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

Dan Boneh
dabo@cs.princeton.edu
Department of Computer Science
Princeton University
Princeton, NJ 08544

Ramarathnam Venkatesan
venkie@bellcore.com
Bellcore Room-1C341B,
445 South Street,
Morristown NJ 07960

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 $O_{\alpha,\beta}(x)$ that on input x computes the k most significant bits of $\alpha g^x + \beta \bmod p$, find $\alpha, \beta \bmod p$. We present many other applications of this problem including: (1) MSB's in El-Gamal encryptions, Shamir Message passing scheme etc. are hard to compute. (2) Factoring with hints. Our results lead us to suggest a new variant of Diffie-Hellman key exchange, for which we prove the most significant bit is hard to compute.

Keywords: Key-Exchange, Diffie-Hellman, ElGamal, Rounding in Lattices, Hard Bits, Most Significant Bits.

Technincal report TR–515–96
Computer Science Department,
Princeton University
Princeton, NJ, 08544
March 1996

1 Introduction

1.1 Motivation

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}. It is believed that computing the function $\mathrm{DH}_g(g^a, g^b) = g^{ab}$ is hard as DLP.

After the secret key agreement, Alice and Bob want to use a block-cipher for securing the session with encryption. For practicality and speed they may wish to use 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 the length of the prime p has a length of 1024 bits, 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 the 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.

A number of cryptographic schemes that have been proposed in the literature and used 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 mention ElGamal's public key cryptosystem [ElG85], Shamir's Message Passing scheme [Kob], Bellare-Micali Non-interactive Oblivious Transfer [BM89] and Okamoto Conference Key sharing scheme [O88].

In this paper, we study the security of the MSB's of Diffie-Hellman key exchange schemes and the related schemes mentioned above. Our results lead us to suggest a new variant of the Diffie-Hellman key exchange protocol.

1.2 Preliminaries and Statement of Results

Notation Throughout the paper we use the notation $x \bmod p$ to denote the unique integer a in the range $[0, p-1]$ satisfying $x \equiv a \pmod{p}$.

Most Significant bits We will often refer to the function $\mathrm{MSB}_k(x)$ where $0 \leq x < p$. Given a prime p the function $\mathrm{MSB}_k(x)$ is defined to be 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 $|x - z| < p/2^{k+1}$. The index of the (small) subinterval of [1,p-1] in which a number lies, is often used as the hard bits, e.g. ([BM],[LW]).

This notion of MSB_k is very similar to the standard definition of the most significant bit. These notions differ by at most 1 bit. That is, all our results will also work using the standard definition of the most significant bit. Given the k most significant bits of x we construct the number z as above by appending the appropriate number of zeros to the given bits.

To address the questions at hand, we suggest studying the following problem:

Hidden Number Problem: Fix p and k. Let be $O_{\alpha,\beta,g}(x)$ be an oracle that on input x computes the k most significant bits of $\alpha g^x + \beta \bmod p$. The task is to compute the hidden numbers α and β modulo p, given access to the oracle $O_{\alpha,\beta,g}(x)$.

One could query the oracle at chosen values for x and we assume that the oracle always gives the correct answer. (We briefly discuss the case of oracles that sometimes give wrong answers later.) Note

that the way the oracle is queried imposes a 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. Being able to deal with this restriction is quite crucial for the applications at hand. This makes the attack hard and differs from the methods used when this restriction is relaxed, which we describe next.

Oracles with Unrestricted Queries The case when one can query at arbitrary multipliers for α is well studied. Namely assume that the oracle computes $O_{\alpha,\beta}(x) = \mathrm{MSB}(\alpha x + \beta) \bmod p$. For the case of $\beta = 0$, Alexi, Chor, Goldreich and Schnorr [ACGS] completely solve this problem. Their method works, even when the oracle is noisy: namely, it computes the MSB only on $1/2 + \epsilon, \epsilon > 0$ fraction of x correctly. However it is essential to be able to query the oracle at arbitrary points. Also, in the unrestricted query case, it is easy to see that it corresponds to the case of a linear congruential generator $x_{i+i} = ax_i + b \bmod p$, which truncates x_i and outputs their MSB's. The problem of reconstructing integer variables satisfying linear congruences was studied in [FHKLS 88]. Our results in some cases also improve on the results of [FHKLS 88] by requiring only $O(\sqrt{\log p})$ MSB's of the sequence, improving on their $p/\log\log p$ bits bound.

1.3 Main Results

We first prove the following two results regarding the hardness of MSB's of Diffie-Hellman keys.

