^x$. We have

$$\begin{aligned} (g^x)^{30/5} = y^{30/5} &\Rightarrow (3^6)^x = 16^x = 26^6 = 1 \\ (g^x)^{30/3} = y^{30/3} &\Rightarrow (3^{10})^x = 25^x = 26^{10} = 5 \\ (g^x)^{30/2} = y^{30/2} &\Rightarrow (3^{15})^x = 30^x = 26^{15} = 30. \end{aligned}$$

(Once again, we omit the "mod 31" since this is understood.) Solving each equation, we obtain

$$x = 0 \bmod 5, \quad x = 2 \bmod 3, \quad x = 1 \bmod 2,$$

and so $x = 5 \bmod 30$. Indeed, $3^5 = 26 \bmod 31$. ♢

Assuming q with factorization as above, and assuming the baby-step/giant-step algorithm is used to solve each of the smaller instances of the discrete logarithm problem, the running time of the entire algorithm will be $\mathcal{O}(\mathsf{polylog}(q) \cdot \sum_{i=1}^{k} \sqrt{q_i})$. Since q can have at most $\log q$ factors, this simplifies to $\mathcal{O}(\mathsf{polylog}(q) \cdot \max_i\{\sqrt{q_i}\})$. Depending on the size of the largest known factor of q, this can be a marked improvement over the $\mathcal{O}(\sqrt{q})$ algorithm given in the previous section. In particular, if q has many small factors then the discrete logarithm problem in a group of order q will be relatively easy to solve via this approach. As discussed in Section 7.3.2, this motivates choosing q to be prime for cryptographic applications.

If q has prime factorization $q = \prod_{i=1}^{k} p_i^{e_i}$, the Pohlig-Hellman algorithm as described above solves the discrete logarithm in a group of order q in time $\mathcal{O}\left(\mathsf{polylog}(q) \cdot \max_i\{\sqrt{p_i^{e_i}}\}\right)$. Using additional ideas, this can be improved to $\mathcal{O}\left(\mathsf{polylog}(q) \cdot \max_i\{\sqrt{p_i}\}\right)$; see Exercise 8.2.

8.2.3 The Discrete Logarithm Problem in $\mathbb{Z}_N$

The algorithms shown in the preceding two sections are *generic*, in the sense that they are oblivious to the underlying group in which the discrete logarithm problem is defined (except for knowledge of the group order). The purpose of this brief section is merely to emphasize that non-generic algorithms, which make use of the particular (representation of the) group under consideration, can potentially perform much better.

Consider the task of computing discrete logarithms in the (additive) group $\mathbb{Z}_N$ for arbitrary N. The problem is trivial with respect to the base $g = 1$: the discrete logarithm of element $y \in \mathbb{Z}_N$ is simply the integer y itself since $y \cdot 1 = y \bmod N$. Note that, formally speaking, the 'y' on the left-hand side of this equation denotes the integer y while the 'y' on the right-hand side denotes the element $y \in \mathbb{Z}_N$. Nevertheless, the particular nature of the group $\mathbb{Z}_N$ allows us to essentially view these two instances of 'y' interchangeably.

Things are only mildly more complicated if a generator $g \neq 1$ is used. (Exercise 8.3 deals with the case when g is not a generator of $\mathbb{Z}_N$.) Let $g \in \mathbb{Z}_N$ be a generator and say we want to compute x such that $x \cdot g = y \bmod N$ for some given value y. Using Theorem B.18 (along with the fact that 1 is a generator), we have $\gcd(g, N) = 1$. But then g has a multiplicative inverse g^{-1} modulo N (and this inverse can be computed efficiently as discussed in Appendix B.2.2). The desired solution is simply $x = y \cdot g^{-1} \bmod N$.

It is interesting to pinpoint once again exactly what non-generic properties of $\mathbb{Z}_N$ are being used here. In this case, the algorithm implicitly uses the fact that an operation (namely, multiplication modulo N) *other than* the group operation (i.e., addition modulo N) is defined on the elements of the group.

