6.1.3 Generic Algorithms for Discrete Logarithms

Several of the algorithms we have described for the discrete logarithm problem can be applied to any cyclic group. Such algorithms are called *generic*, because they do not depend on any property of the representation of the group. Examples of generic algorithms for the discrete logarithm problem are Shank's algorithm, the Pollard rho algorithm and the Pohlig-Hellman algorithm. Algorithms which are not generic include the Extended Euclidean algorithm (for finding discrete logs in $(\mathbb{Z}_n, +)$) and the index calculus algorithm (for finding logs in $({\mathbb{Z}_p}^*, \cdot)$ for p prime). These non-generic algorithms are more efficient than generic ones; however, they can be applied only in certain special situations.

The Shank's and rho algorithms have the property that their complexity (in terms of the number of groups operations required to run the algorithm) is roughly $\sqrt{n}$, where n is the order of the (sub)-group in which the discrete lograrithm is being sought. In this section, we will provide a proof of a result of Shoup which says that these algorithms are essentially optimal within the class of generic algorithms.

We begin by giving a precise description of what we mean by a generic algorithm. We consider a cyclic group or subgroup of order n, which is therefore isomorphic to $(\mathbb{Z}_n, +)$. An *encoding* of $(\mathbb{Z}_n, +)$ is any injective mapping $\sigma : \mathbb{Z}_n \rightarrow S$, where S is a finite set. The encoding function specifies how group elements are represented.

Any discrete logarithm problem in a (sub)group of cardinality n can be specified by defining a suitable encoding function. For example, consider the multiplicative group $({\mathbb{Z}_p}^*, \cdot)$, and let α be a primitive element in ${\mathbb{Z}_p}^*$. Let $n = p - 1$, and define the encoding function σ as follows: $\sigma(i) = \alpha^i \bmod p$, $0 \leq i \leq n - 1$. Then it should be clear that solving the DLP in $({\mathbb{Z}_p}^*, \cdot)$ with respect to the primitive element α is equivalent to solving the DLP in $(\mathbb{Z}_n, +)$ with generator 1 under the encoding σ.

A generic algorithm is one that works for any encoding. In particular, a generic algorithm must work correctly when the encoding function σ is a random injective function; for example, when $S = \mathbb{Z}_n$ and σ is a random permutation of $\mathbb{Z}_n$. This is very similar to the random oracle model, in which a hash function is regarded as a random function in order to define an idealized model in which a formal security proof can be given.

We suppose that we have a random encoding, σ, for the group $(\mathbb{Z}_n, +)$. In this group, the discrete logarithm of any element a to the base 1 is just a, of course. Given the encoding function σ, the encoding $\sigma(1)$ of the generator, and an encoding of an arbitrary group element $\sigma(a)$, a generic algorithm is trying to compute the value of a. In order to perform operations in this group when group elements are encoded using the function σ, we hypothesize the existence of a an oracle (or subroutine) to perform this task.

Given encodings of two group elements, say $\sigma(i)$ and $\sigma(j)$, it should be possible to compute the encodings $\sigma((i+j) \bmod n)$ and $\sigma((i-j) \bmod n)$. This is necessary if we are going to add and subtract group elements, and we assume that our oracle will do this for us. By combining operations of the above type, it is possible to compute arbitrary linear combinations of the form $\sigma((ci \pm dj) \bmod n)$, where $c, d \in \mathbb{Z}_n$. However, using the fact that $-j \equiv n - j \pmod{n}$, we observe that we only need to be able to compute linear combinations of the form $\sigma((ci + dj) \bmod n)$. We will assume that the oracle can directly compute linear combinations of this form.

Group operations of the type described above are the only ones allowed in a generic algorithm. That is, we assume that we have some method of performing group operations on encoded elements, but we cannot do any more than that. Now let us consider how a generic algorithm, say GENLOG, can go about trying to compute a discrete logarithm. The input to the algorithm GENLOG consists of $\sigma_1 = \sigma(1)$ and $\sigma_2 = \sigma(a)$, where $a \in \mathbb{Z}_n$ is chosen randomly. GENLOG will be successful if and only if it outputs the value a. (We will assume that n is prime, in order to simplify the analysis.)

GENLOG will use the oracle to generate a sequence of m encodings of linear combinations of 1 and a. The execution of GENLOG can be specified by a list of ordered pairs $(c_i, d_i) \in \mathbb{Z}_n \times \mathbb{Z}_n$, $1 \leq m$. (We can assume that these m ordered pairs are distinct.) For each ordered pair (c_i, d_i), the oracle computes the encoding $\sigma_i = \sigma((c_i + d_i a) \bmod n)$. Note that we can define $(c_1, d_1) = (1, 0)$ and $(c_2, d_2) = (0, 1)$ so our notation is consistent with the input to the algorithm.

