The Modular Inversion Hidden Number Problem

Dan Boneh[1], Shai Halevi[2], and Nick Howgrave-Graham[2]

[1] Department of Computer Science, Stanford University, CA, USA
dabo@cs.stanford.edu
[2] IBM T.J. Watson Research Center, NY, USA
{shaih,nahg}@watson.ibm.com

Abstract. We study a class of problems called Modular Inverse Hidden Number Problems (MIHNPs). The basic problem in this class is the following: Given many pairs $\langle x_i,\ \mathrm{MSB}_k\left((\alpha + x_i)^{-1} \bmod p\right)\rangle$ for random $x_i \in \mathbb{Z}_p$ the problem is to find $\alpha \in \mathbb{Z}_p$ (here $\mathrm{MSB}_k(x)$ refers to the k most significant bits of x). We describe an algorithm for this problem when $k > (\log_2 p)/3$ and conjecture that the problem is hard whenever $k < (\log_2 p)/3$. We show that assuming hardness of some variants of this MIHNP problem leads to very efficient algebraic PRNGs and MACs.

Keywords: Hidden number problems, PRNG, MAC, Approximations, Modular inversion, Lattices, Coppersmith's attack

1 Introduction

In recent years several new complexity assumptions were used to construct efficient cryptosystems. The Decision Diffie-Hellman assumption (DDH) was used to construct chosen ciphertext secure encryption [7] and number theoretic pseudo random functions [15]. The Strong RSA assumption was used to construct efficient signature schemes [10,8]. In this paper we introduce a new class of algebraic complexity assumptions which we call the Modular Inverse Hidden Number Problem (MIHNP). Using MIHNP we construct an efficient number theoretic Pseudo Random Number Generator (PRNG) and an efficient MAC. The basic step in evaluating the MAC and the PRNG is one modular inversion modulo a moderate size prime. No expensive exponentiations are needed.

To describe the basic MIHNP we introduce the following notation that will be used throughout the paper: For an m-bit prime p and $y \in \mathbb{Z}_p$ we use $\mathrm{MSB}_k(y \bmod p)$ to denote any integer $Y \in \mathbb{Z}_p$ satisfying $|Y - y| < p/2^k$. In other words, Y is an approximation to y that (usually) matches y on the k most significant bits. We write $\mathrm{MSB}_k(y)$ where there is no ambiguity about the modulus p. In addition, throughout the paper we define the inverse of $0 \in \mathbb{Z}_p$ to be 0. We consistently use Greek characters to denote hidden values.

MIHNP. An instance of the basic MIHNP problem is as follows: let p be a fixed m-bit prime and k, n be positive integers. Let α be a random hidden element of $\mathbb{Z}_p$. We are given p, k, and $\left\langle x_i, \mathrm{MSB}_k(\frac{1}{\alpha + x_i})\right\rangle$ for *random* values $x_1, \ldots, x_n$.

C. Boyd (Ed.): ASIACRYPT 2001, LNCS 2248, pp. 36–51, 2001.

The problem is to find α. The δ-MIHNP assumption states that there is no polynomial time algorithm for the Basic-MIHNP problem whenever $k < \delta m$.

In other words, given *many* approximations to $(\alpha + x_i)^{-1} \bmod p$ for random $x_i \in \mathbb{Z}_p$ the problem is to find α. The parameters m, n, k are security parameters for the problem. Note that when $n > 2(m/k)$ the hidden number α is uniquely defined with high probability and consequently there is a unique answer to this problem. We show a lattice-based algorithm, that solves this problem when $k > m/3$. We also explain why this algorithm does not extend to solve it for $k < m/3$. As our algorithm represents the current state-of-the-art in lattice reduction techniques, we conjecture that such techniques cannot be used beyond the $m/3$ bound. More generally, we conjecture that the δ-MIHNP assumption holds for any $\delta < 1/3$. In the next section we introduce several variants of MIHNP that are useful for cryptographic constructions. We also show that the MIHNP problem has a simple limited random self reduction.

MIHNP is closely related to several other Hidden Number Problems (HNPs). Hidden number problems were introduced in [4] where they were used to prove the bit security of the Diffie-Hellman secret in $\mathbb{Z}_p$. The standard HNP is as follows: let $\alpha \in \mathbb{Z}_p$ be a hidden random number. Given $\text{MSB}_k(\alpha \cdot x_i \bmod p)$ for random $x_1, \ldots, x_n \in \mathbb{Z}_p$ the problem is to find α. The standard, HNP can be efficiently solved when $k = O(\sqrt{|p|})$, and this solution forms the basis of the bit-security result in [4] (as well as an attack on weak versions of the Digital Signature Algorithm (DSA), see [13]). This is in contrast to MIHNP which appears to be hard even when k is a constant fraction of $|p|$.

2 Approximate Modular Inversion Problems

We introduce several variants of the basic MIHNP and study their properties. The first variant of MIHNP, which we call the *Computational-MIHNP*, is useful for constructing a MAC.

Computational-MIHNP: An instance of the C-MIHNP problem is as follows: let p be a fixed m-bit prime and k, n be positive integers. Let α be a random hidden element of $\mathbb{Z}_p$. We are given p, k, and $\left\langle x_i, \text{MSB}_k(\frac{1}{\alpha + x_i}) \right\rangle$ for *random* values $x_1, \ldots, x_n$. The problem is to construct another pair $\left\langle x, \text{MSB}_k(\frac{1}{\alpha + x}) \right\rangle$ for some $x \neq x_i$. The δ-CMIHNP assumption states that there is no polynomial time algorithm for this problem whenever $k < \delta m$.

Although we cannot prove the equivalence of this problem to the basic MIHNP, we do not know of an algorithm for solving it without first discovering the secret α from the given input. The second variant, which we call the *Decisional-MIHNP* is useful for constructing PRNGs.