8.2.4 The Index Calculus Method

The index calculus method solves the discrete logarithm problem in the cyclic group $\mathbb{Z}_p^*$ (for p prime) in time that is sub-exponential in the length of p. The astute reader may notice that the algorithm as we will describe it bears some resemblance to the quadratic sieve factoring algorithm introduced in Section 8.1.3. As in the case of that algorithm, we will discuss the main ideas used by the index calculus method but leave the details beyond the scope of our treatment. Also, some small changes are made in order to simplify the presentation.

The index calculus method uses a two-stage process. Importantly, the first stage requires knowledge only of the modulus p and the base g and so it can be run as a 'pre-processing step' before y is known. For the same reason, it suffices to run the first stage only once in order to solve multiple instances of the discrete logarithm problem (as long as all these instances share the same p and g).

Step 1. Let $q = p - 1$, the order of $\mathbb{Z}_p^*$. Fix a set $B = \{p_1, \ldots, p_k\}$ of small prime numbers. In this stage, we find $\ell \geq k$ distinct, non-zero values $x_1, \ldots, x_\ell \in \mathbb{Z}_q$ for which $g_i \stackrel{\text{def}}{=} g^{x_i} \bmod p$ is "small", so that g_i can be factored over the integers (using, e.g., trial division) and such that all the prime factors of g_i lie in B. We do not discuss how these $\{x_i\}$ are found.

Following this step, we have ℓ equations of the form:

$$\begin{aligned} g^{x_1} &= \prod_{i=1}^{k} p_i^{e_{1,i}} \bmod p \\ &\vdots \\ g^{x_\ell} &= \prod_{i=1}^{k} p_i^{e_{\ell,i}} \bmod p. \end{aligned}$$

Taking discrete logarithms, we can transform these into linear equations:

$$\begin{aligned} x_1 &= \sum_{i=1}^{k} e_{1,i} \cdot \log_g p_i \mod (p-1) \\ &\vdots \\ x_\ell &= \sum_{i=1}^{k} e_{\ell,i} \cdot \log_g p_i \mod (p-1). \end{aligned} \tag{8.5}$$

Note that the $\{x_i\}$ and the $\{e_{i,j}\}$ are known, while the $\{\log_g p_i\}$ are unknown.

Step 2. Now we are given an element y and want to compute $\log_g y$. Here, we find a value $x^* \in \mathbb{Z}_q$ for which $g^* \stackrel{\text{def}}{=} g^{x^*} \cdot y \bmod p$ is "small", so that g^*

can be factored over the integers and such that all the prime factors of g^* lie in B. We do not discuss how x^* is found.

Say

$$g^{x^*} \cdot y = \prod_{i=1}^{k} p_i^{e_i^*} \bmod p$$
$$\Rightarrow x^* + \log_g y = \sum_{i=1}^{k} e_i^* \cdot \log_g p_i \mod (p-1),$$

where x^* and the $\{e_i^*\}$ are known. Combined with Equation (8.5), we have $\ell + 1 \geq k + 1$ linear equations in the $k + 1$ unknowns $\{\log_g p_i\}_{i=1}^k$ and $\log_g y$. Using linear algebraic[6] methods (and assuming the system of equations is not under-defined), we can solve for each of the unknowns and in particular solve for the desired solution $\log_g y$.

