Hardness of Computing the Most Significant Bits of Secret Keys in Diffie-Hellman and Related Schemes

(Extended Abstract)

DAN BONEH[1] RAMARATHNAM VENKATESAN[2]
Princeton University Bellcore

[1] CS Department, Princeton University, Princeton NJ 08544.
e-mail: dabo@cs.princeton.edu
[2] Math and Cryptography Research Group,
Bellcore, 445 South Street, Morristown NJ 07960. *e-mail*: venkie@bellcore.com

Abstract. We show that computing the most significant bits of the secret key in a Diffie-Hellman key-exchange protocol from the public keys of the participants is as hard as computing the secret key itself. This is done by studying the following *hidden number problem*: Given an oracle $\mathcal{O}_\alpha(x)$ that on input x computes the k most significant bits of $\alpha \cdot g^x \bmod p$, find α modulo p. Our solution can be used to show the hardness of MSB's in other schemes such s ElGamal's public key system, Shamir's message passing scheme and Okamoto's conference key sharing scheme. Our results lead us to suggest a new variant of Diffie-Hellman key exchange (and other systems), for which we prove the most significant bit is hard to compute.

1 Introduction

The discrete logarithm problem (DLP) relative to a base g in $\mathbb{Z}_p^*$ is to find x given $y = g^x$. Assuming this problem to be hard, Diffie and Hellman [DH76] proposed a public key system. Here two participants, Alice and Bob, with private keys a and b respectively, compute g^a and g^b and send each other these values. Then they compute a secret key g^{ab}. Many believe that computing the function $\text{DH}_g(g^a, g^b) = g^{ab}$ is as hard as DLP.

After the secret key agreement, Alice and Bob can secure the session using encryption with a block cipher. A natural way to derive the key for the cipher would be to use a block of bits from g^{ab}. For example, if p is a 1024 bit prime, one may use the 64 most significant bits of g^{ab}. An attacker, who may not be able to compute the whole g^{ab}, may nevertheless succeed in computing this part of the bits of g^{ab} and crack the session. Hence it is important to know if the most significant bits (MSB) of g^{ab} are secure from an adversary who knows both g^a and g^b. Despite their long history, the security of the MSB's has not been shown for Diffie-Hellman keys.

N. Koblitz (Ed.): Advances in Cryptology - CRYPTO '96, LNCS 1109, pp. 129-142, 1996.

A number of cryptographic schemes proposed are related to or based on the Diffie-Hellman function $\mathrm{DH}_g(g^a, g^b) = g^{ab}$. These schemes depend on the "hidden" nature of g^{ab}. For examples, we refer to ElGamal's public key cryptosystem [EG85], Shamir's message passing scheme [Kob87, pp. 96–97], Bellare-Micali non-interactive oblivious transfer [MB89] and Okamoto conference key sharing scheme [Oka88].

In this paper, we study the security of the MSB's of the Diffie-Hellman key exchange scheme and the related schemes mentioned above. We show that the $\sqrt{\log p}$ (or more) MSB's of the Diffie-Hellman secret are as hard to compute as the entire secret. For a 1024 bit prime, our result implies that 32 or more MSB's are hard to compute. Our results are based on rounding in lattices using basis reduction algorithms[LLL82]. Asymptotically this can be improved to $\varepsilon\sqrt{\log p}$ for any fixed $\varepsilon > 0$ by combining the results of Babai[Bab86] and Schnorr [Schnorr, Schn94]

Furthermore, our results lead us to suggest a new variant of the Diffie-Hellman key exchange protocol for which the single most significant bit is as hard to compute as the entire secret. These schemes assume that the function $\mathrm{DH}_g(g^y, 2)$ is hard to compute from g, g^y. It is unknown if this is equivalent to the DH function, but it is equivalent to computing the "base change" function $g, g^y \mapsto 2^y$.

An expanded and updated version of the paper is available from the authors.

2 Discovering a number given an MSB oracle

Let p be a prime number and $n = \lceil \log p \rceil$ be its length in binary. We use $x \bmod p$ to denote the unique integer a in the range $[0, p-1]$ satisfying $x \equiv a \pmod{p}$. Given a prime p, we define $\mathrm{MSB}_k(x)$ as the integer t such that $(t-1) \cdot p/2^k \leq x < t \cdot p/2^k$. For example, $\mathrm{MSB}_1(x)$ is either 0 or 1 depending on whether x is smaller than or greater than $p/2$. For convenience we will sometimes assume that $\mathrm{MSB}_k(x)$ is an integer z satisfying $|x - z| < p/2^{k+1}$. To study the security of parts of the Diffie-Hellman secret we suggest studying the following abstract problem:

HIDDEN NUMBER PROBLEM (HNP): Fix p and k. Let $\mathcal{O}_{\alpha,g}(x)$ be an oracle that on input x computes the k most significant bits of $\alpha g^x \bmod p$:

$$\mathcal{O}_{\alpha,g}(x) = \mathrm{MSB}_k(\alpha \cdot g^x \bmod p)$$

The task is to compute the hidden number α modulo p, in expected polynomial (in $\log p$) time, given access to the oracle $\mathcal{O}_{\alpha,g}(x)$. One could consider the oracle $\mathcal{O}_{\alpha,\beta,g}(x) = \mathrm{MSB}_k(\alpha \cdot g^x + \beta \bmod p)$, but as we shall see, it is not hard to handle non-zero β. Unless otherwise stated, we take $\beta = 0$. Clearly, we wish to solve this problem with as small k as possible.