Decisional-MIHNP: An instance of the D-MIHNP problem is as follows: let p be a fixed m-bit prime and k, n be positive integers. Let α be a random hidden element of $\mathbb{Z}_p$. We are given p and k. The problem is to distinguish the following two ensembles:

$$\left\{ x_1, \mathrm{MSB}_k(\tfrac{1}{x_1+\alpha}), \ldots, x_n, \mathrm{MSB}_k(\tfrac{1}{x_n+\alpha}) \right\} \quad \text{and}$$

$$\left\{ x_1, \mathrm{MSB}_k(r_1), \quad \ldots, \quad x_n, \mathrm{MSB}_k(r_n) \right\}$$

where $\alpha, x_1, \ldots, x_n, r_1, \ldots, r_n$ are chosen uniformly at random in $\mathbb{Z}_p$. The δ-DMIHNP assumption states that no polynomial time algorithm can distinguish these two ensembles with non-negligible advantage whenever $k < \delta m$.

As before, we cannot reduce this problem to either of the previous problems, but we know of no algorithms for D-MIHNP, other than first finding the hidden element α. In a sense, it seems that the tools that we have for designing algorithms for these problems are too crude to distinguish between these variants.

This situation is somewhat analogous to the situation with the various discrete-logarithm assumptions. The basic MIHNP can be viewed as an analog of the Discrete-Log Problem (DLP): given $g^\alpha \bmod p$ find the hidden number α. Just as DLP is often insufficient for cryptographic constructions, we need stronger assumptions that the basic MIHNP for the constructions in this paper. The C-MIHNP can be viewed as an analog of the Computational Diffie-Hellman assumption (CDH), and D-MIHNP is the analog of the Decision Diffie-Hellman assumption (DDH). As is the case with the various MIHNP problems, we also do not have reductions between the various discrete-log problems, yet the only algorithms that we know for solving any of them involve solving discrete-log.

2.1 Random Self Reduction for MIHNP

The MIHNP problem has a simple limited random self reduction among instances modulo the same prime p. The reduction shows that for a prime p if finding $\alpha \in \mathbb{Z}_p$ is hard for a worst case α then it is also hard for a random $\alpha \in \mathbb{Z}_p$.

Suppose there is an algorithm $\mathcal{A}$ that solves the Basic-MIHNP problem with probability ϵ, where the probability is taken over the choice of the x_i's and also over the choice of α. We show that this implies an algorithm $\mathcal{B}$ for solving Basic-MIHNP that works for any fixed α with probability ϵ, where this time the probability is over the choice of the x_i's only.

Given an instance of Basic-MIHNP, $\langle x_i, y_i \rangle$, $i = 1, \ldots, n$, algorithm $\mathcal{B}$ picks a random $r \in \mathbb{Z}_p$, and runs algorithm $\mathcal{A}$ on the Basic-MIHNP problem defined by the tuples $\langle x_i + r, y_i \rangle$, $i = 1, \ldots, n$.

Note that if the original MIHNP instance corresponds to the hidden number α, then the new instance will correspond to the hidden number $\alpha' = \alpha - r$, which is random and independent of the x_i's. It follows that with probability ϵ, the algorithm $\mathcal{A}$ indeed returns α', and then $\mathcal{B}$ can add back r to recover α.

We call this a limited random self reduction, since we only randomize the solution α, and not the elements $x_1, \ldots, x_n$. The computational MIHNP and decisional MIHNP have similar limited random self reductions.

3 Security Analysis of the MIHNP

In this section we analyze the security of MIHNP. We show how to apply the currently known technology in algebraic cryptanalysis to MIHNP, and demonstrate the limitations of that technology when applied to this problem. We know of no better way to distinguish the pairs $\langle x_i, \text{MSB}_k((\alpha + x_i)^{-1})\rangle$ from random, other than to actually recover the secret α (and use the knowledge of this to verify the bits), and so this is the problem we address. That is, we assume that we have a system of equations

$$(\alpha + x_i)(b_i + \epsilon_i) = 1 \pmod{p} \qquad i = 0, \ldots, n, \tag{1}$$

where $\alpha \in Z_p$ is the (large, secret) variable we aim to discover, the x_i's are known, but randomly chosen elements of Z_p, the b_i's are the known most significant bits, and the ϵ_i are variables that correspond to the unknown low order bits, so we have $|\epsilon_i| \leq 2^{m-k}$ for all i. Observe that once we find any of the ϵ_i's we can discover the secret α immediately, from the fact that $\alpha = 1/(b_i + \epsilon_i) - x_i \pmod{p}$. However, as we shall see, typically we find all the ϵ_i simultaneously, or none at all.

We attempt to solve MIHNP using lattice techniques. We set up a lattice that incorporates the relations from Eq. (1), so that the bound on the size of the ϵ_i's will correspond to some small vector in the lattice. If we can make the argument that this vector is by far smaller than any other vector in this lattice, then we could use the LLL lattice reduction algorithm [14] to find it, thereby recovering the ϵ_i's. This framework was used in [4] to solve the original HNP.

Looking at Eq. (1), however, we find that these relations cannot be used directly to set up a lattice. The reason is that each of these relations has a term of the form $\alpha \cdot \epsilon_i$, where α is unbounded (i.e., it can be as large as p), and the ϵ_i's change from one relation to the next. To use in a lattice, one must first "linearize" these relations, and doing so would introduce a new unbounded variable for each of the products $\alpha\epsilon_i$. (We stress that current technology has no problem handling either changing small unknowns such as the ϵ_i, or fixed large unknowns such as α. It is *the product of the two* that makes this problem hard.)

We are therefore forced to eliminate the unknown α from the relations of Eq. (1), before we can use them to set up a lattice. Given the $n + 1$ relations from Eq. (1), we eliminate the unknown α, and produce n relations of the form:

$$(x_i - x_0)(b_0 + \epsilon_0)(b_i + \epsilon_i) - (b_0 + \epsilon_0) + (b_i + \epsilon_i) \;=\; 0 \pmod{p} \tag{2}$$

These relations are already in a form that is amenable for use in a lattice, and we can apply to them (an extension of) the techniques from [6,12], as we now explain. We start by re-writing the left hand side of Eq. (2) as a polynomial in the unknowns ϵ_0 and ϵ_i, namely:

$$f_i(\epsilon_0, \epsilon_i) \stackrel{\text{def}}{=} (x_i - x_0)\epsilon_0\epsilon_i + (b_0(x_i - x_0) + 1)\epsilon_i + (b_i(x_i - x_0) - 1)\epsilon_0 + (b_0 b_i(x_1 - x_0))$$

Notice that the coefficients of this polynomial are known to us (since we know all the b_i's and x_i's), and therefore we can set up a lattice based on their values. To simplify notation, we denote below $f_i(\epsilon_0, \epsilon_i) = A_i\epsilon_0\epsilon_i + B_i\epsilon_i + C_i\epsilon_0 + D_i$.

