Public-Key Cryptosystems from Lattice Reduction Problems

Oded Goldreich* Shafi Goldwasser† Shai Halevi‡
MIT - Laboratory for Computer Science

November 12, 1996

Abstract

We present a new proposal for a trapdoor one-way function, from which we derive public-key encryption and digital signatures. The security of the new construction is based on the conjectured computational difficulty of lattice-reduction problems, providing a possible alternative to existing public-key encryption algorithms and digital signatures such as RSA and DSS.

Keywords: Public-Key Cryptosystems, Lattice Reduction Problems, Collision Free Hashing.

*oded@theory.lcs.mit.edu. On sabatical leave from the Weizmann Institute of Science, Israel.
†shafi@theory.lcs.mit.edu.
‡Contact author at MIT - LCS, NE43-342, 545 Tech. Square, Cambridge, MA 02139, USA. E-mail: shaih@theory.lcs.mit.edu

1 Introduction

The need for public-key encryption and digital signatures is spreading rapidly today as more people use computer networks to exchange confidential documents, buy products and access sensitive data. In fact, several of these tasks are impossible to achieve without the availability of good (secure and efficient) public-key cryptography.

In light of the importance of public key cryptography, it is surprising that there are relatively few proposals of public key cryptosystems which have received any attention. Moreover, the source of security of these proposals almost always relies on the (apparent) computational intractability of problems in finite integer rings, specifically integer factorization and discrete logarithm computations. In this paper we propose a new public key encryption algorithm and digital signature scheme whose security relies on the computational difficulty of lattice reduction problems, in particular the problem of finding closest vectors in a lattice to a given point (CVP). For comparison with existing schemes, we first quickly review some of the most famous public-key encryption and digital signatures proposals, with emphasis on the computational problems their security is based on.

1.1 Previous proposals

The security of the RSA cryptosystem [RSA], is related to the difficulty of **integer factorization** in the sense that discovering the secret key is as hard as factoring integers, although the actual cryptanalysis problem is potentially easier than factoring integers. Other methods, whose security relies on the difficulty of factoring integers, include Rabin's digital signature method [Ra79] (and its variants – e.g., [Wi84]), the semantically-secure public-key encryption of [GM82, BG84], and the existentially unforgeable signature schemes of [GMR85].

The security of the Diffie-Hellman public-key encryption scheme[1] is related to the problem of **computing discrete logarithms** (DLP) in finite fields in the sense that finding the secret key is as hard as computing discrete logarithms. Again, the actual cryptanalysis problem is potentially easier than discrete log computation. The digital signature method of El-Gamal [El85] (and its DSS modification [DSS]) is also no harder to break than it is to solve discrete logarithms in finite fields. A similar paradigm to the above discrete log based schemes, can be carried out over elliptic curves. In that case, the underlying computational problem is the **Elliptic Logarithm** problem, to compute logarithms in the additive group of points defined by elliptic-curves.

The McEliece public-key encryption scheme [Mc79] is substantially different from the above proposals, in that its security is based on a problem from **algebraic coding theory**. The security of this scheme is based on the conjecture that decoding with a "random looking" linear code is as hard as decoding with a truly random linear code, and on the widely believed intractability of decoding with random linear codes. In terms of efficiency, encryption and decryption amount to a matrix-by-vector multiplication which takes time quadratic in the natural security parameter (i.e., the dimension of the matrix). This compares favorably to the cubic time requires in RSA and the other number theoretic proposals above, yet the size of the public key is larger than in the case of RSA (i.e, quadratic rather than linear). The best known cryptanalytic attack against the McEliece system takes time exponential in the dimension of the code, yet the security of the McEliece system has not been studied as extensively as the RSA system. No digital signature scheme based on algebraic coding theory has been proposed to accompany the public-key encryption scheme.

[1] A straightforward modification of their earlier key-exchange protocol [DH76].

In addition, there are general constructions of (semantically-secure [GM82]) public-key encryption schemes based on any trapdoor function [Ya82]. Interestingly, digital signature schemes which are existentially-unforgeable [GMR85], can be constructed based on any one-way function [NY89, Ro90], without need of trapdoor. Thus one may benefit from the slightly more extended variety of candidate one-way functions which, in addition to the above, include a candidate based on the conjectured intractability of decoding random linear codes [GKL] and Ajtai's recent candidate [Aj96] which is based on the worst-case complexity of approximating the shortest vector in a lattice. Unfortunately, these general constructions for digital signatures (i.e., of [NY89, Ro90]) tend to be inefficient.

1.2 The new proposal

In this paper we propose a new trapdoor one-way function relying on the computational difficulty of lattice reduction problems, in particular the problem of finding closest vectors in a lattice to a given point (CVP).

Starting with this trapdoor function, we derive a public-key encryption and digital signature methods, which are asymptotically more efficient than RSA and its variants, in that the computation time for encryption, decryption, signing, and verifying are all quadratic in the natural security parameter. The size of the public key, however, is longer than for the RSA system. Specifically, for security parameter k, the length of the RSA public-key is k and cost of computation time is $O(k^3)$, whereas for the new scheme the public key is of size $O(k^2)$ and the computation time is $O(k^2)$. Thus, our complexities are as in McEliece encryption scheme [Mc79]. We feel that it is high time to reconsider the belief that shorter (private and public) keys are preferable to faster encryption and decryption time (or signing and verification for signatures). In particular, space and communication costs (associated with keys) in Internet applications seem to be less restricted than envisioned for public-key cryptography applications 20 years ago.

Our trapdoor function. The idea underling our construction is that, given *any* basis for a lattice, it is easy to generate a vector which is close to a lattice point (i.e., by taking a lattice point and adding to it a small error vector). However it seems hard to return from this "close-to-lattice" vector to the original lattice point (given an arbitrary lattice basis). Thus, the operation of adding a small error vector to a lattice point can be thought of as a one-way computation.

In order to introduce a trapdoor mechanism into this one-way computation, we use the fact that different bases of the same lattice seems to yield a difference in the ability to find close lattice points to arbitrary vectors in $\mathcal{R}^n$. Therefore the trapdoor information may be a basis of a lattice which allows very good approximation of the closest lattice point problem. Thus, we use two different bases of the same lattice. One basis is chosen to allows computing the function but not inverting it, while the other basis is chosen to allow computing the inverse function by permitting good approximation to the closet lattice vector problem (CVP). For the sake of the introduction, we simply call such a basis a *reduced basis*. In Section 2, we define a reduced basis to be one with a small dual-orthogonality defect (where 'small' is a parameter). Below we give an informal description of our trapdoor one-way function which uses the above ideas.

The parameters of the system includes the security parameter n (which is the dimension of the lattices that we work with) and a "threshold" parameter σ which determines the size of the error-vectors which we add to the lattice points (say, in L_∞ norm).

A particular function and its trapdoor information are specified by a pair of bases of the same (full rank) lattice in $\mathcal{R}^n$: A "non-reduced" basis B which is used to compute the function and a reduced basis R which serves as the trapdoor information and is used for inversion. The "reduced" basis is selected "uniformly" and the "non-reduced" basis is derived from it using a randomized uni-modular transformation.

The input to the function is a lattice point (which is specified by an integral linear combination of the columns of B) and an error vector whose size is bounded by σ. The value of the function on this input is just the vector sum of the two points. To invert the function, we use a reduced basis R in one of Babai's nearest-vector approximation algorithms [Ba86] to find a lattice point which is at most σ away from the given vector.

The cryptanalytic problem underlying our scheme is to approximate the closest vector problem (CVP) in a lattice, given a "non-reduced" basis for that lattice. A related problem is the problem of reducing the given public basis (since one obvious attack is to reduce the given basis and then use the result for inverting the function). See Section 2.1 for a description of these computational problems in lattices.

From trapdoor function to encryption scheme. In order to use the above trapdoor function for public-key encryption, we need a way to embed the message in the arguments to this function. There are several ways to do that, and we discuss some of them in Section 4.2. Here we only describe one of them, in which the message is embedded in the lattice point.

The private and public pair of keys of a user are a pair of two bases of the same lattice of dimension n (the security parameter). The public basis will allow encryption whereas the private basis is chosen to allow decryption. To encrypt a message we first map it to a lattice point by taking the integer combinations "specified" by the message of the public basis vectors, and then add to the lattice point a "small error vector" chosen at random. To decrypt, we look for a lattice point which is close to the ciphertext. By using the private basis, which is a reduced basis, the correct decryption is obtained with high probability. We remark that our encryption algorithm is similar in its algorithmic nature to McEliece's scheme [Mc79].

