Lower Bounds for Discrete Logarithms and Related Problems*

Victor Shoup

IBM Research–Zürich, Säumerstr. 4, 8803 Rüschlikon, Switzerland
sho@zurich.ibm.com

Abstract. This paper considers the computational complexity of the discrete logarithm and related problems in the context of "generic algorithms"—that is, algorithms which do not exploit any special properties of the encodings of group elements, other than the property that each group element is encoded as a unique binary string. Lower bounds on the complexity of these problems are proved that match the known upper bounds: any generic algorithm must perform $\Omega(p^{1/2})$ group operations, where p is the largest prime dividing the order of the group. Also, a new method for correcting a faulty Diffie-Hellman oracle is presented.

1 Introduction

The discrete logarithm problem plays an important role in cryptography. The problem is this: given a generator g of a cyclic group G, and an element g^x in G, determine x. A related problem is the Diffie-Hellman problem: given g^x and g^y, determine g^{xy}.

In this paper, we study the computational power of "generic algorithms"—that is, algorithms which do not exploit any special properties of the encodings of group elements, other than the property that each group element is encoded as a unique binary string. For the discrete logarithm problem, as well as several other related problems, including the Diffie-Hellman problem, we present lower bounds that match the known upper bounds for these problems. We also give a new method for correcting a faulty Diffie-Hellman oracle.

Generic Algorithms

Let $\mathbf{Z}/n$ be the additive group of integers mod n, and let S be a set of bit strings of cardinality at least n. An *encoding function* of $\mathbf{Z}/n$ on S is an injective map σ from $\mathbf{Z}/n$ into S.

A generic algorithm A for $\mathbf{Z}/n$ on S is a probabilistic algorithm that behaves as follows. It takes as input an *encoding list* $(\sigma(x_1), \ldots, \sigma(x_k))$, where each x_i is in $\mathbf{Z}/n$, and σ is an encoding function of $\mathbf{Z}/n$ on S. As the algorithm executes, it may from time to time consult an *oracle*, specifying two indices i and j into the encoding list, and a sign bit. The oracle computes $\sigma(x_i \pm x_j)$, according to the

* This is a revised version of the paper in *Proc. Eurocrypt '97*.

specified sign bit, and this bit string is appended to the encoding list (to which A always has access). The output of A is a bit string denoted $A(\sigma; x_1, \ldots, x_k)$.

Note that the algorithm A depends on n and S, but not on σ; information about σ is only available to A through the oracle.

To measure the running time of such an algorithm, we count both the number of bit operations, and the number of group operations (i.e., oracle queries).

It is readily seen that the classical Pohlig-Hellman algorithm [8] is a generic algorithm. Let p denote the largest prime divisor of n. Assuming the strings in S have a length that is polynomial in $\log n$, this algorithm has a running time of $p^{1/2}(\log n)^{O(1)}$, and this bound holds uniformly for all possible encoding functions. Note that this algorithm makes essential use of the fact that group elements are uniquely encoded as bit strings, which facilitates the use of fast sorting-and-searching techniques.

Pollard's discrete logarithm algorithm [9] also falls into this generic class. This algorithm is much more space efficient than the Pohlig-Hellman algorithm, but its efficiency relies on the heuristic assumption that the encoding function behaves like a random mapping.

As an example, consider the multiplicative group $(\mathbf{Z}/q)^*$ for a prime q, together with a generator g for this group. Here, $n = q - 1$, and the relevant encoding function sends $a \in \mathbf{Z}/n$ to the binary encoding of $g^a \bmod q$.

Of course, not all algorithms for the discrete logarithm problem are generic. Index-calculus methods for $(\mathbf{Z}/q)^*$, for example, do not fall in this category, and our results have no bearing on such algorithms. For groups associated with elliptic curves, however, the only known algorithms for discrete logarithms are generic. Our results imply that for elliptic curves, one cannot substantially improve upon the Pohlig-Hellman algorithm using generic algorithms: some method must be devised to exploit the particular representation of group elements.