3.1 First Attempt: A Linear Approach

As a first attempt at a solution, we set up a lattice of dimension $3n+2$ as follows. The lattice is spanned by the rows of a real matrix M that has the following general structure:

$$M = \begin{pmatrix} E & R \\ 0 & P \end{pmatrix}$$

where E and P are diagonal matrices of dimensions $(2n+2) \times (2n+2)$ and $n \times n$, respectively, and R is a $(2n+2) \times n$ matrix. Each of the first $2n+2$ rows of M is associated with one of the terms in relations from Eq. (2) (i.e., the constant term, the terms ϵ_i, and the terms $\epsilon_0\epsilon_i$), and each of the last n columns is associated with one of the n relations.

The matrix R incorporates the relations themselves. The (i,j) entry in this matrix is just the coefficient in the j'th relation of the term corresponding to row i. The diagonal entries of the matrix P are all equal to p, and the diagonal entries of the matrix E correspond to the bounds on the terms associated with each row. Specifically, if the term which is associated with row i is bounded by B, then entry (i,i) in E is equal to $1/B$. That is, the row corresponding to the constant term has diagonal entry 1, rows corresponding to ϵ_i have diagonal entries $1/2^{m-k}$, and rows corresponding to $\epsilon_0\epsilon_j$ have diagonal entries $1/2^{2(m-k)}$. An example for the matrix M for $n = 2$ is given in Figure 1.

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & D_1 & D_2 \\ 0 & 2^{k-m} & 0 & 0 & 0 & 0 & C_1 & C_2 \\ 0 & 0 & 2^{k-m} & 0 & 0 & 0 & B_1 & 0 \\ 0 & 0 & 0 & 2^{k-m} & 0 & 0 & 0 & B_2 \\ 0 & 0 & 0 & 0 & 2^{2(k-m)} & 0 & A_1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2^{2(k-m)} & 0 & A_2 \\ 0 & 0 & 0 & 0 & 0 & 0 & p & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & p \end{pmatrix} \begin{matrix} \text{row corresponds to} \\ \ldots\ldots\ 1 \\ \ldots\ldots\ \epsilon_0 \\ \ldots\ldots\ \epsilon_1 \\ \ldots\ldots\ \epsilon_2 \\ \ldots\ldots\ \epsilon_0\epsilon_1 \\ \ldots\ldots\ \epsilon_0\epsilon_2 \\ \\ \\ \end{matrix}$$

Fig. 1. The matrix M for the case $n = 2$.

We can now view each one of the relations of Eq. (2) as holding over the integers, by explicitly introducing the appropriate multiple of p. Namely, we have:

$$A_i\epsilon_0\epsilon_i + B_i\epsilon_i + C_i\epsilon_0 + D_i + p \cdot \kappa_i = 0 \tag{3}$$

From the way we constructed this system of n polynomial relations, we know that it has an integer solution $\epsilon_i = e_i, \kappa_i = k_i$ in which all the e_i's are bounded

below 2^{m-k}. Let $\boldsymbol{v}$ be a $(3n+2)$ integer vector containing the values of all the terms in our system of equations, according to this solution. Namely, we set

$$\boldsymbol{v} \stackrel{\text{def}}{=} \langle 1,\ e_0,\ \ldots,\ e_n,\ e_0e_1,\ \ldots,\ e_0e_n,\ k_1,\ \ldots,\ k_n \rangle$$

It follows that for this integer vector $\boldsymbol{v}$ we get:

$$\boldsymbol{v} \cdot M = \left\langle 1,\ \frac{e_0}{2^{m-k}},\ \ldots,\ \frac{e_n}{2^{m-k}},\ \frac{e_0e_1}{2^{2(m-k)}},\ \ldots,\ \frac{e_0e_n}{2^{2(m-k)}},\ 0,\ \ldots,\ 0 \right\rangle$$

Thus, the lattice point $\boldsymbol{v} \cdot M$ has only $2n+2$ non-zero entries, and each of these is less than 1, so its Euclidean norm is less than $\sqrt{2n+2}$.

On the other hand, it is easy to see that the determinant of the lattice $L(M)$ equals $p^n/2^{(m-k)(3n+1)}$, so making use of the Gaussian heuristic[1] for short lattices vectors, we expect that our vector is the shortest point in $L(M)$ as long as

$$\sqrt{2n+2} \ \ll\ \sqrt{3n+2}\left(2^{(k-m)(3n+1)} \cdot p^n\right)^{1/(3n+2)}. \tag{4}$$

Whenever this condition is met we will assume that an adversary can recover the vector $\boldsymbol{v}$ using lattice reduction methods such as LLL (although we note that in practice, the adversary may not find this that easy unless $\boldsymbol{v} \cdot M$ is the shortest vector by a substantial margin).

Substituting $p \approx 2^m$ into Eq. (4) and ignoring low-order terms, this condition is simplified to $2^k \gg 2^{2m/3}$. Therefore, this method can only be used when the number of bits of $1/(\alpha + x_i)$ that we see is more than $2m/3$ (alternatively, when the number of bits that we are missing is less than $m/3$). This gives an algorithm for Basic-MIHNP when $\delta \geq 2/3$.