Our signature scheme. Our signature scheme is similar to the encryption scheme. Regard the message as a n-dimensional vector over the reals. Then, a signature of such vector, is a lattice point which is "close" to it (where closeness is defined by a published threshold). The private basis is reduced so that finding "close" points is possible. Verifying correctness amounts to checking that a signature is indeed a lattice point and that the message is close to the signature.

It is important to remark at the outset, that messages which are close to each other will have the same signature. When applying the method in a setting where this property is desirable (e.g., signing analog signals which may change a little in time), this feature is of great benefit. However, when applying the method to a message space where such property is undesirable, we propose to first hash the message and only then sign it. This is good practice also in case that the scheme is subject to a chosen message attack, as otherwise being able to obtain different signatures of two messages which are close to each other when viewed as points in $\mathcal{R}^n$ will imply the ability to compute a small basis for the lattice which in turn will enable the attacker to find close vectors in a lattice and break the scheme. Interestingly, a family of collision-free hash functions can be constructed assuming that Lattice-Reduction is hard on the worst-case (see below).

1.3 Discussion

Our work was inspired by a remarkable result of Ajtai [Aj96] who introduced a function which is provably a one-way function if approximating the shortest non-zero vector (SVP) in a lattice is hard *on the worst case.* Ajtai's work may be viewed as exhibiting a samplable distribution on lattices and proving that approximating the shortest non-zero vector in lattices chosen according to this distribution is as hard as the worst case instance of approximating the shortest non-zero vector in a lattice. Ajtai's construction, however, does not provide a trapdoor function and thus does not provide a way of doing public-key encryption based on lattice problems. Constructing such a trapdoor function is the novelty and focus of our work.

We remark that the construction of [NY89] can be applied to the one-way function of Ajtai and thus yield a signature scheme based on the SVP problem. However, this construction is generic and thus quite inefficient. In contrast, the signature scheme which we suggest based on the new trapdoor function is more efficient, and based on the computational difficulty of the CVP. Alas, the distribution over CVP instances, induced by our construction, is not known to enjoy the "hardness of the worst-case" property of Ajtai's result.

In retrospect, our encryption scheme bears much similarity to McEliece's scheme [Mc79]. His scheme utilizes a pair of matrices over GF(2), which corresponds to two representations of the same linear code. The encryption method is probabilistic: one multiplies the public matrix by the message vector and adds a random noise vector to the resulting codeword. Thus in both McEliece and our encryption scheme, encryption amounts to a matrix-by-vector multiplication and the addition of a suitable random vector to the result. However, the domains in which these operations take place are vastly different and so is the algebra. Another difference is in the way the private-key is generated. In McEliece's scheme the private-key is a random Goppa code and has structure essential for legitimate decoding. In our scheme the private-key can be chosen uniformly and thus is "structure-less" – legitimate decoding merely depends on a property of such random choices. In both schemes the public-key is obtained by a suitable random linear transformation of the private-key; however, in our scheme the choice of this transformation seems richer. In general, we believe that McEliece's suggestion as well as ours deserve further investigation, especially due to the difference in computational complexity required from the legal sender and receiver in these schemes as compared with the factoring/DLP based schemes.

What can we prove about the security of our proposal? Since complexity theory has yet to produce a non-linear lower bounds for even one NP-complete problems, our proposal is essentially based on the failure of past research efforts to come up with efficient algorithms for the relevant lattice approximation tasks (i.e., SVP and CVP). Using the best known algorithms for approximating the closest vector problem we show in Section 6 that a natural attack on our trapdoor function takes exponential time in the dimension of the lattice. In particular, according to our estimates this attack should be intractable in practice for dimension 200.

Drawing an analogy from the past, in proposing the RSA, Rivest et. al. [RSA] relied on the failure of past research to produce efficient factoring algorithms, but did not reduce factoring to the breaking of their proposal. By now, the assumption that RSA itself is hard to invert (rather than factoring in general) can stand on its own, as it has been subjected to much examination and scrutiny. The structure of our proposal (i.e the key generation process) is more complex than in RSA and requires stating a more complex assumption. Essentially, we need to conjecture that for some natural distribution on lattices and bases for these lattices, the CVP is hard. We don't know if a result similar to Ajtai's can be proved for the distribution which we propose over the CVP instances. (Similarity, it is not known whether such a result can be proved for RSA.) We hope that our suggestion will stir up further investigation into the complexity of lattice problems.

1.4 Organization

In Section 2 we review necessary material about lattices and lattice problems. In Section 3 we describe our construction of a trapdoor function and discuss various parameters and attacks. Section 4 describes our encryption scheme and Section 5 describes our signature scheme. In Section 6 we describe our experimental results

2 Lattices and Lattice Reduction Problems

Notations and conventions. In the sequel we use the following conventions: We denote the set of real numbers by $\mathcal{R}$ and the set of integers by $\mathcal{Z}$. We denote real numbers by small Greek letters (e.g., β, ρ, τ etc.) and integers by one of the letters i, j, k, l, m, n. We denote vectors by bold-face lowercase letters (e.g., $\mathbf{b}, \mathbf{c}, \mathbf{r}$ etc.). We use capital letters (e.g., B, C, R, etc.) to denote matrices or sets of vectors.

In this paper we only care about lattices of full rank, so the definitions below only deal with those.

Definition 1: Given a set of n linearly independent vectors in $\mathcal{R}^n$, $B = \{\mathbf{b}_1, \cdots, \mathbf{b}_n\}$, we define the **lattice spanned by** B as the set of all possible linear combinations of the $\mathbf{b}_i$'s with integral coefficients, namely

$$L(B) \stackrel{\text{def}}{=} \{\sum_i k_i b_i \ : \ k_i \in \mathcal{Z} \text{ for all } i\}$$

We call B a **basis** of the lattice $L(B)$. We say that a set of vectors $L \subset \mathcal{R}^n$ is a lattice if there is a basis B such that $L = L(B)$. If the vector $\mathbf{v}$ belongs to the lattice L, then we say that $\mathbf{v}$ is a lattice-vector (or a lattice point).

Below we briefly mention a few well-known facts about lattices. In the sequel we view a basis for a lattice in $\mathcal{R}^n$ as an $n \times n$ non-singular matrix B whose columns are the basis vectors. Viewed this way, the lattice spanned by B is the set $L(B) = \{B\mathbf{v} : \mathbf{v} \text{ is an integral vector}\}$. We note that there are many different bases for any lattice L. In fact, if the set $B = \{\mathbf{b}_1, \cdots, \mathbf{b}_n\}$ spans some lattice then by taking any vector $\mathbf{b}_i \in B$ and adding to it any integral linear combination of the other vectors we obtain a different basis for the same lattice.

All bases have the same determinant. The first important fact about lattices is that all the bases of a given lattice have the same determinant. This fact follows since there is an integer matrix T such that $BT = C$ and another integer matrix T^{-1} such that $CT^{-1} = B$.

The dual lattice Let $B = \mathbf{b}_1, \cdots, \mathbf{b}_n$ be a basis for some lattice in $\mathcal{R}^n$, $L = L(B)$. Recall that we think of B as an $n \times n$ matrix whose columns are the $\mathbf{b}_i$'s. The *dual lattice of* L is the lattice which is spanned by the rows of the matrix B^{-1}. Let us denote the rows of B^{-1} by $\hat{\mathbf{b}}_1, \cdots, \hat{\mathbf{b}}_n$.

Orthogonality Defect The notion of of the *orthogonality defect* of a basis which was introduced by Schnorr in [Sc87] plays a crucial role in the security of our schemes.

Definition 2: Let B be a real non-singular $n \times n$ matrix. The *orthogonality defect* of B is defined as orth-defect$(B) \stackrel{\text{def}}{=} \frac{\prod_i \|\mathbf{b}_i\|}{det(B)}$, where $\|\mathbf{b}_i\|$ is the Euclidean norm of the i'th column in B.

Clearly we have orth-defect$(B) = 1$ if and only if the columns of B are orthogonal to one another, and orth-defect$(B) > 1$ otherwise. When comparing different bases of the same lattice in $\mathcal{R}^n$, we really only care about the product of the $\|\mathbf{b}_i\|$'s, since $det(B)$ is the same for all of them (and serves just as a normalization factor). In Section 3.5 we demonstrate the importance of the orthogonality defect to the security of our schemes. In particular we show that when we use a basis B for a lattice $L = L(B)$ for our trapdoor function, the work load which is associated with a brute-force attacks on the scheme is proportional to the orthogonality defect *of the corresponding basis for the dual lattice*. It would therefore be convenient for us to define the *dual orthogonality defect* for a matrix.

