The Generic Complexity of Index-Search Problems and Applications to Cryptography*

Ueli Maurer Stefan Wolf

Computer Science Department
Swiss Federal Institute of Technology (ETH Zürich)
CH-8092 Zürich, Switzerland
E-mail addresses: {maurer,wolf}@inf.ethz.ch

March 25, 1998

Abstract

This paper is concerned with the complexity of generic algorithms solving so-called index-search problems such as the discrete logarithm problem in a cyclic group. The index-search problem was defined as follows: given a list $(a_i)_{i\in I}$ of elements of some set S, and given an element b of S, find an $i \in I$ such that $b = a_i$. The notion of a generic algorithm on the other hand was introduced by Shoup. Intuitively, a generic algorithm is a general-purpose algorithm which does not exploit any particular property of the representation of the objects, for instance of the group elements. By using purely information-theoretic arguments, a general lower bound on the complexity of generic index-search problems is proved. Some implications of this result are the optimality of the baby-step giant-step algorithm in many situations, Shoup's result concerning the hardness of the generic discrete logarithm problem, and a complete characterization of groups for which breaking the Diffie-Hellman key-distribution protocol is computationally equivalent in a generic sense to computing discrete logarithms in the same group: this holds exactly for groups whose order is *not* divisible by a large multiple prime factor.

Keywords. Cryptography, Diffie-Hellman protocol, discrete logarithms, index-search problem, generic algorithms, complexity, lower bounds.

1 Introduction

This paper is concerned with lower bounds on the complexity of generic algorithms for solving the so-called *generalized index-search problem.* The index-search problem was defined in [5] as follows. Let $(a_i)_{i\in I}$ be a list of elements of some set S. The index-search problem is, given $b \in S$, to find an index $i \in I$ such that $b = a_i$.

The model of generic algorithms was introduced by Shoup in [9] for cyclic groups. Intuitively, a generic algorithm for a cyclic group is an algorithm that does not make use of any particular property of the representation of the group elements. The model of [9] can be generalized to the index-search problem as follows. Here, a generic algorithm does not exploit any particular property of the representation of the elements of S or more precisely, of any relationship between an element $b = a_i$ of S and its index i. Formally, let S be a set of cardinality n (for example a

*This work was supported by the Swiss National Science Foundation (SNF), grant No. 20–42105.94.

set of arbitrary binary strings), and let a bijective encoding function

$$\sigma : \{0, \ldots, n-1\} \longrightarrow S$$

be chosen randomly from the set of all such functions. Then a generic algorithm for solving the index-search problem takes as input $\sigma(x)$ (for some randomly chosen x) and should output x. The algorithm is allowed to make calls to an oracle that computes the function σ, i.e., that takes as input $c \in \mathbf{Z}_n$ and outputs $\sigma(c)$. In the case of the *generalized* index-search problem, we additionally allow the algorithm to make calls to oracles computing $\sigma(g(x_1, \ldots, x_r))$ from $\sigma(x_1), \ldots, \sigma(x_r)$ for certain fixed functions g. We will restrict ourselves to the case of *polynomial* functions g, assuming the natural ring structure on the index set $\{0, \ldots, n-1\}$.

In Section 2 we prove a general lower bound on the complexity of algorithms solving the generalized index-search problem. We will take a purely information-theoretic viewpoint. The proofs are based on the fact that a generic algorithm can reduce the uncertainty about x (when given $\sigma(x)$) only by testing equality of expressions $\sigma(f(x))$ for certain functions f. In Section 3, we apply the result of Section 2 to certain specific index-search problems such as the generic discrete logarithm problem for cyclic groups, in particular when additionally an oracle solving the Diffie-Hellman problem is given for the same group. One conclusion is that in a generic sense, the Diffie-Hellman problem and the discrete logarithm problem are not equivalent for all groups.

2 A lower bound on the generalized index-search problem

In this section we prove a lower bound on the complexity of generic algorithms solving the generalized index-search problem. The proof is an extension of the technique used by Shoup in [9]. We assume that the algorithm is allowed to make calls to oracles computing

$$\sigma(x_1), \ldots, \sigma(x_{s_i}) \rightsquigarrow \sigma(P_i(x_1, \ldots, x_{s_i}))$$

for some fixed polynomials P_i and $i = 1, \ldots, l$. Theorem 1 gives an upper bound on the success probability of such an algorithm. This bound depends on the running time of the algorithm, on the factorization of n, and on the maximal degree of a polynomial P_i.