The dimension of the lattice. We remark that the same bounds (but no better) could also be achieved from a lattice of smaller dimension that utilizes the fact that $\boldsymbol{v} \cdot M$ has n trailing zeros. However in this analysis we are only interested in showing that there exist bounds on m even for lattices of arbitrary dimension (i.e. we effectively allow the adversary the power to reduce lattices of arbitrary dimension). This means we can ignore efficiency issues regarding the dimension of the lattice, and opt for the easier way to describe and extend the lattices (as above). This assumption is particularly important to note in the subsequent section where the dimension grows exponentially in n.

3.2 Making Use of Multiples

To improve upon the bound of $m/3$, we apply a technique due to Coppersmith [6] to make better use of the relations in Eq. (2). Namely, instead of using only these relations in our lattice, we can use also relations that are derived by taking

[1] In fact, it is possible to prove rigorously, that when the x_i's are chosen at random, the vector $\boldsymbol{v} \cdot M$ is (with high probability) the shortest vector in $L(M)$. This proof will appear in the full version of this paper.

products of them. For example, since we have $f_1(\epsilon_0, \epsilon_1) = 0$ and $f_2(\epsilon_0, \epsilon_2) = 0$, we also know that $\epsilon_2 f_1(\epsilon_0, \epsilon_1) = 0$, and also $f_1(\epsilon_0, \epsilon_1) \cdot f_2(\epsilon_0, \epsilon_2) = 0$. Moreover, since the original relations hold modulo p, then the last relation holds also modulo p^2.

Of course, these additional relations introduce new terms that were not present in the original one. (For example, the relation $f_1 f_2 = 0$ from above has a term $\epsilon_0\epsilon_1\epsilon_2$, which we did not have in the original system.) Nonetheless, we hope that weighing the additional relations against the additional terms, we would be able to get a better result. Hence, our goal here is to add as many relations as possible, while keeping the number of additional terms as small as possible.

Once we decide on a set of relations to use, we construct the lattice in exactly the same way as above. Namely, if we have r relations and t terms, we construct a $(r+t) \times (r+t)$ matrix M with the same structure as above. That is, the top left $t \times t$ sub-matrix E is diagonal with entries that correspond to the (bounds on the) different terms, to its right we put a $t \times r$ matrix R that corresponds to our relations, and at the bottom left we put a diagonal matrix P that would take care of the modular reductions. One difference is that now, if the i'th relation holds modulo p^i, then the corresponding diagonal entry of P will be p^i (rather than just p).

Constructing a lattice. The key aspect of this approach is to choose which relations to put in the lattice, and to analyze the parameters achieved by this lattice. Below we think of the process of adding relations to the lattice as happening in phases. In phase d, we add to the lattice relations that are obtained by multiplying up to d of the original relations. These new relations look like $f_{i_1} \cdots f_{i_d} = 0 \bmod p^d$ for some $0 < i_1, \ldots, i_d \leq n$.

We note that once we have in the lattice some relations (and all their terms), we might as well add other relations that use only terms that already appear in the lattice. For example, if we have the relation $f_1 f_2 = 0$ in the lattice, we might as well also add the relation $\epsilon_1 f_2 = 0$, since every term that appears in $\epsilon_1 f_2$ must already appear in $f_1 f_2$ (because f_1 includes the term ϵ_1). Therefore, once we have $f_1 f_2$ and all its terms, we can add $\epsilon_1 f_2$ "for free". The only exception is that we have to make sure that the relations in the lattice are linearly independent. For example, once we have in the lattice the relations $f_2, \epsilon_0 f_2, \epsilon_1 f_2$ and $f_1 f_2$, we cannot add also the relation $\epsilon_0\epsilon_1 f_2$, as it is linearly dependent on the other relation, by the equality $f_1 f_2 = A_1\epsilon_0\epsilon_1 f_2 + B_1\epsilon_1 f_2 + C_1\epsilon_0 f_2 + D_1 f_2$.

Notations and conventions. In the analysis below we talk about the "weight" of relations or terms. The weight of a relation is the number of original relations that are multiplied. For example, the relation $\epsilon_1 f_2 = 0$ has weight 1, and $f_1 f_2 = 0$ has weight 2. We note that if a relation has weight i, then this relation holds modulo p^i (but not necessarily modulo p^{i+1}). The weight of a term is just its degree. For example, the weight of $\epsilon_0^2\epsilon_1$ is 3. With this notation, the determinant of the lattice is proportional to the total weight of all the relations, and inversely proportional to the total weight of all the terms that are used in these relations.

More precisely, when p is an m-bit prime, and the bound on the ϵ_i's is 2^{m-k}, (i.e., the number of bits of $1/(\alpha+x)$ that we see is k), the determinant of the lattice is roughly $2^{m\cdot\text{weight(relations)}-(m-k)\cdot\text{weight(terms)}}$.

Recall that our goal is to maximize the determinant (since this would imply that the lattice is unlikely to have short vectors, other than the one corresponding to our solution). Below we show, however, that we always have weight(relations) $\leq$ 2weight(terms)/3. Therefore, to get $det(L) > 1$, we must have $m - k < 2m/3$.

The relations. In this analysis we assume that the number n of the original relations can be made as large as we want. Since we aim to show that the approach is bound to fail beyond $2m/3$, we can make this assumption without loss of generality (as adding relations can only help the algorithm). When we analyze phase d, we assume that $n \gg d$, so that we get a good approximation of the sum $\sum_{i=1}^{d}\binom{n}{i}$ by taking just the last term, $\binom{n}{d}$.

The relations that we add in phase d are all the $\binom{n}{d}$ relations of weight d, that are obtained by multiplying d distinct relations (from the original n), and then adding all the relations that are now "for free". We again note that since $n \gg d$, then a vast majority of the relations in phase d are of this form.

Analysis. We start by analyzing the weight of the terms. Since each f_i has all the possible terms for a multi-linear function in ϵ_0, ϵ_i, it follows that a product of d distinct f_i's have all the possible terms with degree at most d in ϵ_0, and at most 1 in all the other ϵ_i's.