Definition 3: Let B be a real non-singular $n \times n$ matrix. The *dual orthogonality defect* of B is defined as orth-defect$^*(B) \stackrel{\text{def}}{=} \prod_i \|\hat{\mathbf{b}}_i\| / det(B^{-1}) = det(B) \cdot \prod_i \|\hat{\mathbf{b}}_i\|$, where $\|\hat{\mathbf{b}}_i\|$ is the Euclidean norm of the i'th row in B^{-1}.

2.1 Hard problems in lattices

The security of our constructions is related to the (conjectured) intractability of a few computational problems in lattices.

The Shortest non-zero Vector Problem (SVP). This problem underlies the security of Ajtai's construction and our collision-free hashing. In this problem we are given a basis B for a lattice in $\mathcal{R}^n$ and our task is to find the non-zero vector in $L(B)$ whose Euclidean norm is minimum. There are no known polynomial-time algorithms for solving the SVP, and it is also not known whether the SVP is $\mathcal{NP}$-hard (although a version of it, where the distance is measured in L_∞ norm, was shown by van Emde Boas [Bo81] to be $\mathcal{NP}$-hard). There are, however, deterministic polynomial-time approximation algorithms for the SVP. The LLL algorithm (due to Lovász, Lenstra and Lenstra [LLL]) approximates the SVP in $\mathcal{R}^n$ up to a factor of $2^{n/2}$ in the worst case. This was later improved by Schnorr [Sc87] to a factor of $(1+\varepsilon)^n$ for any $\varepsilon > 0$.

No polynomial-time algorithm is known for approximating the SVP in $\mathcal{R}^n$ within a polynomial factor in n. Indeed such an approximation has been conjectured to be infeasible to achieve. Recently, Ajtai [Aj96] described samplable distributions which form also a "hard-core distribution" for the SVP. Namely, any (probabilistic polynomial time) algorithm which can approximate the SVP problem with a polynomial approximation ratio on random instances drawn with Ajtai's distribution, can be transformed into a (probabilistic polynomial time) algorithm which achieves a polynomial approximation ratio on **every** instance of the SVP.

The Closest Vector Problem (CVP). This is the "non-homogeneous" version of the SVP. In this problem we are given a basis B for a lattice in $\mathcal{R}^n$ and another vector $\mathbf{v} \in \mathcal{R}^n$, and our task is to find the vector in $L(B)$ which is closest to $\mathbf{v}$ (in some norm). The CVP was shown by van Emde Boas [Bo81] to be $\mathcal{NP}$-hard. In terms of approximation, it was shown in [Ka] that any algorithm which approximates the SVP to within a factor of ρ can be transformed into an algorithm which approximates the CVP to within a factor of $n^{3/2}\rho^2$. Combined with Schnorr's algorithm, this yields a polynomial-time deterministic algorithm which approximates the CVP in $\mathcal{R}^n$ to within a factor of $(1+\varepsilon)^n$ for any $\varepsilon > 0$.

As we explain in Section 3, an attack against our trapdoor function amounts to finding an exact solution for some instance of CVP.

The Smallest Basis Problem (SBP). In this problem, we are given a basis B for a lattice in $\mathcal{R}^n$ and our goal is to find the "smallest" basis B' for the same lattice. There are many variants of this problem, depending on the exact meaning of "smallest". In the context of this paper, we care about bases with small orthogonality defect. Thus, we consider the version in which we look for the basis B' of $L(B)$ which has smallest orthogonality defect. In out public key constructions, finding the private-key from the public-key requires solving some distribution over SBP instances.

For this problem too there are no known polynomial-time algorithms, and the best polynomial-time approximation algorithms for it are again the LLL and Schnorr's algorithms which achieve an approximation ratio of $2^{O(n^2)}$ in the worst case for SBP instances in $\mathcal{R}^n$.

Worst case vs. average case. The upper-bounds above on the performance of the approximation algorithms are all worst-case bounds. However, for the security of our scheme we are more interested in the performance of these algorithms "on the average". In fact, typically the LLL algorithm and its variants perform much better than the above upper-bounds.

The only known theoretical result about the difficulty of "average case" lattice problems is Ajtai's result which we mentioned above [Aj96]. As we explained in the Introduction, however, we could not directly use Ajtai's result for our scheme. We instead propose a trapdoor function and provide some empirical evidence to its security, by testing the difficulty of the distribution of lattice problems which is defined by our scheme against some known approximation algorithms with various parameters. We describe these tests in Section 6.

3 A Candidate Trapdoor Function

In this section we define our candidate trapdoor function and analyze a few possible attacks against it. We start by reviewing the definition of a collection of trapdoor functions

Definition 4: A *collection of one-way trapdoor functions* consists of four (probabilistic) polynomial-time algorithms, Generate, Sample, Evaluate and Invert

Generate. The randomized algorithm Generate takes as input the security parameter (1^n) and outputs a pair (f, t) where f describes a function and t is a trapdoor information. There is a domain D_f which is associated with every function f.

Sample. The randomized algorithm Sample takes as input a function description f (which is part of the output of Generate) and outputs a point $x \in D_f$. The random choices of this algorithm induce a probability distribution over the domain D_f.

Evaluate. The algorithm Evaluate takes as input a function description f and a point $x \in D_f$ and returns the value $f(x)$.

Invert. The algorithm Invert takes as input a function description f, the corresponding trapdoor information t and a point y in the range of f, and returns a point $x \in D_f$ for which $f(x) = y$.

We require that the Invert algorithm be successful with high probability, where this probability is taken over the random coin-tosses of Generate and Sample, and over the coin-tosses of Invert itself (if it happens to be randomized).

A collection Generate, Sample, Evaluate is one-way if Evaluate is a polynomial-time algorithm, and for any probabilistic polynomial time algorithm A, the probability that A succeeds in inverting f - when it is only given $1^n, f, f(x)$ - is negligible. The probability in this case is taken over the coin tosses of GENERATE, SAMPLE and A itself, and is measured against the security parameter which was the input to both GENERATE and A. Namely, we have

$$\Pr_{f,x}\left[A(1^n, f, f(x)) \in f^{-1}[f(x)]\right] = \text{negligible}(n)$$

where the probability is taken over the choice of f by GENERATE, the choice of x by SAMPLE and the internal coin-tosses of the A. (we say that a real-valued function is *negligible* in n, if as n gets larger this function becomes smaller than any polynomial in $1/n$).

3.1 Our construction

GENERATE On input 1^n, we generate two bases B and R of the same full-rank lattice in $\mathcal{Z}^n$ and a positive real number σ. We generate these bases so that R has a low dual-orthogonality-defect and B has a high dual-orthogonality-defect. We describe the generation process in details in Section 3.3. The bases B, R are represented by $n \times n$ matrices where the basis-vectors are the columns of these matrices. In the sequel we call B the "public basis" and R the "private basis". We view (B, σ) as the description of a function $f_{B,\sigma}$ and R as the trapdoor information. The domain of $f_{B,\sigma}$ consists of pairs of vectors $\mathbf{v}, \mathbf{e} \in \mathcal{R}^n$.

SAMPLE Given (B, σ), we output vectors $\mathbf{v}, \mathbf{e} \in \mathcal{R}^n$ as follows:

The vector $\mathbf{v}$ is chosen at random from a "large enough" cube in $\mathcal{Z}^n$. For example, we can pick each entry in $\mathbf{v}$ uniformly at random from the range $\{-n^2, -n^2+1, \cdots, +n^2\}$. [2]

The vector $\mathbf{e}$ is chosen at random from $\mathcal{R}^n$, so that each entry in it has zero-mean and variance σ^2. For example, we can pick each entry in $\mathbf{e}$ as $\pm\sigma$, each with probability $\frac{1}{2}$. Alternatively, if we want $\mathbf{e}$ to have integral entries we can pick each entry as equal to $\pm\lceil\sigma\rceil$ each with probability $\sigma^2/2\lceil\sigma\rceil^2$ and 0 with probability $1 - \sigma^2/\lceil\sigma\rceil^2$.

EVALUATE Given $B, \sigma, \mathbf{v}, \mathbf{e}$, we compute $\mathbf{c} = f_{B,\sigma}(\mathbf{v}, \mathbf{e}) = B\mathbf{v} + \mathbf{e}$.