Summary of Results

In §2 we consider the discrete logarithm problem. Theorem 1 says that any generic algorithm that solves (with high probability) the discrete logarithm problem on $\mathbf{Z}/n$ must perform at least $\Omega(p^{1/2})$ group operations, where p is the largest prime dividing n. The theorem shows that for any algorithm, there must be an encoding function for which it makes $\Omega(p^{1/2})$ queries to the group oracle; we do this by showing that this must hold for a *random* encoding function, and a *random* input.

Theorem 2 deals with the analog for the discrete logarithm problem in non-cyclic groups, which was suggested to the author by Buchmann [3]. Suppose G is the product of r cyclic groups of prime order p. Then any generic algorithm that (with high probability) expresses a given element on a given basis for G must perform at least $\Omega(p^{r/2})$ group operations.

In §3 we consider the Diffie-Hellman problem. Theorem 3 proves the analog of Theorem 1 for the Diffie-Hellman problem.

Theorem 4 shows that if the group order is divisible by only large primes, then it is hard to simply determine which of two possible solutions is correct.

Theorem 5 deals with the problem of solving the Diffie-Hellman problem in subgroups. Suppose we are given an oracle for solving the Diffie-Hellman problem in a group G, and now want to solve the Diffie-Hellman problem in a proper subgroup H. This problem is interesting, as it plays an important role in Maurer's [5] and Boneh and Lipton's [2] reductions from the discrete logarithm problem to the Diffie-Hellman problem: they require Diffie-Hellman oracles for prime-order subgroups. Theorem 5 implies that in the context of generic algorithms, there are situations where the oracle for G does not help at all in solving the problem in H.

In §4 we consider the security of an identification scheme due to Schnorr [10] based on the discrete logarithm problem. While this scheme is known to be secure against "passive" attacks, its security against "active" attacks is not well understood. Theorem 6 shows that this scheme is indeed secure against active attacks when the adversary is a generic algorithm.

In §5 we consider a quite different problem: given a faulty oracle for the Diffie-Hellman problem, how to make it highly reliable? One reason that this problem is interesting is that the reductions of Maurer and Boneh/Lipton mentioned above require reliable oracles. That is, these reductions say that if Diffie-Hellman is "easy," then the discrete logarithm is "easy." However, in proving the security of a cryptosystem based on the Diffie-Hellman problem, one normally assumes that this problem is "hard." The above reductions do not allow one to directly weaken this to an assumption that the discrete logarithm is "hard": that a problem is not "hard" does not imply that it is "easy". For this, one must solve precisely the problem we address: making a faulty oracle reliable.

In light of our Theorem 4, standard techniques for amplifying correctness do not apply to the Diffie-Hellman problem. Theorem 7 and its corollary show how to efficiently turn an oracle that is occasionally correct into one that is almost always correct. The theorem is also useful in the application of the Goldreich-Levin theorem to hard bits of the Diffie-Hellman problem.

Related Work

Babai and Szemerédi [1] proved lower bounds in a "black box" model in which the encoding of group elements is not necessarily unique, and the group oracle must be consulted to test for equality. For a cyclic group of order n, if p is the largest prime divisor of n, their results give an $\Omega(p)$ lower bound. Note that the Pohlig-Hellman algorithm does not work in this model.

More recently, Nechaev [7] considered algorithms for the discrete logarithm problem in the following computational model: an algorithm is allowed to perform group operations and equality tests, but no other operations on group elements are allowed—the notion of encodings of elements does not enter into this model at all. In Nechaev's model, equality testing is *free*, and only the number of group operations are counted. Nechaev proves an $\Omega(p^{1/2})$ lower bound on the number of group operations.

One can view our results as an extension of Nechaev's results to a broader and more natural class of algorithms, and to a wider range of problems related to the discrete logarithm problem.