THEOREM 1.1 *Given an efficient algorithm to compute $3\sqrt{\log p}$ bits of g^{ab} given g^a, g^b, then there is an algorithm to efficiently (in expected polynomial time) compute g^{ab} itself.*

Hence the MSB's of $DH_g(g^a, g^b)$ is secure, if computing DH is hard. For example, in the case of a 1024 bit prime, the first 100 bits are hard to compute. We give an algorithm to solve this problem via *rounding* in lattices [B]. Our reduction algorithm is efficient and its implementations always finds the hidden values quite quickly. We remark that the lattice dimensions are small for usual sizes of p. Even when the oracle gives exactly one bit, the implementation successfully found the solution in all runs.

THEOREM 1.2 *(Small Generator case) If $g \leq 2^k$ computing k most significant bits of $DH_g(g^a, g^b)$ is hard.*

For example, if $g = 2$ is a generator for $\mathbb{Z}_p^*$ even the first bit is hard to compute.

These results, extend to the following related schemes. We list the schemes and associated functions. Below, we assume that the generator and the prime 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 $y = g^x$. To send a message m to Bob, Alice picks a random r and sends g^r, my^r. 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 random r, and sends $y = m^r$. Bob picks a random s and sends $z = y^s$ back to Alice. Alice sends $w = z^{r^{-1}}$ to Bob who computes $m = w^{s^{-1}}$. To break the schemes one computes $SH(g^{ab}, g^a, g^b) = g$.

Okamato Conference Key Sharing . Bob picks r at random and sends to Alice $x = g^r$. Alice picks a random s sends $y = x^s$ back. Bob computes $y^{r^{-1}}$. The associated function is $OK_g(g^{ab}, g^a) = g^b$.

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

THEOREM 1.3 *The most significant $\sqrt{\log p}$ bits of the above protocols are hard to compute, unless the Diffie-Hellman function is easy to compute.*

We remark here that these applications do not follow directly from the above. It suggests new variants on the cryptoschemes as well: for example, in the ElGamal scheme to encrypt a message m one may use the simpler operation of addition as in $m + DH_g(g^a, g^b)$ instead of multiplication if we only need to hide first k-bits of m.

A new variant of key-exchange protocol: We suggest a new variant of Diffie-Hellman key exchange, where computing the MSB of the secret key is hard.

We next discuss applications to factoring.

THEOREM 1.4 *(Factoring given hints) Let $n = pq$ and $k \geq \sqrt{\log n}$. Given random $x_1, \ldots, x_k$, where $x_i \bmod p < p/2^{3\sqrt{\log n}}$, one can factor n in polynomial time.*

This follows readily from the hidden number problem's solution. Given x, if one can generate random x satisfying $x \bmod p < p/2^{\sqrt{\log n}}$ in subexponential time, then one can factor in subexponential time via lattice techniques. Coppersmith solves this problem with $k = 1$ and the bound of $n^{1/6}$. Our bound is better, but we need more samples satisfying the bound. The implication to the Vanstone et. al cryptosystem is discussed in [Coppersmith 96].

Hardcore bits Pseudo-Random Generators Our results do not handle the case when the adversary can predict the most significant bit of g^{ab}. Nevertheless, we believe our methods are useful in handling that case.

THEOREM 1.5 *If $DH_g(g^a, g^b) = g^{ab}$ is hard, there is a pseudo-random generator. Moreover, after the key exchange, Alice and Bob need not interact to iterate the function.*

Many researchers have noted versions of this theorem. It is important to note that the Diffie-Hellman function is not a one-way function, since it not possible to check the output. We show that it is possible to extract cryptographically secure psuedo-random bits from $g^{a,b}$ using Goldreich-Levin theorem on hard-core bits. For iteration purposes we use the function $g^x \mapsto g^{x^2}$ iteratively. This function is as hard as the Diffie-Hellman function.

We remark here that to elude attacks like those in [B 95] resulting from lost session keys in a multi-party scenario, it is suggested for next round of standards that the secret key be hashed with a collision-resistant hash function (e.g. MD5, SHA) to break the algebraic correlations. Extracting hard-core bits uses much weaker hash functions (like inner product mod 2, or $ax + b \bmod p$), and the hash value this can be used as a key for a (pseudo) random function to derive keys that are provably uncorrelated session keys for different sessions.

2 Proofs of the main results

In this section we describe two algorithms for the hidden number problem. For notational convenience we assume that the prime p and the generator g of $\mathbb{Z}_p^*$ are fixed. Throughout this section we let $n = \log p$.

2.1 Using Square Root log$p$ Bits

The main theorem of this section shows that the hidden number problem can solved using an oracle which output $\sqrt{\log p}$ bits.

THEOREM 2.1 *Let α be some integer in the range $[1, p-1]$. Let O be a function defined by $O(x) = \mathrm{MSB}_k(\alpha g^x \bmod p)$ where $k = 3\lfloor\sqrt{\log_2 p}\rfloor$. Then there exists an algorithm which given access to an oracle computing the function O can find α in polynomial (in $\log p$) time.*

The proof relies on rounding techniques in lattices. We therefore first briefly recall the notion of a lattice and cite some relevant 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 $\| v \|$.