INVERT Given R and $\mathbf{c}$, we use Babai's Round-Off algorithm [Ba86] to invert the function. Namely, we represent $\mathbf{c}$ as a linear combination on the columns of R and then round the coefficients in this linear combination to the nearest integers to get a lattice point. The representation of this lattice point as a linear combination on the columns of B is the vector $\mathbf{v}$. Once we have $\mathbf{v}$ we can compute $\mathbf{e}$. More precisely, denote $T \stackrel{\text{def}}{=} B^{-1}R$, so we compute $v \leftarrow T\lceil R^{-1}\mathbf{c}\rfloor$ and $e \leftarrow \mathbf{c} - B\mathbf{v}$.

3.2 The Inversion Algorithm

In this section we show how σ can be chosen so that the inversion algorithm is successful with high probability. Recall that the inversion algorithm succeeds in inverting the function on $\mathbf{c}$ if using the private basis R in Babai's "round off" algorithm results in finding the closest lattice-point to $\mathbf{c}$. Below we suggest two different ways to bound the value of σ, based on the L_1 norm and L_∞ norm of rows in R^{-1}. Both bounds uses the following lemma.

Lemma 3.1: Let R be the private basis used in the inversion of $f_{B,\sigma}(\mathbf{v}, \mathbf{e})$. Then an inversion error occurs if and only if $\lceil R^{-1}\mathbf{e}\rfloor \neq \bar{\mathbf{0}}$.

[2] We do not know if the size of this range has any influence on the security of the construction. The value n^2 is rather arbitrary, and was only chosen to get integers of about 16 bits for the parameters which we work with.

Proof: Let T be the unimodular transformation matrix $T = B^{-1}R$. Then the inversion algorithm is $\mathbf{v} = T\lceil R^{-1}\mathbf{c}\rfloor$ and $\mathbf{e} = \mathbf{c} - B\mathbf{v}$. Obviously, if $\mathbf{v}$ is computed correctly then so is $\mathbf{e}$. Thus, let us examine the conditions under which this algorithm finds the correct vector $\mathbf{v}$. Recall that $\mathbf{c}$ was computed as $\mathbf{c} = B\mathbf{v} + \mathbf{e}$, so

$$\begin{aligned} &T\lceil R^{-1}\mathbf{c}\rfloor \;=\; T\lceil R^{-1}(B\mathbf{v}+\mathbf{e})\rfloor \\ &=\; T\lceil R^{-1}B\mathbf{v} + R^{-1}\mathbf{e}\rfloor \;=\; T\lceil (BT)^{-1}B\mathbf{v} + R^{-1}\mathbf{e}\rfloor \;=\; T\lceil T^{-1}\mathbf{v} + R^{-1}\mathbf{e}\rfloor \end{aligned}$$

But since T is a unimodular matrix (and therefore, so it T^{-1}) and since $\mathbf{v}$ is an integral vector, then $T^{-1}\mathbf{v}$ is also an integral vector. Hence we have $\lceil T^{-1}\mathbf{v} + R^{-1}\mathbf{e}\rfloor = T^{-1}\mathbf{v} + \lceil R^{-1}\mathbf{e}\rfloor$, and therefore

$$\lceil R^{-1}\mathbf{c}\rfloor = T(\;T^{-1}\mathbf{v} + \lceil R^{-1}\mathbf{e}\rfloor\;) = \mathbf{v} + T\lceil R^{-1}\mathbf{e}\rfloor$$

Thus the inversion algorithm succeeds if and only if $\lceil R^{-1}\mathbf{e}\rfloor = \bar{\mathbf{0}}$. □

We proceed to show the bounds on σ. In both the theorems below we assume that each entry in the "error vector" $\mathbf{e}$ is chosen equal to $\pm\sigma$, each with probability a half. We start by asserting that we can choose σ so that we never get any inversion errors.

Theorem 1: Let R be the private basis used in the inversion of $f_{B,\sigma}$, and denote the maximum L_1 norm of the rows in R^{-1} by ρ. Then as long as $\sigma < 1/(2\rho)$, no inversion errors can occur.

Proof: We first introduce a few notations. We denote $\mathbf{d} \stackrel{\text{def}}{=} R^{-1}\mathbf{e}$ and denote the i'th entry in $\mathbf{d}$ and $\mathbf{e}$ by δ_i and ϵ_i respectively. Also, we denote the i'th row in R^{-1} by $\hat{\mathbf{r}}_i$ and the i,j'th element in R^{-1} by ρ_{ij}.

By Lemma 3.1 above, we get an inversion error only when $\lceil R^{-1}\mathbf{e}\rfloor \neq \bar{\mathbf{0}}$, which means that $|\delta_i| > \frac{1}{2}$ for some i. However, since all the entries in $\mathbf{e}$ are equal $\pm\sigma$, we get for every i

$$|\delta_i| = |\hat{\mathbf{r}}_i \circ \mathbf{e}| = |\sum_j \rho_{ij}\epsilon_j| \leq \sigma \cdot \sum_j |\rho_{ij}| \leq \sigma \cdot \rho < \frac{1}{2}$$

□

Although Theorem 1 gives a sufficient condition to get the error-probability down to 0, we may choose to set a higher value for σ in order to get better security. The next theorem asserts a different bound on σ, which guarantee a low error probability.

Theorem 2: Let R be the private basis used in the inversion of $f_{B,\sigma}$, and denote the maximum L_∞ norm of the rows in R^{-1} by $\frac{\gamma}{\sqrt{n}}$. Then the probability of inversion errors is bounded by

$$\Pr[\text{ inversion error using } R\,] \leq 2n \cdot \exp\left(-\frac{1}{8\sigma^2\gamma^2}\right) \qquad (1)$$

Proof: We use the notations $\mathbf{d}, \delta_i, \epsilon_i, \hat{\mathbf{r}}_i$ and ρ_{ij} as in the proof of Theorem 1. We first fix some i and evaluate $\Pr[|\delta_i| \geq \frac{1}{2}]$. Recall that $\delta_i = \hat{\mathbf{r}}_i \circ \mathbf{e} = \sum_j \rho_{ij}\epsilon_j$. Since for all j, $|\rho_{ij}| \leq \gamma/\sqrt{n}$ and $\epsilon_j = \pm\sigma$, each with probability $\frac{1}{2}$, then all the random variables $\rho_{ij}\epsilon_j$ have zero mean and they are all limited to the interval $[-\frac{\sigma\gamma}{\sqrt{n}}, +\frac{\sigma\gamma}{\sqrt{n}}]$. Therefore we can use Hoeffding bound to conclude that

$$\Pr\left[|\delta_i| > \frac{1}{2}\right] = \Pr\left[|\sum_j \rho_{ij}\epsilon_j| > \frac{1}{2}\right] < 2\exp\left(-\frac{1}{8\sigma^2\gamma^2}\right)$$

Using the union bound to bound the probability that any such i exists completes the proof. $\square$

Remark. The last theorem implies that to get the error probability below ε it is sufficient to choose $\sigma \leq \left(\gamma\sqrt{8\ln(2n/\varepsilon)}\right)^{-1}$. In fact, the above bound is overly pessimistic in that it only looks at the largest entry in R^{-1}. A more refined bound can be obtained by considering the few largest entries in each row separately and applying the above argument to the rest of the entries.

Alternatively, we can get an estimate (rather that a bound) of the error probability by using Equation 1 as if all the entries in each row of R^{-1} have the same absolute value. In this case γ is the maximum Euclidean norm of the rows in R^{-1} so we get an estimate of the error-probability in terms of the Euclidean norm of the rows in R^{-1}. This estimate is about the same as the one which we get by viewing each of the δ_i's as a zero-mean Gaussian random variable with variance $(\sigma\|\hat{\mathbf{r}}_i\|)^2$ (where $\|\hat{\mathbf{r}}_i\|$ is the Euclidean norm of the i'th row in R^{-1}).

To get a feeling for the size of the parameters involved, consider the parameters $n = 140$, $\varepsilon = 2^{-30}$. For a certain setting of the parameters which we tested, the Euclidean norm of all the rows in R^{-1} is below $1/30$. Evaluating the expression above for $\gamma = 1/30$ yields

$$\sigma \leq \left(\frac{1}{30}\sqrt{8\ln\left(\frac{280}{2^{-30}}\right)}\right)^{-1} \approx \frac{30}{14.6} \approx 2$$

For the same parameters of R, setting $\varepsilon = 10^{-4}$ yields $\sigma \leq 2.7$

3.3 The Generate Algorithm

In this section we discuss various aspects of the Generate algorithm. We described in Section 3.2 how the value of σ can be computed once we have the private basis R. Now we suggest a few ways to pick R and B. Recall that R, B are two bases for some lattice in $\mathcal{Z}^n$, where R has small dual orthogonality defect and B has a large dual orthogonality defect. Our high-level approach for generating the private and public bases is to choose at random n vectors in $\mathcal{Z}^n$ to get the private basis and then to mix them so as to get the public one. There are two distributions to consider in this process

- A distribution on the lattices in $\mathcal{Z}^n$ which is induced by the choice of the private basis R.
- Once we have the private basis R, there is a distribution which is induced on the bases of $L(R)$ by the process of "mixing" R to get the public basis B.