We assume that the oracle always gives the correct answer. Our methods can handle oracles that produce correct answers with probability (under uniform distribution over its inputs) at least $1 - \frac{1}{n}$. Note that the way the oracle $\mathcal{O}_{\alpha,g}$

is queried imposes a severe restriction. Namely, the multiplier for the hidden α is an element of $\mathbb{Z}_p^*$ for which the querying party knows the discrete logarithm to the base g. This restriction is quite crucial for the applications at hand. So, we work with the following oracle, using $t = g^x$ from $\mathbb{Z}_p^*$, where x is randomly chosen from $[0, p-1]$:

$$\mathcal{O}_\alpha(t) = \mathrm{MSB}_k(\alpha \cdot t \bmod p).$$

This gives rise to the *randomized* or *sampling* version of the HNP : given $\mathcal{O}_\alpha(t)$ where t are chosen uniformly and independently at random in $\mathbb{Z}_p^*$, find α. We study only this version, except in section 5 where we use non-random queries.

If one is allowed to query the oracle $\mathcal{O}_\alpha(t)$ at *chosen* values of t, then recovering α is much simpler. The study of this unrestricted problem is closely related to proving the bit security of the RSA cryptosystem as in [ACGS88], whose methods can be used to completely solve this problem. Namely, with chosen queries, the HNP can be solved when $k = 1$ even when the oracle is noisy; i.e. the oracle computes the MSB only on $1/2 + \varepsilon, \varepsilon > 0$ fraction of x correctly. But it is important for this method that the queries be correlated. A related topic of interest is that of predicting truncated pseudo-random number sequences. (See the remark after the Theorem 3).

This scenario is typical in learning theory, where some learning problems are easy if oracle queries can be arbitrarily chosen, and harder if only random queries are allowed. We here present some algorithms for solving the HNP using *random* queries.

3 Main Results

Solutions to the HNP can be used to prove that computing a part of the Diffie-Hellman secret is as hard as computing the entire secret. We state our first result regarding the HNP and show how it applies to the problems at hand. Below we show that the sampling version of the HNP can be solved using $k = \sqrt{\log p} + \log\log p$ bits, but this can be asymptotically improved to $\varepsilon\sqrt{\log p}$ for any fixed $\varepsilon > 0$ (See the paragraph on *improvements* below). The proofs of the results claimed in this section can be found in the next section.

Theorem 1. *Let α be some integer in the range $[1, p-1]$. Let $\mathcal{O}$ be a function defined by $\mathcal{O}(t) = \mathrm{MSB}_k(\alpha t \bmod p)$ with $k = \lceil\sqrt{n}\,\rceil + \lceil\log n\rceil$. There exists a deterministic polynomial time algorithm A such that*

$$\Pr_{t_1,\ldots,t_d}\left[A(t_1,\ldots,t_d,\mathcal{O}(t_1),\ldots,\mathcal{O}(t_d)) = \alpha\right] \geq \frac{1}{2}$$

where $d = 2\sqrt{n}$ and $t_1, \ldots, t_d$ are chosen uniformly and independently at random from $\mathbb{Z}_p^*$.

One may replace $1/2$ by $\frac{1}{2\sqrt{n}}$ and reduce the number of bits obtained from the oracle by $\log\log p$. Then, the expected run time goes up by a factor $\sqrt{n}$. One can alternately run $\sqrt{n}$ copies of the algorithm in parallel. Our first application of the theorem is regarding the hardness of MSB's of a Diffie-Hellman secret:

Theorem 2. *Set $k = \lceil\sqrt{n}\rceil + \lceil\log n\rceil$. Given an efficient algorithm to compute $\mathrm{MSB}_k(g^{ab})$ from inputs g^a, g^b, there is an algorithm to efficiently (in expected polynomial time) compute g^{ab} itself.*

This theorem remains true even if one can compute the MSB's on all but $1/n$ fraction of the random inputs g^a, g^b. This issue is addressed immediately after the proof of the theorem.

Improvements: With a proper choice of constants in the basis reduction algorithm, one can decrease k to $\lceil\sqrt{n}\rceil$ for large enough n. In the case of a 1024 bit prime, the first 32 (or more) bits of the secret key are hard to compute from the public keys. This is of practical interest as it shows that the 128 MSB's of the Diffie-Hellman secret may be securely used as the session key for the block cipher. A version of our implementation tested with smaller primes produced right answers (i.e. hidden numbers) even for $k = 1$-bit case. Note that as long as the primes are a few thousand bits long, the dimensions of the lattices involved are manageable, making our reductions practical.

Asymptotically one can take $k = \varepsilon\sqrt{\log p}$, for every fixed $\varepsilon > 0$. We achieve this result by combining the results of Babai's lattice rounding [Bab86] and Schnorr's semi reduced basis [Schnorr] to solve the following problem: given any vector, one can find a closest lattice vector having Euclidean norm within $2^{\epsilon d}$ factor of the optimum. This may be of interest on its own and a solution is given in [Schn94].