Theorem 1 *Let for the generalized index-search problem oracles be given for the computation of $\sigma(P_i(x_1, \ldots, x_{s_i}))$ from the values $\sigma(x_1), \ldots, \sigma(x_{s_i})$, where $P_i(X_1, \ldots, X_{s_i})$ is a polynomial in $\mathbf{Z}_n[X_1, \ldots, X_{s_i}]$ of total degree d_i for all $i = 1, \ldots, l$, and where $l \geq 1$. Then every probabilistic algorithm that runs in time at most T and takes as input $\sigma(x)$ answers with x with probability α (taken over random choices of x and σ) at most*

$$\alpha \leq \frac{(T+1)T}{2} \cdot \min\left\{ \frac{(\max\{d_i\})^T}{p} \; , \; \frac{1}{q} \right\} + \frac{1}{n} \; ,$$

where p and q are the largest and the largest multiple *prime factor of n, respectively. In the case where no oracle is given, i.e., $l = 0$, we have $\alpha \leq (T+1)/n$.*

Remark. Note that the bound in Theorem 1 only depends on the maximal total degree occurring. We could also have assumed that the algorithm can make calls to an oracle computing *arbitrary* polynomials (in certain numbers of variables) of degree at most d for some $d \geq 1$.

For the proof we need the following lemma (see [8] or [9]).

Lemma 2 *When given a polynomial $P(X_1, \ldots, X_k)$ over $\mathbf{Z}_{p^t}$ of total degree d, the probability that $P(x_1, \ldots, x_k) = 0$ for random $x_1, \ldots, x_k \in \mathbf{Z}_{p^t}$ is at most d/p.*

Proof of Theorem 1. Let us first consider the case where at least one additional oracle is given, i.e., $l \geq 1$. Before the algorithm has performed a call to one of the oracles, we have[1]

$$H(x \mid \sigma(x)) = \log n$$

or in other words,

$$I(x\,;\,\sigma(x)) = 0\ .$$

This holds because x and the encoding function σ are randomly chosen[2].

By interaction with the given oracles T times, the algorithm can compute a list of expressions $\sigma(f_j(x))$ for $j = 1, \ldots, T+1$ with $f_1(X) = X$ and for all $j \geq 2$ either $f_j(X) = c$ for some constant $c \in \mathbf{Z}_n$ or $f_j(X) = P_i(f_{j_1}(X), \ldots, f_{j_{s_i}}(X))$ for some i and $j_1, \ldots, j_{s_i} < j$. Let $\mathcal{E}$ be the event that $f_j(x) = f_{j'}(x)$ for some $j \neq j'$, $1 \leq j, j' \leq T+1$, and let $\overline{\mathcal{E}}$ be the complementary event. We have

$$I(x\,;\,[\sigma(x), \sigma(f_2(x)), \ldots, \sigma(f_{T+1}(x))] \mid \overline{\mathcal{E}}) = 0\ . \tag{1}$$

Equality (1) states that if all the encoded elements obtained previously are distinct, then the particular encodings of the elements are statistically independent of x. This follows from the fact that the encoding function σ is chosen randomly among all possible encodings of $\{0, \ldots, n-1\}$ with respect to S.

We now derive two upper bounds on the probability of the event $\mathcal{E}$. More precisely, these are bounds that hold for all possible strategies of the algorithm, including adaptive strategies. These bounds depend on the largest prime factor and on the largest *multiple* prime factor of n, respectively.

Let first p be a prime factor n, i.e., $n = p^e s$ with $\gcd(s, p) = 1$ and $e \geq 1$. Determining x modulo n means determining x modulo p^e and modulo s. We can assume without loss of generality that x modulo s is easy to obtain (of course this ability cannot increase the running time of the algorithm) and hence that $n = p^e$. According to Lemma 2 the number of $y \in \mathbf{Z}_n$ such that $f_j(y) = f_{j'}(y)$ for fixed $j \neq j'$ is at most dp^{e-1}, where d is an upper bound on the degrees of f_j and $f_{j'}$. Thus the number of $y \in \mathbf{Z}_n$ such that $f_j(y) = f_{j'}(y)$ for *some* $j \neq j'$ is at most

$$\frac{(T+1)T \cdot (\max\{d_i\})^T p^{e-1}}{2}\ . \tag{2}$$

Hence the probability of the event $\mathcal{E}$ is upper bounded by

$$\mathrm{P}[\mathcal{E}] \leq \frac{(T+1)T \cdot (\max\{d_i\})^T}{2p}\ . \tag{3}$$