Example 8.9

Let $p = 101$, $g = 3$, and $y = 87$. We have $3^{10} = 65 \bmod 101$, and $65 = 5 \cdot 13$ (over the integers). Similarly, $3^{12} = 80 = 2^4 \cdot 5 \bmod 101$ and $3^{14} = 13 \bmod 101$. That is,

$$\begin{aligned} 10 &= \log_3 5 + \log_3 13 \mod 100 \\ 12 &= 4 \cdot \log_3 2 + \log_3 5 \mod 100 \\ 14 &= \log_3 13 \mod 100. \end{aligned}$$

We also have $3^5 \cdot 87 = 32 = 2^5 \bmod 101$, or

$$5 + \log_3 87 = 5 \cdot \log_3 2 \mod 100. \tag{8.6}$$

Using simple algebraic manipulation, we first derive $4 \cdot \log_3 2 = 16 \bmod 100$. This doesn't determine $\log_3 2$ uniquely, but it does tell us that $\log_3 2 = 4, 29, 54,$ or 79 (cf. Exercise 8.3). Trying all possibilities shows that $\log_3 2 = 29$. Plugging this into Equation (8.6) gives $\log_3 87 = 40$. ♢

Running time. It can be shown that with appropriate optimizations the index calculus algorithm runs in time $2^{\mathcal{O}(\sqrt{n \cdot \log n})}$ to compute discrete logarithms in $\mathbb{Z}_p^*$ for p a prime of length n. The important point is that this is sub-exponential in $\|p\|$. Note that the expression for the running time is identical to that for the quadratic sieve method.

[6] Technically, things are slightly more complicated since the linear equations are all modulo $p - 1$, which is not prime. Nevertheless, there exist techniques for dealing with this.

References and Additional Reading

The texts by Wagstaff [127], Shoup [117], Crandall and Pomerance [43], and Bressoud [33] all provide further discussion of algorithms for factoring and computing discrete logarithms, and the latter two books are highly recommended for the reader wishing to understand the state-of-the-art.

Lower bounds on so-called *generic algorithms* for computing discrete logarithms (i.e., algorithms that apply to arbitrary groups without regard for the way the group is represented) are given by Nechaev [100] and Shoup [114].

Exercises

8.1 Here we show how to solve the discrete logarithm problem in a cyclic group of order $q = p^e$ in time $\mathcal{O}(\mathsf{polylog}(q) \cdot \sqrt{p}\,)$. We are given as input a generator g of known order p^e and a value y, and want to compute $x = \log_g y$. Note that p can be computed easily from q (see Exercise 7.9 in Chapter 7).

(a) Show how to find $x \bmod p$ in time $\mathcal{O}(\mathsf{polylog}(q) \cdot \sqrt{p}\,)$.
Hint: Solve the equation
$$\left(g^{p^{e-1}}\right)^{x_0} = y^{p^{e-1}}$$
and use the same ideas as in the Pohlig-Hellman algorithm.

(b) Say $x = x_0 + x_1 \cdot p + \cdots + x_{e-1} \cdot p^{e-1}$ with $0 \le x_i < p$. In the previous step we determined x_0. Show how to compute in $\mathsf{polylog}(q)$ time a value y_1 such that $(g^p)^{x_1 + x_2 \cdot p + \cdots + x_{e-1} \cdot p^{e-2}} = y_1$.

(c) Use recursion to obtain the claimed running time for the original problem. (Note that $e = \mathsf{polylog}(q)$.)

8.2 Let q have prime factorization $q = \prod_{i=1}^{k} p_i^{e_i}$. Using the result from the previous problem, show a modification of the Pohlig-Hellman algorithm that solves the discrete logarithm problem in a group of order q in time $\mathcal{O}\left(\mathsf{polylog}(q) \cdot \sum_{i=1}^{k} e_i \sqrt{p_i}\right) = \mathcal{O}\left(\mathsf{polylog}(q) \cdot \max\{\sqrt{p_i}\,\}\right)$.

8.3 (a) Show that if $ab = c \bmod N$ and $\gcd(b, N) = d$, then:
i. $d \mid c$;
ii. $a \cdot (b/d) = (c/d) \bmod (N/d)$; and
iii. $\gcd(b/d, N/d) = 1$.

(b) Describe how to use the above to compute discrete logarithms in $\mathbb{Z}_N$ efficiently even when the base g is not a generator of $\mathbb{Z}_N$.