In this way, the algorithm GENLOG obtains a list of encoded group elements, $(\sigma_1, \ldots, \sigma_m)$. Because the encoding function σ is injective, it follows immediately that $c_i + d_i a \equiv c_j + d_j a \pmod{n}$ if and only if $\sigma_i = \sigma_j$. This provides a method to possibly compute the value of the unknown a: Suppose that $\sigma_i = \sigma_j$ for two integers $i \neq j$. If $d_i = d_j$, then $c_i = c_j$ and the two ordered pairs (c_i, d_i) and (c_j, d_j) are the same. Since we are assuming the ordered pairs are distinct, it follows that $d_i \neq d_j$. Because n is prime, we can compute a as follows:

$$a = (c_i - c_j)(d_j - d_i)^{-1} \bmod n.$$

(Recall that we used a similar method of computing the value of a discrete logarithm in the Pollard rho algorithm.)

Suppose first that the algorithm GENLOG chooses a set

$$\mathcal{C} = \{(c_i, d_i) : 1 \leq i \leq m\} \subseteq \mathbb{Z}_n \times \mathbb{Z}_n$$

of m distinct ordered pairs all at once, at the beginning of the algorithm. Such an algorithm is called a *non-adaptive* algorithm (Shank's algorithm is an example of a non-adaptive algorithm). Then the list of m corresponding encodings is obtained from the oracle. Define $\mathsf{Good}(\mathcal{C})$ to consist of all elements $a \in \mathbb{Z}_n$ that are the solution of an equation $a = (c_i - c_j)(d_j - d_i)^{-1} \bmod n$ with $i \neq j$, $i, j \in \{1, \ldots, m\}$. By what we have said above, we know that the value of a can be computed by GENLOG if and only if $a \in \mathsf{Good}(\mathcal{C})$. It is clear that $|\mathsf{Good}(\mathcal{C})| \leq \binom{m}{2}$, so there are at most $\binom{m}{2}$ elements for which GENLOG can compute the discrete logarithm after having obtained a sequence of m encoded group elements corresponding to the ordered pairs in $\mathcal{C}$. The probability that $a \in \mathsf{Good}(\mathcal{C})$ is at most $\binom{m}{2}/n$. If $a \notin \mathsf{Good}(\mathcal{C})$, then the best strategy for the algorithm GENLOG is to guess the value of a by choosing a random value in $\mathbb{Z}_n \backslash \mathsf{Good}(\mathcal{C})$. Denote $g = |\mathsf{Good}(\mathcal{C})|$. Then, by conditioning on whether or not $a \in \mathsf{Good}(\mathcal{C})$, can compute a bound on the success probability of the algorithm. Suppose we define the following random variables: $\mathbf{a}$ denotes the event $a \in \mathsf{Good}(\mathcal{C})$; and $\mathbf{b}$ denotes the event "the algorithm returns the correct value of a". Then we have that

$$\begin{aligned} p(\mathbf{b}) &= p(\mathbf{b}|\mathbf{a})p(\mathbf{a}) + p(\mathbf{b}|\overline{\mathbf{a}})p(\overline{\mathbf{a}}) \\ &= 1 \times \frac{g}{n} + \frac{1}{n-g} \times \frac{n-g}{n} \\ &= \frac{g+1}{n} \\ &\leq \frac{\binom{m}{2} + 1}{n}. \end{aligned}$$

A discrete logarithm algorithm is not required to choose all the ordered pairs in $\mathcal{C}$ at the beginning of the algorithm, of course. It can choose later pairs after seeing what encodings of previous linear combinations look like (i.e., we allow the algorithm to be *adaptive*). However, we will show, using an inductive argument, that this does not improve the success probabilty of the algorithm.

Now let GENLOG be an adaptive generic algorithm for the discrete logarithm problem. For $1 \leq i \leq m$, let $\mathcal{C}_i$ consist of the first i ordered pairs, for which the oracle computes the corresponding encodings $\sigma_1, \ldots, \sigma_i$. The set $\mathcal{C}_i$ and the list $\sigma_1, \ldots, \sigma_i$ represent all the information available to GENLOG at time i of its execution.

We claim that the value of a can be computed at time i if $a \in \mathsf{Good}(\mathcal{C}_i)$; and if $a \notin \mathsf{Good}(\mathcal{C}_i)$, then the probability that a has any given value in the set $\mathbb{Z}_n \backslash \mathsf{Good}(\mathcal{C}_i)$ is exactly $1/(n - |\mathsf{Good}(\mathcal{C}_i)|)$. (This probability is the conditional probability of a value of a at time i, assuming that a is chosen randomly from $\mathbb{Z}_n$.)

This claim is true for $i = 1$ (which forms the base case for an inductive proof) because $\mathcal{C}_1 = \{(1, 0)\}$ by definition, and $\mathsf{Good}(\mathcal{C}_1) = \emptyset$. If the claim is true for $i = j - 1$, then it is easy to see that it is also true for $i = j$. Then the claimed result follows by induction. When $i = m$, the claim establishes that the success probability of the algorithm is at most $(\binom{m}{2} + 1)/n$, exactly as in the case of a non-adaptive algorithm.

If GENLOG is guaranteed to compute the desired discrete logarithm, then its success probability is equal to 1. Then the result proved above says that $(\binom{m}{2} + 1)/n \geq 1$, or $m^2 + m \geq 2n$. Hence n is $\Omega(\sqrt{n})$; this provides a lower bound on the complexity of any generic algorithm for the discrete log problem in a (sub)group of prime order n.