It is perhaps worthwhile to point out the precise differences between Nechaev's model and our proposed model of generic algorithms. First, the class of generic algorithms clearly covers all algorithms covered by Nechaev's model. Second, there are algorithms, such as Pollard's algorithm (mentioned above) that are covered by the class of generic algorithms, but apparently not by Nechaev's. Indeed, all algorithms in Nechaev's model are subject to an $O(p^{1/2})$ lower bound on *space* as well as time. However, it is possible to transform any generic algorithm running on a random encoding function into an algorithm in Nechaev's model. This transformation can be accomplished by means of a simulator that preserves time, but not space—the simulator must store the representations of all group elements computed by the algorithm.

For the problem of correcting a faulty Diffie-Hellman oracle, Maurer and Wolf [6] independently devised a scheme based on techniques quite different from ours. It seems that our scheme is substantially simpler and more efficient than theirs.

2 The Discrete Logarithm Problem

The main result of this section is the following.

Theorem 1 *Let n be a positive integer whose largest prime divisor is p. Let $S \subset \{0,1\}^*$ be a set of cardinality at least n. Let A be a generic algorithm for $\mathbf{Z}/n$ on S that makes at most m oracle queries. If $x \in \mathbf{Z}/n$ and an encoding function σ are chosen at random, then the probability that $A(\sigma; 1, x) = x$ is $O(m^2/p)$.*

Note that the above probability is taken over the random choices of σ and x, as well as the coin flips of A. The theorem implies that for any algorithm, there exists an encoding function σ for which it succeeds with probability $O(m^2/p)$, taking the probability over x and the coin flips of A. If we insist that A succeed with probability bounded away from 0 by a constant, this translates into a lower bound of $\Omega(p^{1/2})$ on the number of group operations.

To prove this and several other theorems, we need the following lemma.

Lemma 1 *Let p be prime and let $t \geq 1$. Let $F(X_1, \ldots, X_k) \in \mathbf{Z}/p^t[X_1, \ldots, X_k]$ be a nonzero polynomial of total degree d. Then for random $x_1, \ldots, x_k \in \mathbf{Z}/p^t$, the probability that $F(x_1, \ldots, x_k) = 0$ is at most d/p.*

Proof. For $t = 1$, this is proved in Schwartz [11]. For $t > 1$, one divides the equation $F = 0$ by the highest possible power of p, and obtains a nonzero equation of no greater degree that holds modulo p. If $x_1, \ldots, x_k$ are chosen from $\mathbf{Z}/p^t$ at random, then their images in $\mathbf{Z}/p$ are random as well, and so we can apply the result for $t = 1$. $\square$

We now sketch the proof of Theorem 1. Let $n = p^t s$, where $(p, s) = 1$. Instead of letting the algorithm interact with the actual oracle, we play the following game. Let X be an indeterminant. At any step in the game, the algorithm has computed a list $F_1, \ldots, F_k$ of linear polynomials in $\mathbf{Z}/p^t[X]$, along with a list $z_1, \ldots, z_k$ of values in $\mathbf{Z}/s$, and a list $\sigma_1, \ldots, \sigma_k$ of *distinct* values in S. At the beginning of the game, $k = 2$; $F_1 = 1$ and $F_2 = X$; $z_1 = 1$ and z_2 is chosen at random; σ_1 and σ_2 are chosen at random, subject to $\sigma_1 \neq \sigma_2$. When the oracle is given two indices i and j, we append new values $F_{k+1}, z_{k+1}, \sigma_{k+1}$ to the appropriate lists as follows. We compute $F_{k+1} = F_i \pm F_j \in \mathbf{Z}/p^t[X]$ and $z_{k+1} = z_i \pm z_j \in \mathbf{Z}/s$. If $F_{k+1} = F_l$ and $z_{k+1} = z_l$ for some l with $1 \leq l \leq k$, we set $\sigma_{k+1} = \sigma_l$; otherwise, we set σ_{k+1} to a random element in S distinct from $\sigma_1, \ldots, \sigma_k$.

When the algorithm terminates, it outputs some $y \in \mathbf{Z}/n$. Let y' be the image of y in $\mathbf{Z}/p^t$. Now we choose a random $x \in \mathbf{Z}/p^t$. We say the algorithm wins the game if $F_i(x) = F_j(x)$ for any $F_i \neq F_j$ or if $x = y'$.