Let now q be a *multiple* prime factor of n, i.e., $n = q^e s$ with $\gcd(s, q) = 1$ and $e \geq 2$. As above, we can assume $n = q^e$. Let $x \equiv x_0 + x_1 q + \ldots + x_{e-1} q^{e-1} \pmod{q^e}$ be the q-adic expansion of x modulo q^e. We give a lower bound on the complexity of the computation of x_{e-1}. We can assume $x \equiv x_{e-1} q^{e-1} \pmod{q^e}$. By interacting T times with the given oracles, the algorithm can this time compute $\sigma(f_j(x_{e-1}))$ for $j = 1, \ldots, T+1$ with $f_1(X) = X \cdot q^{e-1}$ and for $j \geq 2$ either $f_j(X) = c$ for some constant $c \in \mathbf{Z}_n$ or $f_j(X) = P_i(f_{j_1}(X), \ldots, f_{j_{s_i}}(X))$ for some i and $j_1, \ldots, j_{s_i} < j$. The crucial fact now is that $f_j(X)$ is a *linear* polynomial in X for all j. This follows by induction and from the fact that $q^e \equiv 0 \pmod{n}$. Hence the number of $y \in \mathbf{Z}_n$ such that $f_j(y) = f_{j'}(y)$ for some $j \neq j'$ is at most

$$\frac{(T+1)Tq^{e-1}}{2}\ , \tag{4}$$

[1] All logarithms in this paper are to the base 2. Here, H and I are the usual entropy and mutual information functions.

[2] Note that x and σ stand for the random variables, and not for the particular values occurring.

and the probability of the event $\mathcal{E}$ is upper bounded by

$$\mathrm{P}[\mathcal{E}] \leq \frac{(T+1)T}{2q} \ . \tag{5}$$

Let now p be the largest and q be the largest *multiple* prime factor of n. From (2) and (4) we conclude that the number of $y \in \mathbf{Z}_n$ such that $f_j(y) = f_{j'}(y)$ for some $j \neq j'$ is at most

$$N := \frac{(T+1)T}{2} \cdot \min\left\{ \frac{(\max\{d_i\})^T}{p} \ , \ \frac{1}{q} \right\} \cdot n \ ,$$

and that

$$\mathrm{P}[\mathcal{E}] \leq \frac{N}{n} = \frac{(T+1)T}{2} \cdot \min\left\{ \frac{(\max\{d_i\})^T}{p} \ , \ \frac{1}{q} \right\} \ .$$

Given that the event $\overline{\mathcal{E}}$ occurs, equation (1) implies that the success probability of the algorithm is upper bounded by $1/(n-N)$. We conclude that the success probability α of the algorithm satisfies

$$\alpha \leq \mathrm{P}[\mathcal{E}] + \mathrm{P}[\overline{\mathcal{E}}] \cdot \frac{1}{n-N} \leq \frac{N+1}{n} \ .$$

Let us finally consider the special case where no oracle for the computation of a function P_i is given. Then the only computations of the algorithm which are *not* statistically independent of σ are to compute

$$\sigma(c_1), \sigma(c_2), \ldots, \sigma(c_T)$$

for certain $c_1, \ldots, c_T$ in time T. With probability at least $1 - T/n$, we have $c_j \neq x$ for all $j = 1, \ldots, T$, and hence

$$H\left(x \,|\, \sigma(x), (c_1, \sigma(c_1)), \ldots, (c_T, \sigma(c_T))\right) = \log(n-T)$$

because σ is completely random. This means that the success probability α of the algorithm is at most

$$\alpha \leq \frac{T}{n} + \left(1 - \frac{T}{n}\right) \cdot \frac{1}{n-T} = \frac{T+1}{n} \ .$$

This concludes the proof of the theorem. □

The bound of Theorem 1 on the success probability of a probabilistic algorithm is in terms of the *worst-case* running time of the algorithm. The following corollary, which is an immediate consequence of Theorem 1, gives a lower bound in terms of the *expected* running time of the algorithm.