We group the terms according to the number of ϵ_i's *other than* ϵ_0 in them. Clearly, we have exactly $(d+1)\binom{n}{j}$ terms with exactly j ϵ_i's other than ϵ_0. (We have $\binom{n}{j}$ ways to choose the ϵ_i's, and then ϵ_0 can have any degree between 0 and d.) The weight of these terms ranges from j (if the degree of ϵ_0 is 0) to $j+d$ (if the degree of ϵ_0 is d). Therefore, the total weight of all the terms is

$$\begin{aligned}\text{weight(terms)} &= \sum_{j=0}^{d}\left(\binom{n}{j}\cdot(j+(j+1)+\cdots+(j+d))\right)\\ &= \sum_{j=0}^{d}\left(\binom{n}{j}\cdot(d+1)(j+\frac{d}{2})\right)\end{aligned}$$

Recall now that we assume that n is large enough with respect to d, so that $\sum_{j=0}^{d}\binom{n}{j} = \binom{n}{d}(1+o(1))$. This implies that also

$$\text{weight(terms)} = \binom{n}{d}(d+1)(3d/2)(1+o(1))$$

By the same argument, the number of terms is $(d+1)\binom{n}{d}(1+o(1))$.

We now proceed to analyze the weight of the relations. First, observe that we cannot have more relations than terms in the lattice, since otherwise we

get linear dependencies. Thus, there are at most $(d+1)\binom{n}{d}(1+o(1))$ relations. Moreover, the weight of each of these relations cannot be more than d, since in phase d we only multiply up to d of the f_i's. Therefore, the total weight of all the relations is bounded by

$$\text{weight(relations)} \leq d \cdot (d+1)\binom{n}{d}(1+o(1))$$

In fact, it is possible to show that this bound is tight, and the total weight of the relations that we get is at least $d^2\binom{n}{d}$. We conclude that in our lattice we must have

$$\frac{\text{weight(relations)}}{\text{weight(terms)}} \;\leq\; \frac{d \cdot (d+1)\binom{n}{d}(1+o(1))}{\binom{n}{d}(d+1)(3d/2)(1+o(1))} \;\leq\; \frac{2}{3} + o(1)$$

(We remark that a more careful analysis can even show a bound of $2/3 - o(1)$.)

3.3 Conclusions from the Analysis of MIHNP

We showed that the Basic-MIHNP problem can be efficiently solved whenever we are given more than $1/3$ of the bits of $(\alpha+x_i)^{-1} \bmod p$. The analysis does not extend beyond $1/3$. (Moreover, near $1/3$ the dimension of the lattice makes it completely infeasible to reduce.) For this reason, we conjecture that this problem is hard when we are given less than $1/3$ of the bits even if a large number of random samples x_i are given. That is, we conjecture that the δ-MIHNP assumption holds whenever $\delta < 1/3$.

3.4 Other Variants of MIHNP

The tools that we devised to analyze the MIHNP can be used also to analyze similar problems. For example, in Section 4 we will be interested in a problem where we are given pairs $(x_i, \beta/(\alpha + x_i) \bmod p),\ i = 1 \ldots n$, and we need to recover both α and β. The corresponding relations that we get are

$$(\alpha + x_i)(b_i + \epsilon_i) = \beta \pmod{p} \qquad\qquad i = 0, \ldots, n,$$

Again, we have a problem with the terms $\alpha\epsilon_i$, but when we eliminate α as before, we would get terms $\beta\epsilon_i$. Hence, to be able to set up a lattice we need to eliminate both α and β. More generally we may consider relations of the form

$$R_i : \ (x_{i0} + y_{i0}\epsilon_i) + \sum_{j=1}^{r}(x_{ij} + y_{ij}\epsilon_i)\alpha_j = 0$$

where the x_{ij}'s and y_{ij}'s are random and known, the α_i's are unknown and unbounded, but common to all these relations, and each ϵ_i is an unknown unique to relation i, but for which we have some bound. As before, the terms $\epsilon_i\alpha_j$ cannot be handled by standard lattice reduction techniques, so we need to first

eliminate the α_j's. We now show that if we need to eliminate r such "unbounded variables", then the lattice-reduction techniques from above can only be used when $k/m > r/(r+2)$ (i.e., the number of hidden bits is less than $m \cdot \frac{2}{r+2}$. Note that for MIHNP we have $r = 1$, and indeed we got the bound $k/m > 1/3$. For the case above of two variables, we get $k/m > 1/2$. Hence, this problem is harder than MIHNP in the sense that it can only be solved when $\delta = k/m > 1/2$.

Assume that we are given given $n+r$ relations. We can set a linear system in the r unknowns $\alpha_1 \ldots \alpha_r$ and r relations $R_{n+1}, \ldots, R_{n+r}$, solve for the unknowns, and then substitute the solution in all the other n relations $R_1 \ldots R_n$ (each time multiplying by the common denominator, to get a polynomial relation rather than a rational one).[2]

Using Cramer's rule for the solution of a linear system, it is easy to verify that the terms that we substitute for the α_j's are multi-linear in $\epsilon_{n+1} \ldots \epsilon_{n+r}$. Hence, after eliminating the "unbounded variables", we are left with n relations $f_i = 0, i = 1 \ldots n$, where f_i is a multi-linear relation in $\epsilon_i, \epsilon_{n+1} \ldots \epsilon_{n+r}$. These relations are the ones we use to set-up a lattice.