Fix i, j with $F_i \neq F_j$, and set $F = F_i - F_j$. Now, since $F \neq 0$, and $\deg F \leq 1$, then by Lemma 1, the probability that $F(x) = 0$ is at most $1/p$. Likewise, the probability that $x = y'$ is at most $1/p$. It follows that the probability that the algorithm wins the above game is $O(m^2/p)$.

To finish the proof, one must only observe that the behavior of this game differs from an actual interaction between the algorithm and oracle only when the algorithm wins the above game. Therefore, the probability that the algorithm outputs the correct answer is bounded by the probability that the algorithm wins the above game.

To make the above argument completely rigorous, one can easily construct a single probability space that is shared by both the actual interaction and the above game, such that

(1) the shared probability space does not change the behavior of either the actual interaction or the above game, and
(2) in this shared space, the event that A outputs the correct answer in the actual interaction is contained in the event that A wins the above game.

The details of this are quite straightforward, and are omitted. That completes the proof of Theorem 1.

We now consider a variation of the discrete logarithm problem that applies to non-cyclic groups.

Suppose that $G = \mathbf{Z}/p \times \cdots \times \mathbf{Z}/p$ is the product of r cyclic groups of order p, where p is prime. The input consists of the encodings of the unit vectors $e_1, \ldots, e_r$, along with the encoding of an element $(x_1, \ldots, x_r) \in G$. The output should be $(x_1, \ldots, x_r)$. Here, an encoding function is an injective map σ from G into some set S of at least p^r bit strings. The following theorem establishes an $\Omega(p^{r/2})$ lower bound for this problem with respect to generic algorithms. Note that a simple generalization of the Pohlig-Hellman algorithm gives a matching upper bound.

Theorem 2 *Let A be a generic algorithm for G on S for the above problem that makes at most m oracle queries. If $(x_1, \ldots, x_r) \in G$ and an encoding function σ*

are chosen at random, then the probability that

$$A(\sigma; e_1, \ldots, e_r, (x_1, \ldots, x_r)) = (x_1, \ldots, x_r)$$

is $O(m^2/p^r)$.

The proof is similar to that of Theorem 1. We sketch the differences. Let $X_1, \ldots, X_r$ be indeterminants. We play the same game as before, but instead of a list of polynomials, we maintain a list of r-tuples, each of which has the form

$$(aX_1 + b_1, aX_2 + b_2, \ldots, aX_r + b_r),$$

where $a, b_1, \ldots, b_r \in \mathbf{Z}/p$. The key observation is that when we add or subtract (component-wise) two r-tuples of this form, we get an r-tuple of the same form. Also, by Lemma 1, the probability that a nonzero r-tuple of this form vanishes when $X_1, \ldots, X_r$ are substituted with random values is at most $1/p^r$. The rest of the proof goes as before.

One can easily extend the above theorem to an arbitrary finite abelian group $G = \mathbf{Z}/n_1 \times \cdots \times \mathbf{Z}/n_r$, obtaining a lower bound of $\Omega(p^{k/2})$, where p is a prime and k is the number of moduli n_i divisible by p.

3 The Diffie-Hellman Problem

In this section, we prove a lower bound for the Diffie-Hellman problem.

Theorem 3 *Let n be a positive integer whose largest prime divisor is p. Let $S \subset \{0,1\}^*$ be a set of cardinality at least n. Let A be a generic algorithm for $\mathbf{Z}/n$ on S that makes at most m oracle queries. If $x, y \in \mathbf{Z}/n$ and an encoding function σ are chosen at random, then the probability that $A(\sigma; 1, x, y) = \sigma(xy)$ is $O(m^2/p)$.*

The proof of this is similar to that of Theorem 1. We may assume that the output of A is one of the encodings obtained from the oracle, since otherwise the success probability is bounded by $1/(p - m)$. We play precisely the same game as there, except that we maintain a list of polynomials F_i in the variables X, Y over $\mathbf{Z}/p^t$, where each polynomial has total degree 1. When the algorithm terminates, we pick $x, y \in \mathbf{Z}/p^t$ at random, and we say that the algorithm wins the game if $F_i(x, y) = F_j(x, y)$ for some $F_i \neq F_j$, or if $F_i(x, y) = xy$ for some i. Applying Lemma 1, for fixed i, j, the probability that $F_i - F_j$ vanishes is at most $1/p$, and for fixed i, the probability that $F_i - XY$ vanishes is at most $2/p$. It follows that the probability that the algorithm wins this game is $O(m^2/p)$.