To guide us through the choices of the various parameters, we relied on experimental results (See Section 6). Below we briefly discuss the various parameters which are involved in this process.

3.3.1 Lattice dimension

The first parameter we need to set is the dimension of the lattice (the value of n). Clearly, the larger n is, we expect that our schemes will be more secure. On the other hand, both the space needed for the key pair and the running-time of function-evaluation and function-inversion grow (at least) as $\Omega(n^2)$.

The lattice-reduction algorithm which we used for our experiments is capable of finding a basis with very small orthogonality defect as long as the lattice dimension is no more than 60-80 (depending on other parameters). Beyond this point, the quality of the bases we get from this lattice reduction algorithm degrades rapidly with the dimensions. In particular, we found that in dimension 100, the bases we obtained had a high dial-orthogonality-defect. At the present time, the best "practical lattice-reduction algorithm" which we are aware of is Schnorr's block-reduction scheme (which was used to attack the Chor-Rivest cryptosystem, see [Sc95]). We speculate that working in dimensions about 150-200 might be good enough with respect to this algorithm.

3.3.2 Distribution of the private basis

After setting the dimension, we need to decide on the distribution according to which we choose the private basis. We considered two possible such distributions.

1. Choosing a "random lattice": We choose a matrix R which is uniformly distributed in $\{-l, \cdots, +l\}^{n\times n}$ for some integer bound l. In our experiments, the value of l had almost no effect on the quality of the bases which we got. Therefore we chose to work with small integers (e.g, between ± 4), since this makes some of the operations more efficient.

2. Choosing a "rectangular lattice": We start from the box kI in $\mathcal{R}^n$ (for some number k), and add "noise" to each of the box vectors. Namely, we first pick a matrix R' which is uniformly distributed in $\{-l, \cdots, +l\}^{n\times n}$, and then compute $R \leftarrow R' + kI$.

   The larger the value of k is, this process generates a basis with smaller dual orthogonality factor, but it may also allow an attacker to obtain a basis with smaller dual orthogonality factor by reducing the public basis.

3.3.3 Generating the public basis

Once we have the private basis R, we should pick the public basis according to some distribution on the bases of the lattice $L(R)$. Since every basis of $L(R)$ is obtained as $B = RT$ for some unimodular transformation matrix T, then picking B when we have R is equivalent to picking a "random unimodular transformation". We tried two ways of generating these "random unimodular transformations". Both methods work by multiplying many "elementary matrices", of different forms.

- One type of elementary matrices which we considered are matrices of the form

$$\begin{pmatrix} 1 & & & \star & & & \\ & \ddots & & \vdots & & & \\ & & 1 & \star & & & \\ & & & 1 & & & \\ & & & \star & 1 & & \\ & & & \vdots & & \ddots & \\ & & & \star & & & 1 \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ \star & \cdots & \star & 1 & \star & \cdots & \star \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}$$

  where the $\star$'s represent any integers and the blanks represent zeros. (The first form corresponds to adding to the i'th columns an integer linear combination of the other columns and the second corresponds to adding an integral multiple of the i'th column to all the other columns.) We typically chose the $\star$ components at random from $\{-1, 0, 1\}$ with a bias towards 0 (specifically, we used $\Pr[1] = \Pr[-1] = 1/7$). This was done so that the size of the numbers in the public basis will not grow too fast.

An important parameter in this process is the number of elementary matrices which we multiply together (which we refer to as the "number of mixing steps"). In our experiments we only used matrices of the first form, and went through the values of i in order so as to make sure that we hit them all. Our experiments indicate that using $2n$ such matrices is sufficient.

- Another possible type of elementary matrices are upper- and lower-triangular matrices with ± 1 on the main diagonal. We did very few experiments with these matrices. In these experiments we chose the non-zero entries in L and U (which are lower- and upper-diagonal respectively) from $\{-1, 0, 1\}$. We found that we need to multiply at least 4 LU pairs to prevent LLL from recovering the original basis.

Comparing the two methods, we found that for the same "level of security", the second method required a basis B with larger entries. Thus we used the first method in the most of our experiments. One way to keep the entries of the public basis small (using either of the distributions above) is to LLL-reduce the mixed basis. This does not affect the security of the trapdoor function (since an attacker can do the same thing). However, when used in the encryption scheme which we suggest in Section 4, there may be some advantage in keeping the entries in B "not too small".

3.4 Bases representation.

To make evaluating and inverting the function more efficient, we chose the following representation for the private and public bases. The public bases is represented by the integer matrix B whose columns are the basis-vectors, so that evaluating $f_{B,\sigma}(\mathbf{v}, \mathbf{e}) = B\mathbf{v} + \mathbf{e}$ can be done in quadratic time. To invert $f_{B,\sigma}$ efficiently, however, we do not store the private basis R itself. Instead, we store the matrix R^{-1} and the unimodular matrix $T = B^{-1}R$. Then, to compute $f^{-1}_{B,\sigma}(\mathbf{c})$ we set $\mathbf{v} = T\lceil R^{-1}\mathbf{c}\rfloor$ and $\mathbf{e} = \mathbf{c} - B\mathbf{v}$, both of which can be done in quadratic time.

Representing B, T is easy since they are integral matrices, but R^{-1} is not an integral matrix, so we need to consider how it should be represented. One possibility, of course, is to keep the exact values of all the entries in R^{-1}. After all, the entries of R^{-1} are all rational, and the number of bits it take to write them down is at most polynomial in the number of bits of R. This approach, however, is rather expensive in terms of running time. Although the entries in R are small (typically, only 2-3 bits) the determinant of R is much larger (about 200 bits if R is a 100×100 matrix) which means that we need to work with very large numbers in order to perform operations on R^{-1}. A different approach is to only keep a few bits of each entry in R^{-1}. This, of course, may introduces errors. If we only keep ℓ bits per entry then we get an error of at most $2^{-\ell}$ in each entry.

Clearly, this has no effect on the security of the system (since it only effects the operations done using the private basis), but it may increase the probability of inversion errors. Since we only perform linear operations on R^{-1}, it is rather straightforward to evaluate the effect of adding small errors to its entries. Denote the "error matrix" by $E = (\epsilon_{ij})$. That is, ϵ_{ij} is the difference between the value which is stored for $(R^{-1})_{ij}$ and the real value of that entry. Then we have $|\epsilon_{ij}| < 2^{-\ell}$ for all i, j. When inverting the function, we apply the same procedure as above, but uses the matrix $R' \stackrel{\text{def}}{=} R^{-1} + E$ instead of the matrix R^{-1} itself.

Recall that the value of the function is $\mathbf{c} = B\mathbf{v} + \mathbf{e}$, where $\mathbf{v}$ is an integer vector and $\mathbf{e}$ is the "error vector". Thus the vector $\mathbf{v}'$ computed by the inversion routine is

$$\mathbf{v}' = T\lceil R'\mathbf{c}\rfloor = T\left\lceil (R^{-1} + E)(B\mathbf{v} + \mathbf{e})\right\rfloor = \mathbf{v} + T\left\lceil R^{-1}\mathbf{e} + E(B\mathbf{v} + \mathbf{e})\right\rfloor$$

where the last equality follows since $R^{-1}B\mathbf{v}$ is an integral vector so we can take it out of the rounding operation and then we have $TR^{-1}B\mathbf{v} = \mathbf{v}$. Therefore, we invert correctly if and only if

$\lceil R^{-1}\mathbf{e} + E(B\mathbf{v} + \mathbf{e})\rfloor = \bar{\mathbf{0}}$, which means that all the entries in $R^{-1}\mathbf{e} + E(B\mathbf{v} + \mathbf{e})$ are less than a $\frac{1}{2}$ in absolute value. The size of the entries in the vector $R^{-1}\mathbf{e}$ is analyzed in Section 3.2, so here we only consider the vector $E(B\mathbf{v} + \mathbf{e})$.

Recall that all the entries in E are less than $2^{-\ell}$ in absolute value, and that the error vector $\mathbf{e}$ consists only of small entries (e.g., for our parameters, the entries in $\mathbf{e}$ are always less than 10). Thus the contribution of the vector $E\mathbf{e}$ can be at most $10 \cdot 2^{-\ell}$ in each coordinate, so we might as well ignore it. To evaluate the entries in $EB\mathbf{v}$, assume that we represent each entry in the matrix B using k bits, and each entry in the vector $\mathbf{v}$ using m bits. Then, each entry in the vector $EB\mathbf{v}$ must be smaller than $n \cdot 2^{k+m-\ell}$ in absolute value.