An important result due to Babai [4] shows how given a lattice L and a point v one can find a lattice point which is approximately the closest to v. Using the lattice basis reduction algorithm of Lenstra, Lenstra and Lovasz [13] he proves the following.

THEOREM 2.2 *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*

$$\| v - w \| \leq 2^{\frac{d}{2}-1} \min\{\| v - b \| : \ b \in L\}$$

Proof of Theorem 2.1 We show a probabilistic expected polynomial time algorithm for recovering the hidden number α. Let $d = \sqrt{\log p}$. The algorithm first picks integers $x_1, \ldots, x_d$ in the range $[1, p-1]$ independently and uniformly at random. It then computes $t_i = g^{r_i} \bmod p$. Next it queries the oracle at the points $r_1, \ldots, r_d, 0$. The oracle returns values $a_1, \ldots, a_d, a_{d+1}$ such that for all $i = 1, \ldots, d$

$$|(\alpha t_i \bmod p) - a_i| < p/2^k \qquad \text{and} \qquad |(\alpha \bmod p) - a_{d+1}| < p/2^k \tag{1}$$

To find the hidden α we construct the following lattice (the rows represent the basis vectors):

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

The dimension of this lattice is $d+1$. We refer to the first d vectors in the basis as p-vectors. Notice that when we multiply the bottom vector by α and subtract the appropriate number of p-vectors we obtain a lattice vector

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

where $|r_i - a_i| < p/2^k$ for all $i \leq d$. In other words, the vector $u = (a_1, \ldots, a_d, a_{d+1})$ satisfies:

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

THEOREM 2.3 *(Uniqueness Theorem) Let α be a fixed integer in the range $[1, p-1]$. Recall that $d = \lfloor \sqrt{\log_2 p} \rfloor$ and $k = 3\lfloor \sqrt{\log_2 p} \rfloor$. 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 $u = (a_1, \ldots, a_{d+1})$ be a vector satisfying condition 1. Then with probability $\frac{1}{2}$, there is a unique lattice vector $v \in L$ satisfying $\| u - v \| < \frac{p}{2^{k-d}}$.*

Proof It should be clear that the lattice point v_α constructed above satisfies $\| u - v_\alpha \| < p/2^{k-d}$. Proving that v_α is the only such lattice point requires more work. Let β, γ be two integers. We 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 $\beta \neq \gamma \pmod p$ and they are both integers in the range $[1, p-1]$. Define

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

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

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

This follows since for every $x \in [\frac{2p}{2^{k-d}}, p - \frac{2p}{2^{k-d}}]$ there exists a t such that $(\beta-\gamma)t = x \pmod p$. We now prove that

$$\Pr\left[\exists v \in L \ : \parallel v - u \parallel < p/2^{k-d} \ \mid \ v \neq v_\alpha\right] < 1/2$$

Here the probability is taken over the randomly chosen t_i's. In general, a lattice point v has the form

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

for some integers $\beta, b_1, \ldots, b_d$. Observe that if $\beta = \alpha \pmod p$, but $v \neq v_\alpha$ then trivially $\parallel v - u \parallel > p/2^{k-d}$. This follows since at least one of the components of $v - u$ is bigger in absolute value than $p/2^{k-d}$. Now, suppose $\beta \neq \alpha$. Then

$$\Pr[\parallel v - u \parallel > p/2^{k-d}] \ \geq \ \Pr[\exists i \ : \ \mathrm{dist}_p(t_i\beta, u_i) > p/2^{k-d}] \geq$$

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

Since $\beta \neq \alpha$ there are exactly $p-1$ values of $\beta \bmod p$ to consider. Hence, we obtain

$$\Pr\left[\exists v \in L \ : \parallel v - u \parallel < p/2^{k-d} \mid \ \neq v_\alpha\right] < (p-1) \cdot \left(\frac{5}{2^{k-d}}\right)^d < \frac{1}{2}$$

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

We now apply Theorem 2.2 to the lattice L and the vector u. According to the theorem we can find in polynomial time a lattice vector $w \in L$ such that

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

By Theorem 2.3, with probability $\geq 1/2$, we have that $w = v_\alpha = (r_1, \ldots, r_d, \alpha)$. The last component of this vector contains the hidden element α. This completes the proof of Theorem 2.1. □

In the previous section we noted that the above techniques can be used to solve a more general problem. Let α, β be two integers in the range $[1, p-1]$. Let $O(x)$ be the function defined by

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

where $k = 3\sqrt{\log p}$. The above techniques can be used to recover α, β in expected polynomial time from a an oracle for O. This is done by construction $t_1, \ldots, t_d$ as above and using the lattice:

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

This lattice has dimension $d+2$. 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 oracles answers. Hence, Theorem 2.2 can be applied again to recover α, β.