Corollary 3 *Under the assumptions of Theorem 1, the running time T of any probabilistic algorithm that solves the generalized index-search problem must satisfy*

$$(2T+1)T \cdot \min\left\{ \frac{(\max\{d_i\})^{2T}}{p} \ , \ \frac{1}{q} \right\} \geq \frac{1}{2} - \frac{1}{n} \ .$$

Proof. Assume that a probabilistic algorithm exists which solves the problem in expected time T. We can construct a new algorithm by stopping the execution after time $2T$. This new algorithm has worst-case running time $2T$ and success probability at least $1/2$. The statement follows from Theorem 1, applied to the new algorithm. □

3 Applications to special index-search problems

In this section we apply the results of Section 2 to certain particular index-search problems and show that the bound of Theorem 1 is tight in many cases. These special scenarios are the index-search problem when a cyclic permutation of the index set is computable by an oracle, the discrete logarithm problem for a cyclic group, and the same problem when given an oracle solving the Diffie-Hellman problem for this group.

3.1 The special index-search problem and the baby-step giant-step algorithm

In case of the "pure" index-search problem where no additional oracle besides the oracle computing σ is given, it is clear that the bound $\alpha \leq (T+1)/n$ of Corollary 3 is tight because random brute force search (and random guessing unless $\sigma(c_i)$, $i = 1, \ldots, T$, is equal to $\sigma(x)$) exactly matches this bound.

Let us now consider the case where an additional oracle is given that computes a fixed cyclic permutation π of $\{0, \ldots, n-1\}$. More precisely, the oracle computes

$$\sigma(y) \rightsquigarrow \sigma(\pi(y)) \ .$$

We can assume without loss of generality that this permutation is

$$\pi(y) :\equiv y + 1 \pmod{n} \ .$$

An example for this situation is the computation of discrete logarithms in a cyclic group G with generator g in which it is possible to apply the group operation to an arbitrary group element and g (but not to two arbitrary group elements). In this case the bound of Corollary 3 on the running time of an algorithm solving the index-search problem is of order $\Theta(\sqrt{p})$, where p is the largest prime factor of n.

The so-called *baby-step giant-step algorithm* is applicable in this situation and has a running time of $O(\sqrt{n} \log n)$. The algorithm works as follows. First, the expressions

$$\sigma(x), \sigma(x+1), \ldots, \sigma(x+M)$$

are computed and stored in sorted order for $M := \lceil \sqrt{n} \rceil$. The sorting requires time $O(\sqrt{n} \log n)$. Then

$$\sigma(0), \sigma(M), \sigma(2M), \ldots$$

are computed (by calls to the oracle computing σ) until one of the expressions (say $\sigma(iM)$) is contained in the list $(\sigma(x+j))$. Then we have $x \equiv iM - j \pmod{n}$. The running time of this algorithm is $O(\sqrt{n} \log n)$ and memory space for storing $\sqrt{n}$ pairs $(\sigma(y), y) \in S \times \mathbf{Z}_n$ is required. Hence the bound given by Corollary 3 is asymptotically tight in this case at least if n is a prime number.

3.2 The discrete logarithm problem and the Diffie-Hellman problem

Let G be a finite cyclic group with generator g. The security of the well-known Diffie-Hellman (DH) protocol [3] relies on the difficulty of the following computational problem, the so-called *Diffie-Hellman (DH) problem*: Given two group elements g^x and g^y, compute g^{xy}. Of course this problem is at most as hard as the *discrete logarithm (DL) problem*: Given g^x, compute x.

Shoup [9] proved lower bounds on the complexity of generic algorithms solving the DH problem or the DL problem. Of course the generic DL problem corresponds to the generalized index-search problem, where the algorithm is allowed to make calls to an oracle computing

$$\sigma(x), \sigma(y) \rightsquigarrow \sigma(x \pm y) \ .$$

Shoup's result states that no general-purpose algorithm can solve the DH or the DL problem in groups G of order n substantially faster than in time $\Theta(\sqrt{p})$, where p is the largest prime factor of n. This is exactly the bound that results also from Corollary 3 when all the polynomial functions the oracle can compute are linear. Unlike in the model of Section 3.1, this bound is tight in the case of the DL and DH problems. The Pohlig-Hellman decomposition [7], which allows to reduce the DL problem in G to the same problem in the nontrivial minimal subgroups of G, together with the baby-step giant-step algorithm, achieves this complexity. Surprisingly, this algorithm even works for the generalized index-search problem when oracles are given only for computing $\sigma(y) \rightsquigarrow \sigma(y+1)$ and for $\sigma(y) \rightsquigarrow \sigma(2y)$.