That completes the proof of Theorem 3.

When n is divisible by only large primes, just determining which of two possible answers is the correct one is hard.

Theorem 4 *Let n be a positive integer whose* smallest *prime divisor is p. Let $S \subset \{0,1\}^*$ be a set of cardinality at least n. Let A be a generic algorithm for $\mathbf{Z}/n$ on S that makes at most m oracle queries. Let $x, y, z \in \mathbf{Z}/n$ be chosen at random, let σ be a random encoding function, and let b be a random bit. Also, let $w_0 = xy$ and $w_1 = z$. Then the probability that $A(\sigma; 1, x, y, w_b, w_{1-b}) = b$ is $1/2 + O(m^2/p)$.*

We sketch the proof. We play a similar game as before, this time maintaining a list of polynomials $F_i(X, Y, U, V)$ over $\mathbf{Z}/n$ of total degree 1, assigning to each distinct polynomial a distinct random encoding. We say the algorithm wins the game if for any $F_i \neq F_j$, we have $F_i(x, y, xy, z) = F_j(x, y, xy, z)$, or $F_i(x, y, z, xy) = F_j(x, y, z, xy)$. For a fixed $F_i \neq F_j$, the polynomial $F_i - F_j$ must be nonzero modulo some prime power q^t that exactly divides n. Since the images x, y, and z in $\mathbf{Z}/q^t$ are also uniformly distributed, by Lemma 1, the above condition holds with probability at most $4/q \leq 4/p$. Thus, the probability that the algorithm wins the game is $O(m^2/p)$. Moreover, it is clear that in the actual interaction between the algorithm and the oracle, the probability that the algorithm determines b is bounded by $1/2$ plus the probability that the algorithm wins the above game.

We close this section with a look at the following question. Suppose we have a cyclic group G, and we have an oracle for the Diffie-Hellman problem in G. Can we use this oracle to solve the Diffie-Hellman problem efficiently in a proper subgroup H? It is not difficult to see that if $(|H|, |G|/|H|)$ is divisible only by small primes, then this problem can be solved efficiently. More specifically, if p is the largest prime dividing $(|H|, |G|/|H|)$, the problem can be solved in time $p^{1/2}(\log n)^{O(1)}$. The following theorem shows that this bound is essentially tight, and thus for large p the problem can not be solved efficiently using a generic algorithm.

To study this problem, we extend the notion of a generic algorithm so as to include a Diffie-Hellman oracle: given indices i and j, the oracle computes $\sigma(x_i \cdot x_j)$. The output of such an algorithm A is denoted by $A_{DH}(\sigma; x_1, \ldots, x_k)$.

Theorem 5 *Let n be a positive integer, and let l be a divisor of n such that for some prime p, $l = l'p^s$, $n = n'p^t$, and $t > s > 0$. Let $S \subset \{0,1\}^*$ be a set of cardinality at least n. Let A be a generic algorithm for $\mathbf{Z}/n$ on S that makes at most m oracle queries. If $x \in \mathbf{Z}/n$ and an encoding function σ are chosen at random, then the probability that $A_{DH}(\sigma; 1, lx, ly) = \sigma(lxy)$ is $O((t/s) \cdot m^2/p)$.*

We sketch the proof in the case $l' = n' = 1$. The more general case is dealt with as in Theorem 1. Let $d = \lceil t/s \rceil - 1$. We play the usual game, this time maintaining a list of polynomials $F_i(X, Y)$ in the variables X and Y over $\mathbf{Z}/p^t$, each of which has the form

$$\sum_{k=0}^{d} p^{sk} \sum_{i+j=k} a_{ij} X^i Y^j.$$

The key observation is that when we add, subtract, *or even multiply* two polynomials of this form, we get a polynomial that is also of this form. When the algorithm terminates, we select $x, y \in \mathbf{Z}/p^t$ at random, and the algorithm wins the game if for some $F_i \neq F_j$, $F_i(x, y) = F_j(x, y)$ or for some i, $F_i(x, y) = p^s xy$. By Lemma 1, this happens with probability at most $O(dm^2/p)$.