2.2 Applications

2.2.1 MSB's of Diffie-Hellman and related problems

Proof of Theorem 1.1. Given the oracle for the computing $\mathrm{MSB}_k(g^x, g^y)$ for $k > \sqrt{\log p}$ we construct an oracle $O(r)$ for the hidden number problem as follows: define $h = g^y$ and $\alpha = g^{xy}$. Then

$$O(r) = \mathrm{MSB}_k(\alpha h^r \bmod p) = \mathrm{MSB}_k(DH_g(g^{x+r}, g^y))$$

Theorem 2.1 now directly implies Theorem 1.1. We may assume that g^y is a generator since y can be made random. □

Theorem 1.3 is proved in a similar way. We construct the hidden number problem oracle $O(r)$ using the following equalities.

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

$$\mathrm{SH}(g^{ab}, g^{a+rab}, g^b) = g \cdot (g^b)^r$$

$$\mathrm{OK}_g(g^{a(r+b)}, g^a) = g^b \cdot g^r$$

2.2.2 Factoring with hints

Proof Sketch of Theorem 1.4: Let $n = pq$ and $x_i, i \leq k$ such that $x_i \bmod p < p/2^{3\sqrt{\log n}}$. Then $x_i = a_i p + b_i, b_i < p/2^{3\sqrt{\log n}}$. Thus, $qx_i \bmod n = b_i q < n/2^{3\sqrt{\log n}}$. Now q can be considered as a hidden element mod$n$. Proof and the methods of Theorem 2.1 can be modified (to account for the non-invertible elements in $\mathbb{Z}_n$) to recover the hidden element given random x_i satisfying the above condition.

We discuss the issue of predicting truncated Random Number Sequences in the final version.

2.3 A nonuniform result

We show that there exist a polynomial number of advice bits which depend only on p which enable one to solve the hidden number problem in polynomial time using an oracle which returns only $\log\log p$ bits.

THEOREM 2.4 *Let p be a prime and g a generator of $\mathbb{Z}_p^*$. Set $m = [\log\log p]$. For a hidden α define the function $O(r) = \mathrm{MSB}_m(\alpha \cdot g^r \bmod p)$. Then there exists a polynomial time algorithm which takes a polynomial number of advice bits depending only on p and recovers α given an oracle for the function O.*

The proof relies on a new lattice rounding technique to solve the hidden number problem. We intend to query the oracle O at uniformly and independently chosen inputs $r_1, \ldots, r_d$. The value of d will be specified later. Set $t_i = g^{r_i} \bmod p$ then the oracle O will output integers $a_1, \ldots, a_d, a_{d+1}$ such that

$$|(\alpha t_i \bmod p) - a_r| < R \qquad \text{and} \qquad |(\alpha \bmod p) - a_{d+1}| < R$$

To keep the discussion general we do not specify the value of R for now. Note that a_{d+1} is found by querying the oracle at $r = 0$. Our goal is to recover α from this information.

The algorithm we propose works as follows: first we form the lattice L used in Section 2.1. Recall that L is spanned by the rows of the matrix:

$$L = \begin{pmatrix} p & 0 & \dots & 0 & 0 \\ 0 & p & \dots & 0 & 0 \\ & \vdots & & & \vdots \\ 0 & 0 & \dots & p & 0 \\ t_0 & t_1 & \dots & t_d & 1 \end{pmatrix}$$

As we did in Section 2.1 we form that vector $u = (a_1, \dots, a_d, a_{d+1})$. Define the lattice vector $v_\alpha \in L$ by

$$v_\alpha = (t_1\alpha \bmod p, t_2\alpha \bmod p, \dots t_d\alpha \bmod p, \alpha)$$

Given u our algorithm will output the vector v_α. Clearly α can be recovered from the vector v_α. To construct the vector v_α from u we intend to use a different basis of the lattice L. Let $b_1, \dots, b_{d+1}$ be some basis of L. Given u the algorithm performs two steps.

1. Write $u = \sum_{i=1}^{d+1} y_i b_i$ for some $y_i \in \mathbb{R}$.
2. Set $v = \sum_{i=1}^{d+1} \lfloor y_i \rceil b_i$ where $\lfloor y_i \rceil$ is the integer closest to y_i.

The question is when does this type of rounding produce the right answer, i.e. when does $v = v_\alpha$?. The next lemma gives a tight criteria for when this occurs. We need the following notation: for a matrix W whose columns are the vectors $w_1, \dots, w_d$ define

$$L_{1,\infty}(W) \stackrel{def}{=} \max_i L_1(w_i)$$

where $L_1(w_i)$ is the L_1 norm of w_i.