Reducing k to $\log\log p$ *bits :* Recently, results from [BV96] show that there exist polynomial number of advice bits depending only on p and g which enable one to solve the HNP in polynomial time using an oracle which returns only $\log\log p$ bits. This improves the $\sqrt{\log p}$ bound of Theorem 1 and can be applied to the Okamato conferencing scheme below (but not yet to other schemes in this paper). The proof relies on a new analysis of a lattice rounding technique with respect to a natural norm over matrices.

3.1 Related schemes

Theorem 1 is general enough to apply to many other DH-related cryptographic schemes, three of which we define next. For convenience, we assume that the generator g and the prime p have been agreed upon by Bob and Alice and all the required inverses exist.

ElGamal public key encryption: Bob picks a random x and publishes $y = g^x$ as his public key. To send a message m to Bob, Alice picks a random r and sends g^r, my^r. Bob can decode the message by computing $my^r/(g^r)^x$. To break the scheme one has to compute the function $EL_g(g^x, g^r, mg^{xr}) = m$.

Shamir message passing scheme: To send a message m Alice picks a random r, and sends $y = m^r$ to Bob. Bob picks a random s and sends $z = y^s$ back to Alice. Alice sends $w = z^{r^{-1}}$ to Bob who computes $m = w^{s^{-1}}$. Here r^{-1}, s^{-1} denote inverses modulo $p-1$. Breaking this scheme requires computing $SH(g^{rs}, g^r, g^s) = g$.

Okamato conference key sharing: Bob picks r at random and sends to Alice $x = g^r$. Alice picks a random s and sends $y = x^s$ back. Bob computes $y^{r^{-1}} = g^s$ which is the conference key they use. Since the conference key is determined by Alice's bits alone she can distribute the same key to all members of the conference. Cracking this scheme needs computing the function $OK_g(g^{rs}, g^r) = g^s$.

The equivalence of the above functions to Diffie-Hellman was studied in [SS95].

Theorem 3. *Set* $k = \lceil\sqrt{n}\rceil + \lceil\log n\rceil$. *Given an efficient algorithm to compute* $\text{MSB}_k(.)$ *for any one of the functions* $EL_g(g^x, g^r, mg^{xr}) = m$, $OK_g(g^{rs}, g^r) = g^s$ *or* $SH(g^{rs}, g^r, g^s) = g$, *there is an algorithm to efficiently (in expected polynomial time) compute the corresponding function in its entirety.*

Hence, the k MSB's of messages in the ElGamal public-key system and Shamir message passing, as well as the MSB's of secret keys in Okamato's scheme are as hard to compute as the whole from their corresponding public values. To apply Theorem 1 to these schemes one must derive several relations satisfied by the associated functions, which we do in Section 4.2.

When the generator g in the HNP is small, Theorem 1 can be improved. In Section 5 we show that for a generator g one can solve the HNP using an oracle returning only $k = \log g$ bits. For instance, when $g = 2$, an oracle returning the single most significant bit suffices. This result leads us to suggest a new variant of the Diffie-Hellman protocol for which computing the most significant bit is as hard as computing the entire secret. This variant is described in Section 5.1.

Remark: (Predicting Truncated Random Number Sequences) Here an iterative pseudo-random generator computes a sequence $x_i, i \geq 1$, and outputs $\text{MSB}_k(x_i)$. The task is to observe the outputs for a short period and break the generator by inferring the whole sequence. See [FHK*88] for discussion and references. One equation they study is $x_{i+1} = ax_i \bmod M$ and they show how to break the generator when k is constant times $\log M/(\log\log M)$ (Theorem 3.1 page 273 and its accompanying remark 3). This corresponds to restricting oracle queries to $a, a^2, a^3, \cdots$ whereas in our case the queries can be chosen randomly and independently, resulting in our bound $\sqrt{\log M}$.

A brief discussion on unpredictable bits and constructing pseudo-random generators is given in the Appendix.

4 Proofs

We now turn to the proof of Theorem 1. For notational convenience we assume that the prime p and the generator g of $\mathbb{Z}_p^*$ are fixed. Throughout this section we let $n = \log p$.

The proof relies on rounding techniques in lattices. We review briefly some relevant definitions and results. A (full rank) lattice L is defined to be the set of points

$$L = \left\{ y \ : \ y = \sum_{i=1}^{d} t_i b_i \ , \ t_i \in \mathbb{Z} \right\} ,$$

where the b_i are linearly independent vectors in $\mathbb{R}^d$. The set $\{b_i\}_{i=1}^{d}$ is called the basis of the lattice and d is the dimension of the lattice. We denote the L_2 norm of a vector $v \in \mathbb{R}^d$ by $\| v \|$.

Using the lattice basis reduction algorithm of Lenstra, Lenstra and Lovasz [LLL82], Babai [Bab86] shows how given a lattice L and a point v, one can find a lattice point which is approximately the closest to v.