4 Analysis of an Identification Scheme

An identification scheme is an interactive protocol that allows one party P to prove its identity to another party V. To do this, P has a public key, which is known to all parties, and a private key, which is known only to himself.

Such a scheme is considered secure if an adversary can not feasibly make V believe it is conducting the protocol with P. One can allow the adversary to first interact with P, pretending to be V (but not not necessarily following V's protocol), in order to gain some information about P's secret key that will be of use in its impersonation attempt. Such an attack is called "active." An attack where no prior interaction with P is allowed is called "passive." Clearly, security against active attacks is preferable to security against passive attacks.

An identification scheme due to Schnorr [10] runs as follows. Let G be a cyclic group of order n, with a publicly known generator g. P's private key is an element $x \in \mathbf{Z}/n$, and its public key is $h = g^x$. The value x is randomly chosen. In the first step of the protocol, P generates $r \in \mathbf{Z}/n$ at random, computes $h' = g^r$, and sends h' to V. Upon receiving h', V chooses $e \in \mathbf{Z}/n$ at random, and sends e to P. Upon receiving e, P computes $y = r + xe \in \mathbf{Z}/n$ and sends y to V. Upon receiving y, V checks that $g^y = h'h^e$. If this identity holds, V accepts; otherwise, V rejects.

In his paper, Schnorr shows that this protocol is secure against passive attacks, assuming the discrete logarithm is hard. We prove that the scheme is secure against a active adversary that behaves as a generic algorithm.

Theorem 6 *Consider the above identification scheme in a generic setting; that is, there is an encoding function σ mapping elements of $\mathbf{Z}/n$ into a set S of bit strings. Suppose that the adversary makes no more than m interactions with P or queries to the group oracle, and that σ is chosen at random. Suppose also that the private key x is chosen at random. Then the probability that the adversary successfully impersonates P is $O(m^2/p)$, where p is the largest prime dividing n.*

The above probability is taken over σ, x, and the coin tosses of all of the players. In proving this theorem, we allow the adversary to interact with several instances of P in parallel—we do not require that one interaction ends before the next one begins.

We sketch the proof for $n = p$; the more general case is dealt with as in Theorem 1.

We use the same type of game argument that we used in proving the other theorems, but with a few changes. In this game, we maintain a list of degree 1 polynomials $F_i(X, R_1, R_2, \ldots, R_m)$ in $m + 1$ variables over $\mathbf{Z}/p$, corresponding to the group elements the adversary has seen so far, along with a corresponding list of random encodings.

Initially, the list of polynomials contains the two polynomials 1 and X. Whenever the adversary starts an interaction with P for the kth time, we add the polynomial R_k to the polynomial list, and a distinct random encoding to the list of encodings. Whenever the adversary consults the group oracle, we add to the polynomial list the sum of the appropriate polynomials; we either either re-use an encoding or generate a distinct random encoding, as appropriate.

Now suppose the adversary sends a challenge e to the lth instance of P. In our game, we do the following: we choose $y \in \mathbf{Z}/p$ at random, and send y to the adversary as the response from P; we also go through our list of polynomials and substitute $y - eX$ for the variable R_l wherever it appears. If upon making this substitution any two distinct polynomials in the list become equal, we quit and we say the adversary wins. Otherwise, we continue the game.