Lemma 2.5 *Let A be the matrix whose rows are the basis vectors b_i. If $L_{1,\infty}(A^{-1}) < 1/(2R)$ then the vector v constructed above satisfies $v = v_\alpha$.*

Proof Since the b_i form a basis of L we know that there exists a vector $x \in \mathbb{Z}^{d+1}$ such that $xA = v_\alpha$. Let $y \in \mathbb{R}^{d+1}$ be the vector such that $yA = u$. We need to prove that the theorem's hypothesis implies $\lfloor y_i \rceil = x_i$ for every coordinate i. Here x_i, y_i are the i'th coordinates of the vectors x, y respectively. Since x_i is an integer it suffices to prove that $|x_i - y_i| < 1/2$ for all i.

Set $W = A^{-1}$ and $W = (w_{i,j})$. Then

$$\begin{cases} xA = v_\alpha \\ yA = u \end{cases} \implies (x-y)A = v_\alpha - u \implies x - y = (v_\alpha - u)A^{-1} \implies$$

$$|x_i - y_i| \le \sum_{j=1}^{d+1} |w_{j,i}| \cdot |(v_\alpha)_j - u_j| \le R \sum_{j=1}^{d+1} |w_{j,i}| \le R \cdot L_{1,\infty}(W) < \frac{1}{2}$$

This completes the proof of the lemma. □

At this point we need to recall the notion of the dual lattice. Let L be a lattice spanned by the vectors $b_1, \dots, b_d \in \mathbb{R}^d$. The dual lattice of L, denoted L^*, is the set of vectors

$$L^* = \left\{ w \in \mathbb{R}^d \quad \text{s.t.} \quad \forall u \in L \ : \ (u, w) \in \mathbb{Z} \right\}$$

One can readily show that if A is the matrix whose rows are $b_1, \dots, b_d$ then the dual lattice is spanned by the columns of A^{-1}. Using this terminology Lemma 2.5 implies that we must find a basis of the dual lattice whose $L_{1,\infty}$ norm is as small as possible.

By inverting (and transposing) the matrix L we obtain that the dual lattice of our lattice L is spanned by the rows of the matrix:

$$L^* = \frac{1}{p}\begin{pmatrix} 1 & 0 & \dots & 0 & -t_1 \\ 0 & 1 & \dots & 0 & -t_2 \\ & \vdots & & & \vdots \\ 0 & 0 & \dots & 1 & -t_d \\ 0 & 0 & \dots & 0 & p \end{pmatrix}$$

We prove that with high probability (over the choice of the random t_i) the lattice L^* has a basis with low $L_{1,\infty}$ norm.

THEOREM 2.6 *Let p be a prime and $d > 4 + \log p + \log\log p$. Let $t_1, \dots, t_d$ be integers chosen uniformly and independently at random in the range $[0, p-1]$. Consider the corresponding lattice L^*. Then with probability $1/2$ there exists a basis of this lattice with $L_{1,\infty} < 3\log^2 p/p$.*

The proof relies on the following lemma.

Lemma 2.7 *Let $p, d, t_1, \dots, t_d$ as above. Set $m = \lceil \log\log p \rceil + 4$ and $n = m + \lfloor \log p \rfloor$. By assumption $d > n$. Then with probability $1/2$ for all $k = m+1, m+2, \dots, n$ there exist subsets $S_k \subseteq \{1, \dots, k-1\}$ satisfying:*

$$2^{n-k} \le t_k + \sum_{i \in S_k} t_i \bmod p < 2 \cdot 2^{n-k}$$

Proof Let $k \in [m, n]$ be an integer. For each non empty set $S \subseteq \{1, \dots, k-1\}$ define the random variable $T_S = t_k + \sum_{i \in S} t_i \bmod p$. The set $\{T_S\}_{S \in \subseteq \{1,\dots,k-1\}}$ is a collection of pair wise independent random variables each uniformly distributed in the range $[0, p-1]$. Similarly, define the indicator random variable X_S as

$$X_S = \begin{cases} 1 & \text{if} \quad 2^{n-k} \le T_S < 2 \cdot 2^{n-k} \\ 0 & \text{otherwise} \end{cases}$$

For notational convenience we define $X_\emptyset$ to be the fixed value $1/2^{k-m+1}$. The set $\{X_S\}$ is also a collection of pair wise independent random variables with $E[X_S] = 2^{n-k}/p \ge 1/2^{k-m+1}$. We set $Z = \sum_S X_S$. Then by pair wise independence we obtain

$$E[Z] \ge 2^{k-1} \cdot \frac{1}{2^{k-m+1}} = 2^{m-2} \quad \text{and} \quad \mathrm{Var}[Z] = \sum_S \mathrm{Var}[X_S] < 2^{m-1}$$