Theorem 4. *[Bab86] Let L be a lattice of dimension d. Given a point $v \in \mathbb{R}^d$, there exists a polynomial time algorithm which finds a lattice point $w \in L$ such that*

$$\| v - w \| \leq 2^{\frac{d}{4}} \min\{\| v - b \| : \ b \in L\}.$$

Usually, the accounts of LLL algorithm use $2^{d/2}$ as the fudge factor rather than $2^{\frac{d}{4}}$ as stated above. By adjusting the constants in the definition of an LLL-reduced basis, one can improve this to $2^{d/4.6}$.

Proof of Theorem 1. We show a polynomial time algorithm for recovering the hidden number α. Recall that $d = 2\lceil\sqrt{n}\rceil, k = \lceil\sqrt{n}\rceil + \lceil\log n\rceil$ and we are given randomly chosen integers $t_1, \ldots, t_d$ and corresponding integers $a_1, \ldots, a_d$ such that for all $i = 1, \ldots, d$:

$$|(\alpha t_i \bmod p) - a_i| < p/2^k \qquad (1)$$

To find the hidden α, we construct the $d + 1$ dimensional lattice L spanned by the rows of the matrix

$$\begin{pmatrix} p & 0 & 0 & \ldots & 0 & 0 \\ 0 & p & 0 & \ldots & 0 & 0 \\ & \vdots & & & & \vdots \\ 0 & 0 & 0 & \ldots & p & 0 \\ t_1 & t_2 & t_3 & \ldots & t_d & 1/p \end{pmatrix}$$

We refer to the first d vectors in the basis as p-vectors. Notice by multiplying the bottom vector by α and subtracting the appropriate multiples of p-vectors we obtain a lattice vector

$$v_\alpha = (r_1, \ldots, r_d, \ \alpha/p)$$

where $|r_i - a_i| < p/2^k$ for all $i \leq d$. Define u to be the vector $u = (a_1, \ldots, a_d, 0)$. Then v_α is a lattice point whose distance from u is at most $\sqrt{d+1}\, p/2^k$. In other words,

$$\min\{\| u - w \| : \ w \in L\} \leq \sqrt{d+1}\, p/2^k$$

The following theorem shows that with high probability all lattice points which are this close to u have a special structure. Here the probability is over the random choice of $t_1, \ldots, t_d$.

Theorem 5 (Uniqueness Theorem). *Set $d = 2\lceil\sqrt{n}\rceil$ and $\mu = \frac{1}{2}\sqrt{n}+3$. Let α be a fixed integer in the range $[1, p-1]$. Choose integers $t_1, \ldots, t_d$ uniformly and independently at random in the range $[1, p-1]$. Let L be the lattice constructed as above and $u = (a_1, \ldots, a_d, 0)$ be a vector satisfying*

$$|(\alpha t_i \bmod p) - a_i| < p/2^{\mu}$$

Then with probability at least $\frac{1}{2}$ all $v \in L$ with $\| u - v \| < \frac{p}{2^{\mu}}$ are of the form:

$$v = (t_1\beta \bmod p, \ldots, t_d\beta \bmod p,\ \beta/p) \textit{ where } \alpha \boxminus \beta \pmod{p}$$

Proof. Let β, γ be two integers. Define the *modular distance* between β and γ as

$$\mathrm{dist}_p(\beta, \gamma) = \min_{b \in \mathbb{Z}} |\beta - \gamma - bp|$$

For example, $\mathrm{dist}_p(1, p) = 1$. Suppose $\beta \not\equiv \gamma \pmod{p}$ and they are both integers in the range $[1, p-1]$. Define

$$A = \Pr_t\left[\mathrm{dist}_p(\beta t, \gamma t) > 2p/2^{\mu}\right]$$

where t is an integer chosen uniformly at random in $[1, p-1]$. Then

$$A = \Pr_t\left[\frac{2p}{2^{\mu}} < (\beta - \gamma)t \bmod p < p - \frac{2p}{2^{\mu}}\right] = \frac{\lfloor p - \frac{2p}{2^{\mu}} \rfloor - \lceil \frac{2p}{2^{\mu}} \rceil}{p-1} \geq 1 - \frac{5}{2^{\mu}}$$

This follows since for every $x \in [\frac{2p}{2^{\mu}}, p - \frac{2p}{2^{\mu}}]$ there exists a t such that $(\beta - \gamma)t = x \pmod{p}$. In general, a lattice point v has the form

$$v = (\beta t_1 - b_1 p,\ \beta t_2 - b_2 p, \ldots, \beta t_d - b_d p,\ \beta/p)$$

for some integers $\beta, b_1, \ldots, b_d$. Suppose $\| v - u \| < p/2^{\mu}$. We show that with probability at least $\frac{1}{2}$ the vector v satisfies $\beta \equiv \alpha \pmod{p}$ and $\beta t_i - b_i p \in [0, p]$ for all i. Observe that if $\beta = \alpha \pmod{p}$, then $\beta t_i - b_i p \in [0, p]$ for all i. Otherwise at least one of the components of $v - u$ is bigger in absolute value than $p/2^{\mu}$.