For example, if we work in dimension 200, use 16 bits for each entry in B and 16 bits for each entry in $\mathbf{v}$, and keep only the 64 most significant bits of each entry in R^{-1} then the entries in $EB\mathbf{v}$ will be bounded by $200 \cdot 2^{16+16-64} \approx 2^{-24}$. Thus, a sufficient condition for correct inversion is that each entry in $R^{-1}\mathbf{e}$ is less than $\frac{1}{2} - 2^{-24}$ in absolute value (as opposed to less than $\frac{1}{2}$ which we get when we store the exact values for R^{-1}). Clearly, this has almost no effect on the probability of inversion errors.

3.5 Security Analysis

In this section we provide some analysis for the security of the suggested trapdoor function by considering several possible attacks and trying to analyze their work-load. We start with evaluating the work-load of a brute-force attack.

3.5.1 Brute-Force Attack

An obvious pre-processing step in every attack on this construction is to reduce the public basis B to get a better basis B' which can then be used to invert the function. Notice that the only feature of R which we used when we analyzed the error-probability is that the rows of R^{-1} have small Euclidean norm (in other words, R has a very small dual orthogonality defect). If we can find another basis with this property, then we can use it just as well. However, finding a basis with a very low dual orthogonality defect is assumed to be a hard problem.

Thus, we assume that even after the lattice reduction, the attacker still have a basis B' with a rather large dual orthogonality defect.[3] For the sake of simplicity, we assume that the basis used by the attacker is the public basis B. Trying to use the basis B for inverting the function in the same manner as we use the basis R means that given the ciphertext $\mathbf{c} = B\mathbf{v} + \mathbf{e}$, we compute $B^{-1}\mathbf{c} = \mathbf{v} + B^{-1}\mathbf{e}$. Then we can do an exhaustive search for the vector $\mathbf{d} \stackrel{\text{def}}{=} B^{-1}\mathbf{e}$. Below we give an approximate analysis for the size of the search space that the attacker needs to go through before it finds the correct vector $\mathbf{d}$.

Denote the i'th entry in $\mathbf{d}$ and $\mathbf{e}$ by δ_i and ϵ_i respectively. The i'th row of B^{-1} by $\hat{\mathbf{b}}_i$ and the (i,j)'th element in B^{-1} by β_{ij}. Using these notations we can write $\delta_i = \hat{\mathbf{b}}_i \circ \mathbf{e} = \sum_j \beta_{ij}\epsilon_j$, and therefore

$$E[\delta_i] = 0 \text{ and } VAR[\delta_i] = \sum_j \beta_{ij}^2 E[\epsilon_j^2] = (\sigma \|\hat{\mathbf{b}}_i\|)^2$$

where $\|\hat{\mathbf{b}}_i\|$ is the Euclidean norm of the i'th row of B^{-1}.

To evaluate the size of this search space for $\mathbf{d}$, we make the simplifying assumptions that each entry δ_i in $\mathbf{d}$ is Gaussian, and that the entries are independent. Based on these simplifying

[3] This will be the case, for example, if the public basis B is obtained by applying a "good lattice-reduction algorithm" to the basis which was obtained by mixing the private basis R.

assumptions, the size of the effective search space is exponential in the differential entropy of the Gaussian random vector $\mathbf{d}$. Recall that the differential entropy of a Gaussian random variable x with variance σ^2 is $h(x) = \frac{1}{2}\log(\pi e \sigma^2)$. Since we assume that the δ_i's are independent, then the differential entropy of the vector $\mathbf{d}$ equals the sum of the differential entropies of the entries, so we get

$$h(\mathbf{d}) = \frac{1}{2}\sum_i \log(\pi e \sigma^2 \|\hat{\mathbf{b}}_i\|^2) = \frac{n}{2}\log(\pi e \sigma^2) + \sum_i \log \|\hat{\mathbf{b}}_i\|$$

so the size of the search space is $2^{h(\mathbf{d})} = (\pi e)^{n/2} \cdot \sigma^n \cdot \prod_i \|\hat{\mathbf{b}}_i\| = (\pi e)^{n/2} \cdot \sigma^n \cdot \text{orth-defect}^*(B)/det(B)$. Note that the term $\det(B)$ in the last expression depends only on the lattice and is independent of the actual basis B.

Typical numeric values. In the experiments which we performed (in dimension 100 with $\sigma = 2$) evaluating this last expression on the (LLL-reduced) public bases resulted in typical work-load of about $10^{70} \approx 2^{230}$.

3.5.2 Other Attacks

In this section we discuss other possible lines of attack against the scheme. One rather obvious improvement on the brute force attack which is described above is to use a better approximation algorithm for the CVP. In particular, instead of using Babai's "Round-off" algorithm we can use the "Nearest-plane" algorithm which was also described in [Ba86]. On a high-level, the difference between the Round-off and the Nearest-plane algorithms is that in the Nearest-plane, the rounding in the different entries are done adaptively (rather that all at once).

One way to describe the Nearest-plane algorithm (which is somewhat different than the way it is described in [Ba86]) is as follows: Given the point $\mathbf{c}$ and the (LLL reduced) basis $B = \{\mathbf{b}_1, \cdots, \mathbf{b}_n\}$ (in the order induced by the LLL reduction). We compute the representation of $\mathbf{c}$ as a linear combination of the $\mathbf{b}_i$'s, $\mathbf{c} = \sum_i \delta_i \mathbf{b}_i$, but we only look at the last coefficient δ_n. We then replace $\mathbf{c}$ with the point $\mathbf{c}' = \mathbf{c} - \lceil \delta_n \rfloor \mathbf{b}_n$, and replace $\mathbf{b}_n$ with a vector $\mathbf{b}_n^*$ which is orthogonal to all the other basis vectors. Denote the new basis by $B' = \{\mathbf{b}_n^*, \mathbf{b}_1, \cdots, \mathbf{b}_{n-1}$. We then apply the same process to $\mathbf{c}'$ and B' (this time looking at the coefficient of $\mathbf{b}_{n-1}$). We repeat this until we eliminate all the vectors from the original basis B. It is clear that if at each step we got the right coefficient then the vector which is left at the end is just the error vector $\mathbf{e}$.

As was pointed to us by Don Coppersmith, this attack can be improved in practice in several different ways:

- Instead of picking the vectors by the order which was induced by LLL, we can pick them by the size of the Euclidean norm in the corresponding rows of B^{-1}. As we showed in the analysis of the brute-force attack, this choice maximizes the probability that the coefficients obtained by rounding are really the right coefficients.

- We can apply a lattice-reduction procedure to the remaining basis-vectors after each iteration (or once every few iterations). This improvement is particularly useful since the performance of the lattice-reduction algorithm improves rapidly as the dimension decreases. Also, we can round more than one coefficient at a time (if there are several vectors for which the corresponding rows of B^{-1} have small norm).

- If all the rows in B^{-1} have a large Euclidean norm, we can apply an exhaustive search similar to our brute force attack to the few entries which has the smallest Euclidean norm. That is, we try to continue the same algorithm for each plausible setting of these entries. Since we only a few entries (and we picked the ones with the smallest norm), we expect that the size of this exhaustive search will be rather small.

To defeat this attack, we must make the dimension of the original lattice large enough so that *all the rows in B^{-1} will have large Euclidean norm.* Although we did not perform extensive tests of this attack, the data that we have so far indicates that when using LLL as our lattice-reduction algorithm, we need to do some exhaustive search even for dimension 120. It seems that when using LLL, this attack is infeasible for dimensions above 140. We still do not have data about the performance of this attack using better lattice-reduction algorithms.

4 Encryption Scheme

Our public-key Encryption scheme is based on our candidate one way trapdoor function in the usual way. That is, to encrypt a message we embed it inside the argument to the function, compute the function and the result is the ciphertext. To decrypt, we use the trapdoor information to invert the function and extract the message from the argument.

Recall from Section 3 that, in high level, our one way trapdoor function takes a lattice vector and adds to it a small error vector. In the context of an encryption scheme, we say that we 'encrypt a lattice vector' by adding to it a small error vector, and the resulting vector in $\mathcal{R}^n$ is the ciphertext. To encrypt arbitrary messages, we specify an (easily invertible) encoding which maps messages into lattice vectors which are then encrypted as above. Decrypting the ciphertext amounts to solving a particular type of CVP instances which was discussed in Section 3. In a nutshell, the Encryption scheme can be described as follows (using the algorithms Generate, Sample, Evaluate and Invert from the description of our trapdoor function).