Now suppose the adversary attempts an impersonation. Without loss of generality, we may assume the adversary has completed all interactions with P that it started. So it has collected a list $F_1, \ldots, F_{m+2} \in \mathbf{Z}/p[X]$ of polynomials along with a list of encodings. In the first step of the protocol, the adversary presents the encoding of some group element corresponding to one of these polynomials, say F_l. Next V chooses $e \in \mathbf{Z}/p$ at random. If $F_l + eX$ is a constant polynomial, we quit and say the adversary wins. Otherwise, the adversary chooses $y \in \mathbf{Z}/p$. Finally, we choose $x \in \mathbf{Z}/p$ at random, and we say that the adversary wins if $y = F_l(x) + ex$ or if $F_i(x) = F_j(x)$ for any $F_i \neq F_j$.

That completes the description of the game. First observe that the behavior of this game deviates from that of the actual interaction only if the adversary wins the game. So it suffices to bound the probability that the adversary wins the game. It is relatively straightforward to show that this is $O(m^2/p)$. One observation to bear in mind is the following. When making a substitution $y - eX$ for a variable R_k, one need only count pairs of polynomials $F_i \neq F_j$ such that $F_i - F_j \in \mathbf{Z}/p[X, R_k]$ and the coefficient of R_k is nonzero. But note that if we count this pair when substituting for R_k, we will not count this pair when we later make a substitution for some other R_l. Thus, the total number of pairs we need to count is $O(m^2)$.

5 A Diffie-Hellman Self-Corrector

In this section, we consider the following problem. Let G be a cyclic group of order n with generator g. Suppose we have a "faulty" oracle for the Diffie-Hellman problem; that is, given g^a and g^b, the oracle outputs g^c, such that $c \equiv ab \pmod{n}$ with probability at least ϵ. We take this probability to be over the random choice of a and b mod n, and any coin tosses of the oracle. Here, ϵ is small, but nonnegligible. The problem is to use this oracle to build an efficient algorithm for the Diffie-Hellman problem whose output is almost certainly correct for all inputs. One motivation for this problem is again the reductions of [5] and [2] from the discrete logarithm problem to the Diffie-Hellman problem; these reductions require a nearly-perfect oracle—a faulty oracle will simply not do.

Given such an oracle, using the standard random self-reduction, we can run it $O(1/\epsilon)$ times so that with high probability one of its outputs is correct. However, as we have seen, in the generic model we have no hope of determining which output is correct.

We consider the following, more general, problem. We define a (k, δ) *Diffie-Hellman oracle* as follows: for all inputs g^a, g^b, it produces a list of k elements in G such that this list contains g^{ab} with probability at least δ. The problem is to use this oracle to solve the Diffie-Hellman problem.

Another situation in which this type of oracle arises is in the hard-bit construction of Goldreich and Levin [4], where a bit-predicting oracle can be turned into this type of oracle.

Theorem 7 *Given a (k, δ) Diffie-Hellman oracle with $\delta > 7/8$, we can construct a probabilistic (generic) algorithm for the Diffie-Hellman problem with the following properties. For given α, with $0 < \alpha < 1$, the algorithm makes $O(\log(1/\alpha))$ queries to the (k, δ) oracle, and performs an additional $O(\log(1/\alpha)k \log n + (\log n)^2)$ group operations. For all inputs, the output of the algorithm is correct with probability at least $1 - \alpha$.*

As an immediate corollary, we have:

Corollary 1 *Given a faulty Diffie-Hellman oracle that has a success probability of ϵ, we can construct a probabilistic algorithm for the Diffie-Hellman problem with the following properties. For given α, with $0 < \alpha < 1$, the algorithm makes $O(\epsilon^{-1} \log(1/\alpha))$ queries to the faulty oracle, and performs an additional $O(\epsilon^{-1} \log(1/\alpha) \log n + (\log n)^2)$ group operations. For all inputs, the output of the algorithm is correct with probability at least $1 - \alpha$.*

To prove Theorem 7, we assume that n is known, and that for all prime factors p of n, $k^2/p < 1/8$. If this does not hold, we can partially factor n, and apply the Pohlig-Hellman algorithm to the "smooth" part. A straightforward calculation shows that this takes $O(k \log n + (\log n)^2)$ group operations. So we can assume that n is of the desired form.