Chebyshev's inequality now gives:

$$\Pr[Z < 1] \le \Pr\left[|Z - E[Z]| > 2^{(m-3)/2}\sqrt{\mathrm{Var}(Z)}\right] < \frac{1}{2^{m-3}}$$

Hence, $Z \ge 1$ with probability at least $1 - \frac{1}{2^{m-3}}$. When $Z \ge 1$ there exists at least one set S_k satisfying the required condition. Therefore, the required S_k exists with probability at least $1 - \frac{1}{2^{m-3}}$. We conclude that the probability that S_k exists for all $k = m+1, m+2, \dots, n$ satisfies

$$\Pr[\forall k \ : \ S_k \text{ exists}] \ge 1 - \frac{n-m}{2^{m-3}} > \frac{1}{2}$$

This concludes the proof of the lemma. □

Proof of Lemma 2.6 The proof follows by multiplying the matrix L^* above on the left by two unimodular matrices. Let n, m be defined as in Lemma 2.7. Then since d satisfies $d > n + m$ then

with probability at least $1/2$ there exist sets $S_{m+1}, S_{m+2}, \ldots, S_n$ satisfying the conditions of Lemma 2.7. For all $k \leq m$ and $k > n$ we define $S_k = \emptyset$. Define the $d+1 \times d+1$ matrix $U = (u_{i,j})$ by

$$u_{i,j} = \begin{cases} 1 & \text{if} \quad j \in S_i \text{ or } i = j \\ 0 & \text{otherwise} \end{cases}$$

All the elements on the diagonal of U are 1. Furthermore, since the sets S_k satisfy $S_k \subseteq \{1, \ldots, k-1\}$ for all k it follows that U is a lower diagonal matrix and hence unimodular. The rows of the matrix UL^* form a basis of the dual lattice L^*. Multiplying L^* by U on the left has the effect of replacing each basis vector b_k by the vector $b_k + \sum_{i \in S_k} b_i$.

After rearranging the rows of UL^* and reducing the elements in the last column modulo p the resulting basis has the following structure:

$$\frac{1}{p} \begin{pmatrix} & & u_1 \\ & & \vdots \\ & & u_{n-m} \\ & B & t_1 \\ & & \vdots \\ & & t_m \\ & & p \end{pmatrix}$$

where B is a $d+1 \times d$ matrix with $0,1$ entries. By the choice of the sets S_k we can arrange things so that the elements $u_1, \ldots, u_{n-m}$ satisfy $2^{k-1} \leq u_k < 2^k$. This property of the u_k enables one to write any integer x in the range $[1, 2^k]$ as $x = \sum_{i=1}^{k-1} x_i u_i$ where all $x_i = 0, 1, 2, 3$. Specifically, any integer x in the range $[0, p]$ can be written as $x = \sum_{i=1}^{n-m} x_i u_i$ where $x_i = 0, 1, 2, 3$.

We now transform the above basis into a basis with low $L_{1,\infty}$ norm. Let $b_1, \ldots, b_{d+1}$ be the basis vectors. Let $x^{(k)}$ be the right most entry in the vector b_k, e.g. $x^{(1)} = u_1$, $x^{(2)} = u_2$, $x^{(d+1)} = p$, etc. We replace the basis vector b_k by the vector $b_k - \sum_{i=1}^{k-1} x_i^{(k)} b_i$ where the $x_i^{(k)}$ satisfy $x^{(k)} = \sum_{i=1}^{k-1} x_i^{(k)} x^{(i)}$ with $x_i^{(k)} = 0, 1, 2, 3$. The unimodular matrix which performs this transformation is:

$$V = \begin{pmatrix} 1 & 0 & 0 & 0 & \ldots & 0 \\ -x_1^{(2)} & 1 & 0 & 0 & \ldots & 0 \\ -x_1^{(3)} & -x_2^{(3)} & 1 & 0 & \ldots & 0 \\ & \vdots & & & \vdots & \\ -x_1^{(d+1)} & -x_2^{(d+1)} & -x_3^{(d+1)} & & \ldots & 1 \end{pmatrix}$$

The resulting basis has the structure

$$\frac{1}{p} \begin{pmatrix} & & 1 \\ & & 0 \\ & B' & \vdots \\ & & 0 \end{pmatrix}$$

where B' is a $d+1 \times d$ matrix whose entries are all less than $3(n-m) < 3 \log p$ in absolute value. Thus, the $L_{1,\infty}$ norm of the resulting basis is at most $3 \log^2 p / p$. This completes the proof of Theorem 2.6. $\square$

The proof of Theorem 2.4 follows from Theorem 2.6. The advice bits used by the rounding algorithm will be a set of integers $t_1, \ldots, t_d$ along with a basis of L^* satisfying the conditions of Theorem 2.6. The theorem shows that indeed there exist integers $t_1, \ldots, t_d$ for which the required basis

exists[1]. Since the $L_{1,\infty}$ norm of such a basis is less than $(\log\log p)/p$, Lemma 2.5 shows that for the rounding algorithm to find the correct α it suffices that the oracle return only $\log logp$ bits. This completes the proof of Theorem 2.4. □

We note that constructing a basis satisfying the conditions of Lemma 2.6 is likely to be hard. It requires one to solve certain instances of the modular subset sum problem. Impagliazzo and Naor [11] noted that these instanced are likely to be hard.