Now, suppose $\beta \not\equiv \alpha \pmod{p}$. Then

$$\Pr\left[\| v - u \| > p/2^{\mu}\right] \geq \Pr\left[\exists i \ :\ \mathrm{dist}_p(t_i\beta, a_i) > p/2^{\mu}\right] \geq$$
$$\Pr\left[\exists i \ :\ \mathrm{dist}_p(t_i\beta, t_i\alpha) > 2p/2^{\mu}\right] = 1 - (1 - A)^d \geq 1 - \left(\frac{5}{2^{\mu}}\right)^d$$

Since $\beta \not\equiv \alpha \pmod{p}$ there are exactly $p - 1$ values of $\beta \bmod p$ to consider. Hence, the probability there exists a lattice point contradicting the statement of the theorem is at most

$$(p-1) \cdot \left(\frac{5}{2^{\mu}}\right)^d < \frac{1}{2}$$

The last inequality follows from the fact that $d(\mu - \log_2 5) > \log p + 1$. This completes the proof of the theorem. ■

By applying Theorem 4 to the lattice L and the vector u, we now can find in polynomial time a lattice vector $w \in L$ such that

$$\| u - w \| \leq 2^{\frac{d}{4}} \min\{\| u - b \| : b \in L\} \leq 2^{\frac{d}{4}} \cdot \sqrt{d+1}\, p/2^k < p/2^{\mu},$$

where $\mu = \frac{1}{2}\sqrt{n} + 3$ as in the uniqueness theorem. The last inequality follows since $\mu < k - \frac{d}{4} - \frac{1}{2}\log(d+1)$ for large n. Since $\mu < k$ Theorem 5 shows that with probability $\geq 1/2$ we have $w = v_\alpha = (r_1, \ldots, r_d, \alpha/p)$, from which one can recover the hidden element α easily. This completes the proof of Theorem 1. ■

The technique above can be used to solve a more general problem. Let α, β be two integers in the range $[1, p-1]$. Let $\mathcal{O}(t)$ be the function defined by

$$\mathcal{O}(t) = \mathrm{MSB}_k(\alpha t + \beta \bmod p)$$

where $k = \sqrt{n}$. Then one can recover α, β in expected polynomial time from random samples of the function $\mathcal{O}(t)$. Given random integers $t_1, \ldots, t_d$ in the range $[0, p-1]$ construct a $d+2$ dimensional lattice L spanned by the rows of the matrix

$$\begin{pmatrix} p & 0 & 0 & \ldots & 0 & 0 & 0 \\ 0 & p & 0 & \ldots & 0 & 0 & 0 \\ \vdots & & & & \vdots & & 0 \\ 0 & 0 & 0 & \ldots & p & 0 & 0 \\ t_1 & t_2 & t_3 & \ldots & t_d & 1/p & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1/p \end{pmatrix}.$$

Let b_{d+1} be the second to last vector in the lattice and b_{d+2} be the last vector. The vector $\alpha \cdot b_{d+1} + \beta \cdot b_{d+2}$ minus an appropriate number of p-vectors will be very close to the vector constructed from the oracle's answers. With larger values of k and μ, the uniqueness theorem (Theorem 5) can be generalized and then Theorem 4 can be applied to recover α, β.

4.1 Proof of hardness of Diffie-Hellman MSB's

Proof of Theorem 2. Let A be an efficient algorithm computing $\mathrm{MSB}_k(g^{ab})$ given g^a, g^b. We show that given g^a, g^b one can use the algorithm A to compute g^{ab} and thus break the Diffie-Hellman protocol. Set $\alpha = g^{ab} \bmod p$ and $h = g^b$. Let $\mathcal{O}(x)$ be the function $\mathcal{O}(x) = \mathrm{MSB}_k(\alpha \cdot h^x \bmod p)$. Then

$$\mathcal{O}(x) = \mathrm{MSB}_k(\alpha \cdot h^x) = \mathrm{MSB}_k(g^{ab} \cdot (g^b)^x) = \mathrm{MSB}_k(\mathrm{DH}_g(g^{a+x}, g^b)) = A(g^{a+x}, g^b)$$

Hence, the algorithm A can be used to evaluate the function $\mathcal{O}(x)$ for arbitrary x. We obtain a hidden number problem which can be solved in expected polynomial time according to Theorem 1. This shows that $\alpha = g^{ab}$ can be found if the algorithm A exists. ■

We now sketch how to handle the case of A that has low error rate. More precisely, we assume that A has success probability of $> 1 - 1/n$ when its inputs are random. In this case A answers all $O(\sqrt{n})$ queries correctly with a high probability.

Let x, y be random and $\gamma = ay + bx + xy$. We need not compute γ but we can compute g^γ and $\mathcal{O}(\gamma)$ (short hand for a multivariate oracle with respect to generators $g, g'g^b$ and inputs x, y, xy):

$$\begin{aligned}\mathcal{O}(\gamma) &= \text{MSB}_k(\alpha \cdot g^\gamma) = \\ &= \text{MSB}_k(g^{ab} \cdot (g^{ay+bx+xy})) = \text{MSB}_k(\text{DH}_g(g^{a+x}, g^{b+y})) = A(g^{a+x}, g^{b+y})\end{aligned}$$

By using the algorithm A as the oracle, and solving a HNP (Theorem 1), we get the required result.

4.2 Proofs for DH-Related Schemes

We first derive some relations for the functions $EL_g(a,b,c), SH(a,b,c)$ and $OK_g(a,b)$ defined in Section 3. We note that our queries are restricted to random points in the reductions. For simplicity, we first consider relations useful for randomizing one of the inputs to the algorithms used to simulate the hidden number oracles.