For given g^a, g^b, the following algorithm either reports failure, or outputs a single value g^c. The algorithm makes two queries to the (k, δ) oracle and performs an additional $O(k \log n)$ group operations. The probability that it reports failure is at most $3/8$. The conditional probability that $g^c \neq g^{ab}$, given that it does not report failure, is $2/7$. The Diffie-Hellman algorithm then simply runs the above algorithm $O(\log(1/\alpha))$ times, taking the majority of the non-failure outputs.

We call the (k, δ) oracle twice, first with g^a, g^b, obtaining a list $g_1, \ldots, g_k$ of group elements. Next, we choose $x, y \in \{0, \ldots, n-1\}$ at random, and send $(g^a)^x g^y, g^b$ to the (k, δ) oracle, obtaining a list $g'_1, \ldots, g'_k$ of group elements. Next, for all $1 \leq i \leq k$ and $1 \leq j \leq k$, we test if

$$g_i^x (g^b)^y = g'_j. \tag{1}$$

If (1) is satisfied for a unique pair (g_i, g'_j), we output g_i; otherwise, we report failure. Note that standard sorting-and-searching techniques can be used to make this last step efficient.

The claimed running-time bound is easily verified. We now analyze its correctness. Let $z = ax + y$. Fix i and j, and suppose $g_i = g^c$ and $g'_j = g^d$. Suppose $c \equiv ab \pmod{n}$. Then (1) is satisfied if and only if and $d \equiv zb \pmod{n}$. Now suppose $c \not\equiv ab \pmod{n}$. Then for some prime power p^t that exactly divides n, we must have $c \not\equiv ab \pmod{p^t}$. In this case, the probability that (1) holds is at most the conditional probability that for random $x, y \bmod p^t$, $cx + by \equiv d \pmod{p^t}$, given that $ax + y = z \pmod{p^t}$. This is equal to the probability that for fixed z and random x, $(c - ab)x + bz - d \equiv 0 \pmod{p^t}$, which is by Lemma 1 at most $1/p$.

There are three mutually exclusive events of interest: the algorithm either (F) reports failure, (I) produces an incorrect output, or (C) produces a correct output.

$\Pr[F] + \Pr[I]$ is bounded by the probability that one of the lists does not contain a correct output, or that any extraneous relations (1) hold. This happens with probability at most $1/8 + 1/8 + k^2/p \leq 3/8$.

$\Pr[I]$ is bounded by the probability that one of the lists does not contain a correct output. This is because if both lists contain a correct output, any extraneous relations (1) that hold will cause the algorithm to report failure. This probability is thus bounded by $1/8 + 1/8 = 1/4$.

It trivially follows that $\Pr[F]$ is bounded by $3/8$. Moreover, by a simple calculation, $\Pr[I]/(\Pr[I] + \Pr[C])$ is bounded by $2/7$.

References

1. L. Babai and E. Szemerédi. On the complexity of matrix group problems I. In *25th Annual Symposium on Foundations of Computer Science*, pages 229–240, 1984.
2. D. Boneh and R. J. Lipton. Algorithms for black-box fields and their application to cryptography. In *Advances in Cryptology–Crypto '96*, pages 283–297, 1996.
3. J. Buchmann, 1995. Personal communication.
4. O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *21st Annual ACM Symposium on Theory of Computing*, pages 25–32, 1989.
5. U. Maurer. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms. In *Advances in Cryptology–Crypto '94*, pages 271–281, 1994.
6. U. Maurer and S. Wolf. Diffie-Hellman oracles. In *Advances in Cryptology–Crypto '96*, pages 268–282, 1996.
7. V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994. Translated from *Matematicheskie Zametki*, 55(2):91–101, 1994.
8. S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Trans. Inf. Theory*, 24:106–110, 1978.
9. J. M. Pollard. Monte Carlo methods for index computation mod p. *Mathematics of Computation*, 32:918–924, 1978.

10. C. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4:161–174, 1991.
11. J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.