2.4 The case of a small generator

In the previous section we proved that a hidden number can be recovered using an oracle that output $3\sqrt{\log p}$ bits. Recall that the oracle output the $3\sqrt{\log p}$ most significant bits of $\alpha \cdot g^x \bmod p$ for a given x. The generator g was an arbitrary generator of $\mathbb{Z}_p^*$. In this sections we show that if the generator g is less than 2^k then in fact only k bits suffice to reconstruct a hidden number. For instance, when $g = 2$ only the most significant bit is needed.

For a generator g of $\mathbb{Z}_p^*$ we define the significant bits 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.

THEOREM 2.8 *Let α be some integer in the range $[1, p-1]$. Let O be a function defined by $O(x) = SB_g(\alpha g^x \bmod p)$ for some generator g of $\mathbb{Z}_p^*$. Then there exists an algorithm which given access to an oracle computing the function 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 $U = t \cdot \frac{p}{g^r}$ where $t = 1, \ldots, g^r$. 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 L and U to be used at 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. □

In the case when $g = 2$ the function $SB_g(x)$ is simply the $\mathrm{MSB}_1(x)$ function used in Section 2. Hence, in this case the theorem proves that the most significant bit is sufficient for recovering the hidden number α.

[1]Theorem 2.6 shows even more; it shows that for most tuples $(t_1, \ldots, t_d)$ such a basis is known to exist.

2.4.1 A variant of Diffie-Hellman and its bit security

As was mentioned in the introduction, Theorem 2.8 suggests a new variant of the Diffie-Hellman protocol. For a fixed prime p define the Diffie-Hellman function $DH_h(h^x, h^y)$ to be $h^{xy} \pmod p$. Suppose we have an oracle $O(a,b)$ which given $a = h^x$ and $b = h^y$ outputs $\mathrm{MSB}(h^{xy} \bmod p$. Observe that $O(h^{x+r}, h^y) = \mathrm{MSB}(h^{xy}(h^y)^r \bmod p)$. Hence, if we set $g = h^y$ and $\alpha = h^{xy}$ then using the oracle $O(a,b)$ one can easily construct an oracle O' defined as $O'(r) = \mathrm{MSB}(\alpha g^r)$. Theorem 2.8 now shows that when $h^y = g = 2$ the oracle O' enable one to recover $\alpha = h^{xy}$ from h^x, h^y in polynomial time. This shows that when $h^y = 2 \bmod p$ the MSB of the Diffie-Hellman function is as hard as the entire function. We write this in the following corollary:

Corollary 2.9 *Let O be an oracle that given $g, g^x \bmod p$ computes the most significant bit of $DH_g(g^x, 2)$. Then given the oracle O it is possible to recover $DH_g(g^x, 2)$ in polynomial time.*

This corollary suggests a new variant of the Diffie-Hellman protocol. Say Alice and Bob wish to perform secret key exchange. We assume that they have already agreed on a prime p. Alice picks a random number x in the range $[1, p-1]$ such that $\gcd(x, p-1) = 1$. She computes $g = 2^x \pmod p$ and sends g to Bob. Bob picks a random number y in the range $[1, p-1]$ and sends g^y to Alice. The secret they agree on is $\alpha = 2^y \pmod p$. 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 has to compute the value $DH_g(g^y, 2)$. Corollary 2.9 shows that computing the MSB of the secret shared by Alice and Bob using this scheme is as hard as computing the entire secret. Thus this new scheme has the advantage of provable bit security.

A drawback of this new variant is that it relies on the hardness of $DH_g(g^y, 2)$ for it security. 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. Thus, at least the standard heuristic discrete log argument supports the security of this variant.

2.4.2 A more general rounding technique

Theorem 2.8 is very useful when the generator g is small. Unfortunately the proof does not seem to generalize to the case of arbitrary generator. In this section we give a more powerful proof of a theorem slightly weaker than Theorem 2.8. This proof is likely to generalize to the case of arbitrary generator though at the present this is still an open problem.