Lemma 6. *The functions $EL_g(a,b,c), SH(a,b,c)$ and $OK_g(a,b)$ satisfy the following relations:*

$$\begin{aligned} EL_g(g^{x+r}, g^y, mg^{xy}) &= m \cdot (g^{-y})^r \\ SH(g^{xy}, g^{x+rxy}, g^y) &= g \cdot (g^y)^r \qquad (\textit{when } \gcd(yr+1, p-1) = 1\,) \\ OK_g(g^{y(x+r)}, g^y) &= g^x \cdot g^r \end{aligned}$$

Proof. These relations are derived as follows:

$$EL_g(g^{x+r}, g^y, mg^{xy}) = El_g(g^{x+r}, g^y, (mg^{-yr}) \cdot g^{y(x+r)}) = m \cdot (g^{-y})^r$$

To prove the second relation define $h = g^{yr+1}$. Then

$$SH(g^{xy}, g^{x+rxy}, g^y) = SH(h^{\frac{xy}{yr+1}}, h^{\frac{x+rxy}{yr+1}}, h^{\frac{y}{yr+1}}) = h = g \cdot (g^y)^r$$

The middle equality follows since $\frac{x+rxy}{yr+1} \cdot \frac{y}{yr+1} = \frac{xy}{yr+1}$ and hence h must be the value of the Shamir function on those three inputs. The third relation follows since

$$OK_g(g^{y(x+r)}, g^y) = g^{x+r} = g^x \cdot g^r$$

■

Proof of Theorem 3. Let us consider the Shamir function. The case of the other two functions is similar. Let A be an efficient algorithm computing $\text{MSB}_k(m)$ given m^x, m^y, m^{xy}. Set $g = m^y \bmod p$ and define $\mathcal{O}(r)$ to be the function $\mathcal{O}(r) = \text{MSB}_k(m \cdot g^r \bmod p)$. Then,

$$\begin{aligned}\mathcal{O}(r) &= \text{MSB}_k(m \cdot (m^y)^r) = \\ &= \text{MSB}_k\big(SH(m^{xy}, m^{x+rxy}, m^y)\big) = A(m^{xy}, m^{x+rxy}, m^y)\end{aligned}$$

The second equality follows by Lemma 6. Notice that m^{x+rxy} can be computed as $m^x \cdot (m^{xy})^r$. It follows that given m^x, m^y, m^{xy} the algorithm A can be used to simulate the oracle $\mathcal{O}(r)$. Hence, we obtain a HNP which can be solved in expected polynomial time according to Theorem 1. This shows that the secret message m can be found if the algorithm A exists. ∎

The same argument which appears after the proof of Theorem 2 can be used to show that Theorem 3 holds even when the oracle is allowed to make mistakes on a $\frac{1}{n}$ fraction of the inputs. This is done by randomizing the inputs to the oracle A. For the Shamir scheme one can randomize the inputs to the oracle by using the relation $SH(g^{xyt^2}, g^{tx(1+ry)}, g^{ty(1+sx)}) = g^{(1+yr)(1+xs)}$. We pick r, s, t at random, set $\gamma = yr + xs + xyrs$ and use

$$\begin{aligned}\mathcal{O}(\gamma) &= \text{MSB}_k(m \cdot m^\gamma) = \\ &= \text{MSB}_k\big(SH(m^{xyt^2}, m^{tx(1+ry)}, m^{ty(1+sx)})\big) = A(m^{xyt^2}, m^{tx(1+ry)}, m^{ty(1+sx)})\end{aligned}$$

The value m^γ can be easily computed given m^x, m^y, m^{xy}.

5 The case of a small generator

The original formulation of the HNP asks one to discover a hidden number α from an oracle which outputs the k most significant bits of $\alpha \cdot g^r \bmod p$. In the previous section we proved that setting $k = O(\sqrt{\log p})$ suffices. In this section we show that when the generator g is small this result can be improved. For instance, when $g = 2$ only the most significant bit is needed; i.e., $k = 1$ suffices. This result leads us to suggest a new variant of the Diffie-Hellman protocol.

For a generator g of $\mathbb{Z}_p^*$ we define the significant bit function $SB_g(x \bmod p)$ to be an integer t such that $(t-1)p/g \leq x < tp/g$. Clearly $t \in [0, g-1]$ and therefore the function SB_g returns at most $\log_2 g$ bits of information. Notice that when $g = 2$ the function $SB_2(x)$ is the same as the $\text{MSB}_1(x)$ function used in Section 2.

Theorem 7. *Let α be some integer in the range $[1, p-1]$. Let $\mathcal{O}$ be a function defined by $\mathcal{O}(x) = SB_g(\alpha g^x \bmod p)$ for some generator g of $\mathbb{Z}_p^*$. Then there exists an algorithm which, given access to an oracle computing the function $\mathcal{O}$, can find α in polynomial time in $\log p$.*

Proof. Let U and L be upper and lower bounds on α, i.e. $L \leq \alpha < U$. Initially we set $L = 0$ and $U = p$. The algorithm will iteratively decrease the gap between U and L until $U - L < 1$ in which case α is found. Throughout the algorithm we maintain that at the r'th iteration $L = (t-1) \cdot \frac{p}{g^r}$ and $U = t \cdot \frac{p}{g^r}$ for some integer $t \in [1, g^r]$. Initially $r = 0$.