However, for a *particular* group (or a class of groups such as elliptic curves over a finite field), the result of Shoup does *not* imply anything on the security of the DH protocol. On the other hand, an efficient generic *reduction algorithm* from the DL problem to the DH problem proves the equivalence of the two problems *for every particular group* of a certain order. The construction of such reduction algorithms was intensively studied with the objective to prove the equivalence of the DH and DL problems for all groups [2],[4],[5],[1]. We will see that the application of Corollary 3 to this situation shows that the two problems are *not* computationally equivalent in a generic sense for all groups.

The problem of computing discrete logarithms in groups G of order n when given a DH oracle, i.e., an oracle solving the DH problem, for these groups is the generalized index-search problem where oracles for the functions

$$\sigma(x), \sigma(y) \rightsquigarrow \sigma(x \pm y)$$

and

$$\sigma(x), \sigma(y) \rightsquigarrow \sigma(x \cdot y)$$

are given. Thus Corollary 3 implies the following lower bound.

Corollary 4 *The complexity T of a generic algorithm that reduces the DL problem to the DH problem for all groups G of order n satisfies*

$$T = \Omega(\log p)$$

and

$$T = \Omega(\sqrt{q}) \ ,$$

where p and q are the largest and the largest multiple *prime factor of n, respectively.*

Both bounds in Corollary 4 are asymptotically tight. In [2] and [4] (see also [5] and [1]), generic reduction algorithms from the DL problem to the DH problem were presented which match these bounds. The idea of these algorithms is to reduce, using the DH oracle, the DL problem in G to the same problem in certain groups defined over $GF(p)$ (for all prime factors p of $|G| = n$) such as $GF(p)^*$ [2] or elliptic curves over $GF(p)$ [4]. In the case where the orders of these groups are smooth, the reduction is efficient. The technique of [4] implies the following result (under an unproven number-theoretic conjecture on smooth numbers in intervals of type $[m, m + \Theta(\sqrt{m})]$).

Theorem 5 [4] *Let n be a positive integer such that all multiple prime factors of n are of order $(\log n)^{O(1)}$. Then there exists a generic algorithm that computes discrete logarithms in groups G of order n in time $(\log n)^{O(1)}$ and that makes calls to a DH oracle for G.*

Theorem 5 and Corollary 4 together imply (under the same number-theoretic conjecture) the following complete characterization of groups for which there exists an efficient generic reduction algorithm from the DL problem to the DH problem.

Corollary 6 *There exists a polynomial-time generic reduction of the DL problem to the DH problem for groups G of order n if and only if all multiple prime factors of n are of size $(\log n)^{O(1)}$.*

References

[1] D. Boneh and R. J. Lipton, Algorithms for black-box fields and their application to cryptography, *Advances in Cryptology - CRYPTO '96*, Lecture Notes in Computer Science, Vol. 1109, pp. 283–297, Springer-Verlag, 1996.

[2] B. den Boer, Diffie-Hellman is as strong as discrete log for certain primes, *Advances in Cryptology - CRYPTO '88*, Lecture Notes in Computer Science, Vol. 403, pp. 530–539, Springer-Verlag, 1989.

[3] W. Diffie and M. E. Hellman, New directions in cryptography, *IEEE Transactions on Information Theory*, Vol. 22, No. 6, pp. 644–654, 1976.

[4] U. M. Maurer, Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms, *Advances in Cryptology - CRYPTO '94*, Lecture Notes in Computer Science, Vol. 839, pp. 271–281, Springer-Verlag, 1994.

[5] U. M. Maurer and S. Wolf, On the complexity of breaking the Diffie-Hellman protocol, Tech. Rep. 244, Computer Science Department, ETH Zürich, April 1996. To appear in *SIAM Journal of Computing.*

[6] U. M. Maurer and S. Wolf, Diffie-Hellman oracles, *Advances in Cryptology - CRYPTO '96*, Lecture Notes in Computer Science, Vol. 1109, pp. 268–282, Springer-Verlag, 1996.

[7] S. C. Pohlig and M. E. Hellman, An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance, *IEEE Transactions on Information Theory*, Vol. 24, No. 1, pp. 106–110, 1978.

[8] J. T. Schwartz, Fast probabilistic algorithms for verification of polynomial identities, *Journal of the ACM*, Vol. 27, No. 4, pp. 701–717, 1980.

[9] V. Shoup, Lower bounds for discrete logarithms and related problems, *Advances in Cryptology - EUROCRYPT '97*, Lecture Notes in Computer Science, Vol. 1233, pp. 256–266, Springer-Verlag, 1997.