Let L^* be the lattice spanned by the rows of the matrix:

$$A = \frac{1}{p} \begin{pmatrix} 1 & 0 & \dots & 0 & -g \\ 0 & 1 & \dots & 0 & -g^2 \\ & \vdots & & & \vdots \\ 0 & 0 & \dots & 1 & -g^d \\ 0 & 0 & \dots & 0 & p \end{pmatrix}$$

where $d > \log_g p$. Lemma 2.5 shows that to prove Theorem 2.8 it suffices to construct a basis of L^* with low $L_{1,\infty}$ norm. Write $p = \sum_{i=0}^{d} b_i g^i$ where all b_i satisfy $0 \le b_i < g$. This is possible since

$d > \log_g p$. We multiply the basis A by a unimodular matrix to obtain a new basis B as follows:

$$B = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ -g & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & -g & 1 & \dots & 0 & 0 & 0 \\ & \vdots & & & \vdots & & \\ 0 & 0 & 0 & \dots & -g & 1 & 0 \\ b_1 & b_2 & b_3 & \dots & b_{d-1} & b_d & 1 \end{pmatrix} \cdot A = \frac{1}{p} \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 & -g \\ -g & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & -g & 1 & \dots & 0 & 0 & 0 \\ & \vdots & & & \vdots & & \\ 0 & 0 & 0 & \dots & -g & 1 & 0 \\ b_1 & b_2 & b_3 & \dots & b_{d-1} & b_d & b_0 \end{pmatrix}$$

For this basis we have $L_{1,\infty}(B) = gd/p$. Therefore, taking $R = \frac{p}{3gd}$ in Lemma 2.5 guarantees that the rounding algorithm of Section 2.3 will recover α. Taking $R = \frac{p}{3dg}$ corresponds to an oracle which returns more than $\log 3dg$ of the most significant bits of αg^k. Since $\log 3dg \leq \log g + \log\log p + 2$ we see that the hidden number problem can be solved using this many bits. For small g this is not as strong as Theorem 2.8. However, the above algorithm uses the general MSB oracle rather than the special SB oracle used in the algorithm of Theorem 2.8.

We are hopeful that the above proof can be generalized to show that a small number of bits suffices to solve the hidden number problem for an arbitrary generator g. To do so one must construct a basis of L^* with low $L_{1,\infty}$ norm for a general g.

References

[1] W. Alexi, B. Chor, O. Goldreich, C. Schnorr, "RSA and Rabin functions: Certain parts are as hard as the whole", SIAM Journal on Computing, Nov 1988, Vol 7 No 2.

[2] M. Bellare, Micali. S, "Non-interactive oblivious transfer and applications", Crypto 89, pp 547-557.

[3] M. Blum, Micali .S, " How to generate cryptographically strong sequences of pseudo-random bits", 1982, pp 112-117.

[4] L. Babai, "On Lovasz' lattice reduction and the nearest lattice point problem", Combinatorica, Vol. 6, 1986, pp. 1–13.

[5] Coppersmith, the Don, "Finding a Small Root of a Univariate Modular Equation," IBM Research Report RC 20223, October 11, 1995; revised November 8, 1995. To appear, Eurocrypt 96.

[6] D. Coppersmith, "Finding a Small Root of a Bivariate Integer Equation; Factoring with high bits known," IBM Research Report RC 20280, 11/17/95. To appear, Eurocrypt 96.

[7] W. Diffie, Hellman E.M., "New directions in cryptography", IEEE. Trans. Inform. Theory, IT-22, No. 6, pp 644-654. (nov 1976)

[8] T. Elgamal, "A public-key cryptosystem and a signature scheme based on Discrete Logarithms", IEEE Trans. on Info. Theory, IT-31, No 4, pp 469-472. (july 1985).

[9] A. Frieze, J. Hastad, R. Kannan, J. Lagarias, A. Shamir, "Reconstructing Truncated Integer Variables Satisfying Linear Congruences", SIAM J. of Computing, Vol 17, No 2, 1988.

[10] O.Goldreich, L.A. Levin, "Hard Core bits based on Any one-way function", STOC 89.

[11] R. Impagliazzo, Naor M., "Efficient cryptographic schemes provably as secure as subset sum", Proc. FOCS 1989, pp. 236–241.

[12] N. Koblitz, "A course in Number Theory and Cryptography", Springer Verlag 1987.

[13] A. Lenstra, H. Lenstra, L. Lovasz, "Factoring polynomial with rational coefficients", Mathematiche Annalen, Vol. 261, 1982, pp. 515–534.

[14] D. Long, Wigderson A, "The discrete logarithm hides $O(logn)$ bits" Vol 17, No 2, 1988.

[15] T. Okamoto, "Encryption and Authentication Schemes Based on Public Key Systems, Ph. D Thesis, The Univ. of Tokyo, (1988).

[16] K. Sakurai, Shizuya H, "Relationships among Computational powers of Breaking Discrete Log Cryptosystems", Eurocrypt 95, pp 341-351.