Generating Keys. On security parameter 1^n, run algorithm Generate(1^n) to get the triplet (B, R, σ). We let the public key be (B, σ) and the secret key to be (R^{-1}, T) where $T = B^{-1}R$.

Encryption. On input message s and public key (B, σ), we first apply some (randomized) encoding function $\mathbf{v} \leftarrow Enc(s)$ to encode s as a vector $\mathbf{v} \in \mathcal{Z}^n$. We note that this encoding is in fact the only component of the encryption scheme which is not directly implied by the trapdoor function construction. We discuss this encoding function in Section 4.2. (For now, we let Enc, Dec denote a pair of public and easy to compute functions such that $Dec(Enc(s)) = s$.)

Once we computed $\mathbf{v}$, we pick at random an "error-vector" $\mathbf{e} \in \mathcal{R}^n$ according to the distribution induced by the Sample algorithm from Section 3. We then apply the function $f_{B,\sigma}$ to $\mathbf{v}$ and $\mathbf{e}$ to get the ciphertext $\mathbf{c} \leftarrow f_{B,\sigma}(\mathbf{v}, \mathbf{e}) = B\mathbf{v} + \mathbf{e}$.

The operations involved in encrypting a message are therefore: (1) Encoding it as a integer vector; (2) Choosing a random vector; and (3) Performing one matrix-vector multiplication and one vector addition. Thus we have an $O(n^2)$ algorithm for encryption (where n is the dimension we work in).

Decryption. To decrypt $\mathbf{c}$ we use the private key to invert the function $f_{B,\sigma}$ by setting $\mathbf{v} \leftarrow T\lceil R^{-1}\mathbf{c}\rfloor$. We then extract the message s from the vector $\mathbf{v}$ by setting $s = Dec(\mathbf{v})$. Decrypting a message amounts to two matrix-vector multiplications and one rounding operation on a vector. Thus we also have an $O(n^2)$ algorithm for decryption.

Detecting decryption errors. One property of the above decryption procedure is that although there is a probability of error, it is still possible to verify when the message is decrypted correctly. This enables the legitimate user to identify decryption errors, so that it can take measures to correct them. Recall that we encrypt the lattice point $\mathbf{p}$ by adding to it a small error vector $\mathbf{e}$, thus obtaining the ciphertext $\mathbf{c} = \mathbf{p} + \mathbf{e}$. When we decrypt $\mathbf{c}$ and find a lattice point $\mathbf{p}'$ (which we hope is the same as $\mathbf{p}$), we can verify that this is the right lattice point by checking that the error $\mathbf{e}' = \mathbf{c} - \mathbf{p}'$ is indeed small. For example, if we pick the error vector so that it never contains any entry larger than σ, then we can check that $\epsilon_i \leq \sigma$ in each component. Thus we get

Fact 4.1: If the underlying lattice does not contain any non-zero vector with $L_\infty \leq 2\sigma$ then decryption errors can always be detected.

Plaintext Awareness. It seems that our scheme enjoys some weak notion of "plaintext awareness" in that there is no obvious way to generate from scratch a valid ciphertext (i.e., one which the decryption algorithm can decrypt) without knowing the corresponding lattice point. Still this plaintext awareness is limited, since after seeing one valid ciphertext c, it is possible to generate other valid ciphertexts without knowing the corresponding lattice-points (simply by adding any lattice point to c).

4.1 Partial Information Attacks

In addition to the attacks on the underlying trapdoor function which were outlined in Section 3.5, there are types of attacks which only make sense in the context of encryption scheme. Namely, rather than trying to recover the original message itself, the attacker can instead try to extract from the ciphertext some partial information about the message (e.g., the value of a specific bit in it). On way in which such partial information attacks can be mounted against this scheme is as follows:

Recall that the ciphertext is computed as $\mathbf{c} = B\mathbf{v} + \mathbf{e}$ and therefore $B^{-1}\mathbf{c} = \mathbf{v} + \mathbf{d}$, where $\mathbf{d}$ is defined $\mathbf{d} \stackrel{\text{def}}{=} B^{-1}\mathbf{e}$. Thus, the i'th entry of $(B^{-1}\mathbf{c})$ is equal to $\nu_i + \delta_i$ (ν_i, δ_i are the i'th entries in $\mathbf{v}, \mathbf{d}$ respectively). We saw in Section 3.2 that if the Euclidean norm of the row $\hat{\mathbf{b}}_i$ in B^{-1} is small, then the variance of δ_i will also be small (notice that the dual-orthogonality-defect of B may still be large because of other rows in B^{-1} that have much larger Euclidean norm). In particular, if $\sigma \cdot \|\hat{\mathbf{b}}_i\| < 1$ then there is a reasonable probability that $|\delta_i| < 1/2$, in which case ν_i is just the i'th entry of the rounded vector $[B^{-1}\mathbf{c}]$.

Thus, an attacker could just focus on the rows of B^{-1} which have low Euclidean norm, and try to compute the corresponding entries in $\mathbf{v}$. Knowing some of the entries in $\mathbf{v}$ may - in turn - give some partial information about the message s. More generally, the adversary may view the i'th entry of $B^{-1}\mathbf{c}$ as an estimate for ν_i (which is probably accurate up to $\sigma\|\hat{\mathbf{b}}_i\|$), and use this partial knowledge about the entries in $\mathbf{v}$ to obtain some partial knowledge about s.

Somewhat surprisingly, for the purpose of this attack - reducing the basis B does not seem to help (of course, as long as the resulting basis is not "reduced enough" to break the underlying trapdoor function). To see why, consider the unimodular transformation T' between the original basis B and the reduced basis B' ($T' = (B')^{-1}B$). Since $\mathbf{c}$ is computed using the original matrix B, then when trying to extract partial information using B' we compute

$$\mathbf{v}' = (B')^{-1}\mathbf{c} = (B')^{-1}(B\mathbf{v} + \mathbf{e}) = (B')^{-1}B\mathbf{v} + (B')^{-1}\mathbf{e} = T'\mathbf{v} + (B')^{-1}\mathbf{e}$$

If $(B')^{-1}$ has rows with small Euclidean norm, then the attacker may be able to learn the corresponding entries in $T'\mathbf{v}$, but this still does not seem to yield an estimate about any entry in $\mathbf{v}$. It follows that in this encryption scheme, it may be useful to publish public basis which is not LLL reduced.

In any case, foiling the partial information attacks requires a careful design of the encoding scheme $\mathbf{v} \leftarrow Enc(s)$, so that partial information that can be revealed about $\mathbf{v}$ will not yield partial information about s. This is discussed next.

4.2 Encoding messages as vectors in $\mathcal{Z}^n$

In this section we discuss ways to encode messages as vectors in $\mathcal{Z}^n$. As we mentioned above, we would like to have an encoding scheme such that knowing a few entries in v exactly and knowing some rough estimate on all the other entries still yields "almost no information" about s.

In choosing an encoding function, there are other parameters (besides security) which need to be considered. Perhaps the most important of them is to obtain high bandwidth: Since for every encryption operation we end up sending the vector $\mathbf{c} = B\mathbf{v} + \mathbf{e}$, we would like to use as much of this bandwidth as possible for message bits.

The Trapdoor Function Paradigm: Using hard bits. The first approach is a generic one. Since we have a candidate for a trapdoor one-way function, we may use hard-core bits of this function as the message bits. In particular, we can use the general construction of Goldreich-Levin, [GL84]) which shows how and where to hide hard core bits in a pre-image of any one-way function. (This construction enables hiding $\log n$ bits in one function evaluation.)

This approach has the advantage of being able to prove that it is impossible to even distinguish in polynomial time between any two messages, under the assumption that we started with a trapdoor function. The major drawback is in terms bandwidth, since we can only send $\log n$ bits at a time for one function evaluation. Moreover, since this approach is generic, it doesn't provide us with any insight which we may exploit to increase the bandwidth.

Using the low-order bits in v. Another approach is to embed the bits of s directly in the vector $\mathbf{v}$. Since an attacker can get an estimate for the entries in $\mathbf{v}$, then it is clear that we need to embed s in the least significant bits of these entries. Also, the fact that the attacker may be able to learn exactly some of the entries in $\mathbf{v}$ implies that we should not put any bits of s in those entries. Note that we know in advance which are the "weak entries", since these correspond to the rows in B^{-1} with small Euclidean norm.