Consider the r'th iteration. Then $U = t \cdot \frac{p}{g^r}$ and $L = (t-1) \cdot \frac{p}{g^r}$ for some integer t. Since $L \leq \alpha < U$ we have

$$0 \leq \alpha g^r - pt < p$$

The algorithm will now query the oracle at the point $x = r$. By definition, the oracle returns a number z such that

$$(z-1)\frac{p}{g} \leq \alpha g^r \bmod p < z\frac{p}{g}$$

Since $\alpha g^r \bmod p = \alpha g^r - pt$ we can rewrite the above inequality as:

$$\frac{p(z-1) + ptg}{g^{r+1}} < \alpha < \frac{pz + ptg}{g^{r+1}}$$

We now take these lower and upper bounds to be the L and U used in the next iteration. Observe that $U - L = p/g^{r+1}$. This shows that the gap between U and L decreased as expected completing the proof of the theorem. ■

Theorem 7 shows that the HNP can be solved using $k = \log g$ most significant bits when the generator g is used. For small values of g this improves on the result of the previous section. Unlike the algorithm described in the previous section, this algorithm relies on the ability to query the oracle at chosen inputs.

5.1 A variant of Diffie-Hellman and its bit security

Theorem 7 suggests a new variant of the Diffie-Hellman protocol. This new variant is motivated by the following corollary which states that the most significant bit of $\mathrm{DH}_g(g^x, 2)$ is as hard to compute as all of $\mathrm{DH}_g(g^x, 2)$. For clarity we observe that

$$\mathrm{DH}_g(g^x, 2) = \mathrm{DH}_g(g^x, g^{\log_g 2}) = g^{x \log_g 2} = 2^x \pmod p$$

Corollary 8. *Given an efficient algorithm A to compute $\mathrm{MSB}_1(\mathrm{DH}_g(g^x, 2))$ from g, g^x there is an algorithm to efficiently (in polynomial time) compute $\mathrm{DH}_g(g^x, 2)$ itself.*

Proof. Set $\alpha = \mathrm{DH}_g(g^x, 2) = 2^x$. We have already seen (Theorem 2) that the algorithm A can be used to define a hidden number oracle

$$\mathcal{O}(r) = \mathrm{MSB}_1(\mathrm{DH}_g(g^{x+r}, 2)) = \mathrm{MSB}_1(\alpha \cdot 2^r \bmod p)$$

Hence, by Theorem 7, α can be found in polynomial time. ■

The corollary suggests a Diffie-Hellman protocol where Alice always sends the value 2 to Bob. The idea is for Alice to pick a random generator g for which she knows a value x satisfying $g^x = 2 \bmod p$. More precisely suppose Alice and Bob have already agreed on a prime p. To perform secret key exchange they use the following protocol which we call the MODIFIED DIFFIE-HELLMAN PROTOCOL:

1. Alice picks a random number $x \in [1, p-1]$ with $\gcd(x, p-1) = 1$. She computes $g = 2^x \pmod p$ and sends g to Bob.
2. Bob picks a random number y in the range $[1, p-1]$ and sends g^y to Alice.

The secret they both agree on is $\alpha = \text{DH}_g(g^y, 2) = 2^y \pmod p$. Clearly Bob can compute this value. Alice can compute this value since

$$2^y = g^{yx^{-1}} \pmod p$$

where x^{-1} is the inverse of x modulo $p-1$.

An adversary who wishes to discover the secret shared by Alice and Bob observes g, g^y and must compute the value $\text{DH}_g(g^y, 2)$. Corollary 8 shows that computing the MSB of the secret shared by Alice and Bob is as hard as computing the entire secret. The next corollary summarizes this.

Corollary 9. *Given an oracle that computes the single most significant bit of the modified Diffie-Hellman secret, there exists a polynomial time algorithm that computes the entire secret.*

In the same spirit we can design a new variant of the ElGamal public key scheme in which the single most significant bit of the message is as hard to compute as the whole message. Let p be a prime. Alice randomly chooses an integer x with $\gcd(x, p-1) = 1$. Her public key is $g = 2^x \pmod p$. Her private key is $y = x^{-1} \pmod{p-1}$. To send a message m Bob picks a random r and sends $m \cdot 2^r, g^r$ to Alice. Alice can compute 2^r since $2^r = (g^r)^y \pmod p$ and recover the message m. To break this scheme an adversary must compute $El'(g, g^r, m \cdot 2^r) = m$. The same arguments as above show that given an oracle computing the single most significant bit of m is equivalent to computing all of m. We caution here that when used for ElGamal signatures, smooth generators are bad [B96].

A possible drawback of these new variants is that they rely on the hardness of $DH_g(g^y, 2)$ for their security which needs further study. This is a special case of the Diffie-Hellman function which could potentially be easier to break. The standard heuristic way of arguing about the security of the Diffie-Hellman protocol is to argue that the corresponding discrete log problem is hard. In our case the corresponding discrete log problem is that of computing discrete log of 2 base g. One can easily show that computing discrete log of 2 base g is as hard as computing discrete log of any x base g (observe that $\log_g x = \log_g 2 / \log_x 2$). Thus, one can argue that the standard heuristic discrete log argument supports the security of this variant.