As we did for MIHNP, we set-up a lattice not only using the f_i's themselves, but also using products of them. As before, we use relations that we obtain by multiplying d distinct f_i's (for some parameter d, and under the assumption that $n \gg d$). A product of d such f_i's is a relations

$$p(\epsilon_{i_1}, \epsilon_{i_2}, ..., \epsilon_{i_d}, \epsilon_{n+1}, \ldots, \epsilon_{n+r}) = 0$$

where p is multi-linear in the ϵ_{i_j}'s, and has degree d in $\epsilon_{n+1} \ldots \epsilon_{n+r}$. We want to count the total weight of the terms and relations in this lattice. As we know, if $n \gg d$, then it is sufficient to consider only these terms that include exactly d distinct e_i's, other than $\epsilon_{n+1} \ldots \epsilon_{n+r}$. So there are $\binom{n}{d}$ ways of choosing the ϵ_{i_j}'s, and for each choice we have $(d+1)^r$ possible combinations of the degrees of $\epsilon_{n+1} \ldots \epsilon_{n+r}$. Namely, for a specific choice of $\epsilon_{i_1}, \epsilon_{i_2}, ..., \epsilon_{i_d}$, the terms that we get are exactly all the terms in the expression

$$(\epsilon_{i_1} \cdot \epsilon_{i_2} \ \cdots \ \epsilon_{i_d}) \cdot \left((1 + \epsilon_{n+1} + \cdots + \epsilon_{n+1}^d) \ \cdots \ (1 + \epsilon_{n+r} + \cdots + \epsilon_{n+r}^d)\right)$$

This means that for this choice of ϵ_{i_j}'s, we have $(d+1)^r$ terms, and the weight of these terms vary between d and $d + rd$. The total weight of all these terms is

$$\sum_{k_1=0}^{d} \sum_{k_2=0}^{d} \ldots \sum_{k_r=0}^{d} (d + k_1 + k_2 + \ldots k_r) \ = \ (d+1)^r \cdot (d + rd/2)$$

Therefore, we have $\binom{n}{d}(d+1)^r$ terms, of total weight $\binom{n}{d}(d+1)^r \cdot d(1 + r/2)$. On the other hand, we cannot have more relations than terms, and the weight of a relation cannot be more than d, so the total weight of the relations is

[2] Clearly, this is not the only way to eliminate the unbounded variables. For example, we can solve different sets of relations for these unknowns, depending on the relation to which we want to substitute. However, tracing through the arguments below, the method we use here seems to give the smallest number of terms.

at most $\binom{n}{d}(d+1)^r \cdot d$. (This bound is tight, since it can be shown that for random relations, the total weight is at least $\binom{n}{d}(d+1)^r \cdot (d-r)$.) Recall that the determinant of our lattice is roughly $2^{m\cdot\text{weight(relations)}-(m-k)\cdot\text{weight(terms)}}$. To get the determinant above 1, we therefore must have

$$m \cdot \binom{n}{d}(d+1)^r d > (m-k) \cdot \binom{n}{d}(d+1)^r d(1+r/2)$$

which means that $m > (m-k)(1+r/2)$, or $k/m > r/(r+2)$.

4 Cryptographic Applications

The apparent intractability of MIHNP, suggests that it may be useful as the basis for cryptographic applications. Indeed, we show below how to use the decision-MIHNP assumption and the computational-MIHNP assumptions, respectively, to get an efficient pseudorandom generator and a MAC.

4.1 Pseudorandom Generator

The decision-MIHNP immediately suggests a construction of a PRNG. The input to this generator would be "the secret" a, and n random points $x_1..x_n \in \mathbb{Z}_p$. The output would be the points $x_1..x_n$ together with (say) 1/4 of the bits of $1/(a+x_i) \bmod p$ for all i. More precisely, we have the following system:

Parameters. The parameters of the system include an m-bit prime p, and two other parameters, n and k, where k specifies how many bits of $1/(a+x)$ we output, and n specifies how many x'es we have in the input of the generator. These parameters are discussed in more details below.

The generator. On parameters p, n and k, the generator input is a sequence $(x_1, .., x_n, a)$ of $n+1$ elements in Z_p. The output is the sequence

$$G(a, x_1, .., x_n) \stackrel{\text{def}}{=} \left(x_1, ..., x_n, \text{MSB}_k(\frac{1}{a+x_1}), ..., \text{MSB}_k(\frac{1}{a+x_n}) \right)$$

The security of this generator follows immediately from the decision-MIHNP assumption. We note that this is a pseudorandom number generator, but not pseudorandom bit generator, since the output distribution is not the uniform one. There are standard techniques for transforming this to a pseudorandom bit generator. Any of a number of standard extractors could be used for this purpose [16,11].

Proposition 1. *Under the D-MIHNP assumption, G is a secure pseudorandom generator.*

One point worth mentioning is the re-keying of the generator from the previous output. It is well known, see [3], that it is secure to do this, if the underlying generator is itself secure. In our case this means that we may fix the $x_1, ..., x_n$ once at the start of the whole procedure, and then use just the $\text{MSB}_k(1/(a+x_i))$ part of the output to re-key a and form the output bits of the PRNG.

Parameters and performance. The parameters m (the size of the prime p) and k (the number of bits to output from each $1/(a+x_i)$) must be chosen such that solving the MIHNP with k output bits modulo a prime of size $|p| = m$ is infeasible. More precisely, if we assume that the threshold for feasible solution is when the adversary sees $\geq m/3$ of the bits of $1/(a+x_i)$, and we want security level of 2^r, we need to make sure that our generator outputs at most $m/3 - r$ of the bits. This means that we have a tradeoff between the number of the bits that we output (which is related to the expansion of the generator) and the size of the prime that we work with.

A reasonable setting is to set $r = k$ (i.e., output as many bits of $1/(a+x_i)$ as our security parameter). With this setting, we should choose m so that $k \leq m/3 - k$, namely $m \geq 6k$. (Another constraint is that to get security level 2^r, we must hide at least $2r$ bits of $1/(a+x_i)$, to avoid birthday-type attacks. In the current setting, however, this constraint is subsumed by the previous one.) An invocation of the generator G stretches a random input of length $(n+1)m$ bits, into a pseudorandom output of length $n(m+k)$ bits. Hence, each invocation generates $nk - m$ pseudorandom bits.

For a numerical example, assume that we want to get security level of 2^{80}. We then set $m = 6 \cdot 80 = 480$ and $k = 80$ (i.e., we work with a 480-bit prime, and output 80 of the bits of $1/(a+x_i)$). With these parameters, each invocation of G generates $80n - 480$ pseudorandom bits (so we must choose $n > 6$ to get any expansion). In our example below we use $n = 10$.

A naive implementation of this generator would require n modular inversions to compute $\text{MSB}_k(\frac{1}{a+x_i}), i = 1...n$. Therefore, the cost of this implementation is roughly k bits per inversion (for a sufficiently large n). Keeping with the numerical example above, choosing, for example, $n = 10$, the size of the seed (which is the amount of state we keep) is 4800 bits (= 600 bytes), and we get $nk - m = 320$ pseudorandom bits at the cost of 10 inversions, or 32 bits per modular inversion. Keeping a larger state results in more bits per inversion. For example, setting $n = 20$, we have 9600 bits (= 1200 bytes) of state, and we get 1120 bits at the cost of 20 inversions, which is 56 bits per inversion.

Even this naive implementation is already quite fast. With a careful implementation, the cost of modular inversion can be as small as only a few multiplications [1]. Moreover, since we work in a relatively small field, the operations can be quite fast. Finally, we note that the modular inversions are independent of each other, so it is trivial to parallelize this computation.

Speedup via batching. One way to speed up the computation, is to trade modular inversions for multiplications by using batching. The idea, first discovered by Peter Montgomery, is as follows: To compute $1/(a+x_i), i = 1...n$, we first compute the product $\pi = \prod_i (a+x_i)$, then invert only this product to get $\pi^{-1} \bmod p$, and finally compute $1/(a+x_j) = \pi^{-1} \cdot \prod_{i \neq j}(a+x_i)$. It is not hard to see that one can compute all the values $1/(a+x_i)$ using only $3(n-1)$ multiplications and one modular inversion. (For this, one needs to keep in memory up to n intermediate values during the computation.) If inversion is more expensive than three multiplications (as is the case for all the multi-precision software libraries

that we know), then this implementation will be more efficient than the naïve one.

Back to our numeric example, with $n = 10$ we get 320 bits for one inversion and 28 multiplications, which is about 11 bits per multiplication. With $n = 20$ we get 1120 bits for one inversion and 58 multiplications, which is roughly 19 bits per multiplication. Hence, our generator is more efficient than other algebraic generators, e.g. the pseudorandom generator due to Gennaro [9] which is based on the problem of discrete-log with small exponent. The generator of [9] generates approximately one pseudorandom bit per multiplication. Furthermore, Gennaro's generator uses a much larger prime field. Other algebraic generators, such as the Blum-Blum-Shub generator [2], generate a small number of pseudorandom bits per multiplication modulo a much larger modulus than the one we use. The exact comparison of our generator to BBS depends on the number of bits per round output by the BBS generator.

Even faster variants. We can increase the speed even further by slightly modifying the generator itself. Below we describe two such modifications.

Re-defining the output. To speed the batching implementation, we change the output of the generator, so that it would be easier to compute this output from the intermediate value π^{-1} that we get during the computation. Specifically, we set

$$G'(a, x_1, .., x_n) \stackrel{\text{def}}{=} \left(x_1, ..., x_n, \text{MSB}_k(\frac{1}{\pi_1}), ..., \text{MSB}_k(\frac{1}{\pi_n}) \right)$$

where the π_j's are defined by $\pi_j = \prod_{i \neq j}(a + x_i)$.

We stress that the security of G' does not seem to be equivalent to our original D-MIHNP problem. Rather, this generator defines yet another variant of D-MIHNP. Still, the analysis from Section 3 applies in exactly the same manner to this variant too.

Implementing G' using the batching technique takes only $2m - 1$ multiplications and one inversion (and can also be parallelized easier than with G). Hence, in our numerical example we get 10 bits per multiplication for $n = 8$, or 28 bits per multiplication for $n = 20$.

Using a harder MIHNP problem. Another possibility is to use harder variants of MIHNP. For example, instead of using $f_\alpha(x) = 1/(\alpha + x)$ as our underlying function, we can use $f_{\alpha,\beta}(x) = \beta/(\alpha + x)$ where both α and β are secret.

(We mention in passing that just like the original MIHNP, this variant too has limited random self-reducibility (in α and β). This is because we can set $y_i = (x_i + s) \cdot r^{-1}$ and then we have $\frac{\beta}{\alpha+y} = \frac{\beta r}{(\alpha r+s)+x}$. For any fixed α, β (with $\beta \neq 0$), if we choose r, s uniformly at random (with $r \neq 0$) then $\alpha r + s, \beta r$ are uniformly random and independent.)

From the analysis in Section 3.4, it follows that this problem is infeasible to solve when the number of "missing bits" is more than $m/2$ (as opposed to $2m/3$ for the original MIHNP). This means that we may be able to output as many

as $m/2 - r$ bits for security level of 2^r. Assuming that we still set $r = k$, this argument suggests that we must set m (the size of our prime) so that $k \leq m/2-k$, or $m \geq 4k$.

Each invocation of the generator now stretches $m(n + 2)$ random bits into $n(m + k)$ pseudorandom bits, so we get $nk - 2m$ pseudorandom bits per invocation. For a numerical example, to get security level of 2^{80}, we choose $m = 320$, and $k = 80$ (i.e., work with a 320-bit prime and output 80 bits of $\beta/(\alpha + x_i)$). Working with $n = 10$, we have 3840 bits of state and 160 bits per invocation (same as for the $n = 8$ example from above). However, this generator does roughly 25% more operations, but in a smaller field ($|p| = 320$ instead of $|p| = 480$), so we expect it to be nearly twice as fast. Similarly, using $n = 22$ we get 1120 bits per application with state of 7040 bits, which is the same number of bits per invocation, and somewhat smaller state than the $n = 20$ example above. Again, we do 10% more operation over a smaller field, so we expect the overall running time to be roughly twice as fast.

4.2 Message Authentication Code

The computational-MIHNP directly implies an efficient "weak MAC", secure under known (random) message attacks. The parameters p and k are chosen just as for the generator, and the secret MAC key is an element $\alpha \in Z_p$. To authenticate a message χ, one adds the authentication tag

$$\mathrm{MAC}_\alpha(\chi) = \mathrm{MSB}_k(\frac{1}{\alpha + x})$$

Proposition 2. *Suppose the δ-Computational MIHNP assumption holds. Then when $k < \delta(\log_2 p)$ the above MAC is secure under known (random) message attacks.*

The proof is immediate. The cost per MAC computation is thus just one modular inversion. Moreover, if we need to compute MAC for many messages $x_1...x_n$, we can use the same batching trick from the previous section to speed up this computation.