An interesting point is that breaking modified the Diffie-Hellman scheme modulo a composite is as hard as factoring integers. We leave the details for the final version of the paper. The modified scheme modulo a composite still retains the property that the most significant bit of the secret is as hard to compute as the entire secret.

6 Concluding Remarks

We proved that $\sqrt{\log p}$ MSB's of the Diffie-Hellman secret keys, or the messages in the ElGamal public key encryption system or Shamir's three pass protocol are as hard to compute as the entire message. The same result holds for the secret key of the Okamoto conference scheme, for which our recent work [BV96] improves the result to $\log logp$ bits in the non-uniform model.

It would be interesting to improve these results to a single bit case, and also to the case of a noisy oracle that answers only $\frac{1}{2} + \varepsilon$ fraction of its queries correctly. The security of the modified DH-scheme and its underlying function $g, g^y \mapsto 2^y(\text{mod} p)$ definitely warrants a study. The question analogous to the standard DH and DLOG equivalence (mod p) in this case is interesting as well.

Acknowledgments

We are grateful to Don Coppersmith and an anonymous Crypto96 reviewer for comments on small generators in the modified DH protocol. We thank A.K. Lenstra and C.P. Schnorr for discussions.

References

[ACGS88] W. Alexi, B. Chor, O. Goldreich, and C. Schnorr. RSA and Rabin functions: Certain parts are as hard as the whole. *SIAM J. Computing*, 17(2):194–209, Nov. 1988.

[Bab86] L. Babai. On Lovasz' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6:1–13, 1986.

[B96] D. Bleichenbacher. Generating ElGamal Signatures without knowing the secret key *EuroCrypt96*, pp 10-18, 1996

[BV96] D. Boneh, R. Venkatesan. Basis reduction with a matrix norm and its applications. *Manuscript*, 1996.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[EG85] T. El-Gamal. A public key cryptosystem and a signature scheme based on the discrete logarithm. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[FHK*88] A. Frieze, J. Hastad, R. Kannan, J. Lagarias, and A. Shamir. Reconstructing Truncated Integer Variables Satisfying Linear Congruences. *SIAM J. Computing*, 17(2):262–280, 1988.

[GL] O. Goldreich, L.A. Levin. Hard core bits based on any one way function. In *Proc. ACM Symp. on Theory of Computing* 1989.

[Kob87] N. Koblitz. *A course in number theory and cryptography*. Springer-Verlag, 1987.

[LLL82] A. Lenstra, H. Lenstra, and L. Lovasz. Factoring polynomial with rational coefficients. *Mathematiche Annalen*, 261:515–534, 1982.

[MB89] S. Micali M. Bellare. Non-interactive oblivious transfer and applications. In *Proc. CRYPTO 89*, pages 547–557, 1989.

[Oka88] T. Okamoto. *Encryption and Authentication Schemes Based on Public Key Systems.* PhD thesis, Univ. of Tokyo, 1988.

[Schnorr] C, Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms *Theoretical Computer Science*, 53:201–224,1987

[Schn94] C.P. Schnorr: Reduced Lattice Bases and Successive Minima. *Combinatorics, Probability and Computing* 3, (1994), 507-522.

[SS95] K. Sakurai and H Shizuya. Relationships among the computational powers of breaking discrete log cryptosystems. In *Proc. EUROCRYPT 95*, pages 341–355, May 1995.

Appendix

Hard core bits and Pseudo-Random Generators: While one hopes to show that the MSB's are pseudo-random, one may obtain cryptographically unpredictable bits from DH-assumption. We very briefly outline this here and we rely heavily on the proofs of the Goldreich-Levin theorem.

Let $F(g^{ab}, g^a, g^b) = g^a, g^b$. If the triple is not of the given form, we may assume that F outputs $00\ldots0$. Inverting F is "hard". That is, computing $F^{-1}(g^a, g^b)$ is at least as hard as computing $\mathrm{DH}_g(g^a, g^b)$. But F is not a one-way function in the usual sense, since computing g^{ab} from g^a, g^b is hard as well. Then put $x = Ay$ where A is a $k \times |y|$ random Toeplitz Matrix and $y = g^{ab}$. We claim that the distribution of x, g^a, g^b, A can not be distinguished with significant advantage from the distribution when x is replaced by a purely random string of equal length. Otherwise, using the theorems in [GL] there exists an algorithm that outputs a short list of candidates for g^{ab}, g^a, g^b that contains the right one. So one can randomly choose one from the list. Alternately, if additional information such as plain text cipher text pairs using the hash value as a key for a block cipher, we can pick the right value from the list.

The results in [GL] allow us up to $k = \epsilon \log s(n)$, where $s(n)$ is a lower bound on the time for inverting F on all but a negligible fraction of non-zero instances. Since the security of DLP is sub exponential, k will be at most sub linear. However it is conceivable that linear number of bits (without hashing) of g^{ab} are hard to predict. Current rounds of standards advocate hashing g^{ab} with a collision resistant hash function (e.g. MD6,SHA). The construction above uses a simpler and well understood hash function.