The "weak MAC" above can be converted to a MAC secure against chosen message attack, using standard techniques. For example, one could apply the MAC to a random string r and then use a one-time signature based on r to sign the message x. However, these generic conversion techniques (from security against a known message attack to security against a chosen message attack) make the MAC much less efficient. We do not know whether the MAC from above is by itself secure against chosen message attack. This would require a version of the computational MIHNP assumption, where the x_i's can be chosen by the attacker. Currently we cannot tell whether this chosen message MIHNP problem is intractable.

5 Conclusions and Open Problems

In this paper we proposed the MIHNP, and variants thereof, as new and potentially hard mathematical problems. We presented a few efficient cryptographic constructions based on these problems. To justify the hardness of these MIHNP problems we used the most up-to-date lattice analysis techniques to solve MIHNP and even allowed the attacker the power to reduce infeasibly large lattices. Our best algorithm works whenever the fraction of given bits is greater than one third of the length of the modulus. However, the lattice based approach does not extend to solve the MIHNP when less than a third of the bits is given. We therefore conjectured the MIHNP is hard in this case. MIHNP is an interesting and efficient building block for cryptographic systems. It clearly deserves further study.

One particularly interesting question to answer is how much easier the MIHNP problem becomes if the x_i are not randomly chosen, but adversarially chosen. If the Computational-MIHNP remains hard when the x_i's are chosen adversarially then we obtain an efficient MAC from MIHNP. Also, it is very interesting to see whether any non-lattice approaches shed any light on the hardness of these MIHNP problems.

Lastly we mention that the analysis we have used for MIHNP, can be heuristically applied to certain modular polynomials arising from using the Diffie–Hellman protocol with elliptic curves (ECDH). As was recently done in [5], we may apply our results to proving statements on the bit security of ECDH. Specifically, to prove the bit security of ECDH on a specific curve E, it is sufficient to solve the following hidden-number problem (called ECHNP): We are given

$$\left\langle (x_i, y_i),\ \mathrm{MSB}_k\left(\left(\frac{\psi - y_i}{\chi - x_i}\right)^2 - x_i - \chi\right)\right\rangle$$

for many *random* points $(x_i, y_i) \in E$, and we need to find the hidden point $(\chi, \psi) \in E$. The (heuristic) analysis from Section 3.4 can be applied to this problem too, and it suggests that ECHNP can be solved for $\delta > 3/5$. This would mean that given an algorithm that computes the top $3/5$ fraction of bits in (the x-coordinate of) the ECDH secret, one can devise an algorithm to compute all the bits. However, since the analysis in Section 3.4 is only a heuristic, one does not immediately get a proof of bit-security for ECDH.

We leave further details to a subsequent paper, but mention that while we were able to convert the heuristic analysis into a formal proof in some cases, the result that we get is very weak: We can only prove that for some small constant $\epsilon \approx 0.02$, computing a $(1-\epsilon)$ fraction of the bits in the ECDH secret is as hard as computing them all. This is related to a recent result of Boneh and Shparlinksi [5] which shows that if ECDH is hard on some curve E then there is no single efficient algorithm that predicts one bit of the ECDH secret for many curves isomorphic to E. Our result applies to blocks of bits (rather than a single bit), but is stronger than [5] in the sense that it applies to a specific curve rather

than a family of curves. We show that if ECDH is hard on a specific curve then the top $(1 - \epsilon)$ fraction of the bits of the ECDH secret on that curve cannot be efficiently computed.

References

1. E. Bach, J. Shallit, "Algorithmic number theory, Volume I: efficient algorithms", MIT press, 1996.
2. L. Blum, M. Blum, M. Shub, "A simple unpredictable pseudo-random number generator", SIAM J. Comput. 15, 2 (1986) 364–383.
3. M. Blum and S. Micali. *How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits.* SIAM J.Computing, 13(4):850–864, November 1984.
4. D. Boneh, Venkatesan R., "Hardness of Computing the Most Significant Bits of Secret Keys in Diffie-Hellman and Related Schemes", Proc. of Crypto, 1996, pp. 129–142, 1996.
5. D. Boneh, I. Shparlinksi, "On the unpredictability of bits of the elliptic curve Diffie–Hellman scheme", In *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pp. 201–212. Springer-Verlag, 2001.
6. D.Coppersmith, "Small solutions to polynomial equations, and low exponent RSA vulnerabilities", J. of Cryptology, Vol. 10, pp. 233–260, 1997.
7. R. Cramer and V. Shoup, "A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack", in proc. Crypto '98, pp. 13–25, 1998.
8. R. Cramer and V. Shoup, "Signature schemes based on the Strong RSA Assumption", Proc. 6th ACM Conf. on Computer and Communications Security, 1999.
9. R. Gennaro. An improved pseudo-random generator based on discrete log. In *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pp. 469–481. Springer-Verlag, 2000.
10. R. Gennaro, S. Halevi, T. Rabin, "Secure hash-and-sign signature without random oracles", Proc. Eurocrypt '99, pp. 123–139, 1999.
11. R. Impagliazzo, D. Zuckerman, "How to Recycle Random Bits", FOCS, 1989.
12. N. Howgrave-Graham. Finding small roots of univariate modular equations revisited. In proceedings *Cryptography and Coding*, Lecture Notes in Computer Science, vol. 1355, Springer-Verlag, pp. 131–142, 1997.
13. N. Howgrave-Graham, N. Smart. Lattice attacks on digital signature schemes. manuscript.
14. A. Lenstra, H. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, vol. 261, pp. 515–534, 1982.
15. M. Naor, O. Reingold, "Number theoretic constructions of efficient pseudo random functions", Proc. FOCS '97. pp. 458–467.
16. A. Ta-Shma, D. Zuckerman, and S. Safra, "Extractors from Reed-Muller Codes", FOCS, 2001.