We start by examining the simple case in which we only use the least-significant-bit of each entry (except for the "weak entries"). and pick all the other bits at random. Then, given an estimate $\tilde{\nu}_i = \nu_i + \delta_i$ for the entry ν_i, the attacker should decide whether the number in that entry was even or odd (that is, whether the message bit is a 0 or 1).

If we assume that each entry in $\tilde{\nu}_i$ can be approximated by a Gaussian random variable with mean ν_i and variance $\sigma^2\|\hat{\mathbf{b}}_i\|^2$ (which is reasonable since $\tilde{\nu}_i$ is a sum of n independent random variable which are all "more or less the same", then given the experimental value $\tilde{\nu}_i$, the statistical advantage $|\Pr[\nu_i \text{ is even } \mid \tilde{\nu}_i] - \Pr[\nu_i \text{ is odd } \mid \tilde{\nu}_i]|$ is exponentially small in $\sigma\|\hat{\mathbf{b}}_i\|$. Thus, if the Euclidean norm of $\hat{\mathbf{b}}_i$ is large enough, then the attacker, who knows $\tilde{\nu}_i$, gets only a small statistical advantage in guessing the corresponding bit of s. If we have a row of B^{-1} with very high Euclidean norm, then we may be able to use the corresponding entry of $\mathbf{v}$ for ℓ message-bits. It can be shown that the statistical advantage in guessing any of these bits is at most exponentially small in $\sigma\|\hat{\mathbf{b}}_i\|/2^\ell$. If the Euclidean norm of each individual row in B^{-1} is too small, we can represent each bit of s using several entries by making that bit the XOR of the least significant bit in all those entries.

Reducing mod 2 Notice that using only the least significant bits for the bits of s is really a linear operation, since we can write $\mathbf{v} = \mathbf{s} + 2\mathbf{r}$, where $\mathbf{s}$ is the $\{0,1\}$ vector with the message bits and $\mathbf{r}$ is a random integer vector. Therefore, when using this encoding we should consider attack in which all the matrices involved are reduced modulo 2, and the attacker tries to compute the vector $\mathbf{v}$ mod 2.

Namely, we have $\mathbf{c} = B\mathbf{v} + \mathbf{e} = B\mathbf{s} + 2B\mathbf{r} + \mathbf{e}$, so when reduce the last equation mod 2 we get $\mathbf{c} = B\mathbf{s} + \mathbf{e} \pmod 2$. We can now compute the inverse of B mod 2 *over* Z_2. If such an inverse exists then denote it by B_2^{-1}. In this case we get $B_2^{-1}\mathbf{c} = \mathbf{s} + B_2^{-1}\mathbf{e} \pmod 2$. Notice, however, that $\mathbf{e}$ mod 2 is a random binary vector which is 1 with probability σ^2, and so - for each entry δ_i of $\mathbf{d} = B_2^{-1}\mathbf{e}$ mod 2 - the statistical difference $|\Pr[\delta_i = 0] - \Pr[\delta_i = 1]|$ is exponentially small with n.

5 Signature scheme

In this section we describe a slight modification of our trapdoor function which is more suitable for a signature scheme, and provide an initial assessment of its properties. In this signature scheme, just like in the encryption scheme, the user uses its private basis B to find lattice points which are close to some given vectors in $\mathcal{R}^n$. In this scheme, we "sign a vector in $\mathcal{R}^n$" by providing a lattice point which is "rather close" to that vector. The public key for the signature scheme contains a public basis B for the lattice, and a threshold $\tau > 0$ which defines how close should the lattice point be to the given vector. The choice of τ is discussed in Section 5.3.1.

5.1 Operation

The key-generation procedure amounts to the generation of two bases (as in the GENERATE procedure of the trapdoor function) and to the determination of the threshold τ.

Signature. To sign a message s, we first need to interpret s as a vector in $\mathcal{R}^n$. For this we use some encoding function to get $\mathbf{u} \leftarrow Enc(s)$ (see Section 5.3.2). Then, using the private key (R^{-1}, T) we apply the exact same procedure as for decrypting a message, namely compute $\mathbf{v} \leftarrow T\lceil R^{-1}\mathbf{u}\rfloor$. The vector $\mathbf{v}$ is the signature on s. The signing time is $O(n^2)$ just like for encryption, provided that the encoding time is so bounded.

Notice that $\mathbf{v}$ is an integral vector, which we view as a representation of the lattice point $\mathbf{p} = B\mathbf{v}$. The reason that we expect $\mathbf{p}$ to be "rather close" to $\mathbf{u}$ is that the representation of $\mathbf{u}$ as a linear combination of the columns of R is $R^{-1}\mathbf{u}$, while the representation of $\mathbf{p}$ as a linear combination of the columns of R is $\lceil R^{-1}\mathbf{u}\rfloor$, and these two representations differ by at most a half in each coordinate. We discuss this further in Section 5.3.1.

Verification. To verify a signature $\mathbf{v}$ on message s w.r.t. the public key (B, τ), we compute the vectors $\mathbf{u} \leftarrow Enc(s)$, $\mathbf{p} \leftarrow B\mathbf{v}$, and check that the Euclidean distance between them is less than τ. Namely, the verification process consists of checking the inequality $\|Enc(s) - B\mathbf{v}\| < \tau$. This process too takes time $O(n^2)$, provided again that the encoding time is so bounded.

5.2 On the analog nature of the scheme

We note that because of its "analog nature", our scheme has some properties which are very different than those of other known signature schemes. In particular, notice that if $\mathbf{u}, \mathbf{u}'$ are two vectors in $\mathcal{R}^n$ which are very close to each other (much closer than the threshold τ), then it is very likely that a signature on $\mathbf{u}$ will also be a signature on $\mathbf{u}'$. This "metric preserving" property suggests different signing procedures for digital versus analog data.

If we are signing **digital data** then we should make sure that a signature on one message could not be used to obtain a signature on another message. This can be achieved by the use of a "good hash function" to hash the message before we interpret it as a vector in $\mathcal{R}^n$ (or, alternatively, as the means to map messages to such vectors). If indeed the hash function is good enough, it will ensure that even if two messages to be signed are close to each other at the outset, they will be far apart after being hashed and thus be mapped to different signatures. Note that the *hashing and signing paradigm* is what is necessary and in fact done in practice when using the RSA and DSA signature schemes. The reason is to ensure the difficulty of forging the signature of messages related to those messages signed previously by a legitimate user – a forgery which is otherwise easy in both the case of "bare" RSA and DSA.

On the other hand, the "metric preserving" property may useful when signing **analog data** such as music, speech, images etc. In employing a traditional digital signature scheme to such data, the natural procedure is to first sample the data so as to obtain digital representation of it, and next to apply the signature scheme to this digital representation. This procedure has the disadvantage of potentially mapping close analog signals to different (yet close) digital representations. In particular, minor changes in the either the sampling process or in the analog signal itself, may result in a different digital representation. Consequently, the signature may not be valid when the analog signal changes a little. Thus, a method such as ours, where the analog signal may be signed directly have an advantage of supplying signatures which remain valid (or at least meaningful) under small changes of the analog signal.

Note that the above discussion depends on the encoding of data as vectors in $\mathcal{R}^n$. Each of the two settings calls for a different type of encoding. In the "digital" setting we wish the encoding to scramble messages so to destroy any structure (e.g., related messages should yield unrelated encoding). In the "analog" setting we want the encoding to preserve the metric of the data space (e.g., close analog signals should yield close encoding in $\mathcal{R}^n$). For further details see Section 5.3.2.

5.3 Various choices

In addition to the choices made for the process of selecting the private and public bases (discussed in Section 3), there are two important choices to be made: Firstly, we need to determine the threshold parameter τ, and secondly we need to determine the method of encoding data as vectors in $\mathcal{R}^n$.

5.3.1 Choosing the threshold

In this section we show how the threshold τ should be chosen so that the signature algorithm is successful with high probability, and in Section 5.4 we examine the effects of the choise of the security of the signature scheme.

In the analysis below we use the following notations. Let A be a basis for some lattice in $\mathcal{R}^n$. We denote by $\text{Round}_A(\mathbf{u})$ the lattice point which is generated from $\mathbf{u}$ by considering the representation of $\mathbf{u}$ as a linear combination of the vectors in B and rounding the coefficients to the nearest integers. That is, $\text{Round}_A(\mathbf{u}) \stackrel{\text{def}}{=} A \lceil A^{-1}\mathbf{u} \rfloor$.

Consider now a random vector $\mathbf{u} \in \mathcal{R}^n$ and we try to evaluate the distance between $\mathbf{u}$ and $\text{ROUND}_R(\mathbf{u})$, where R is the private basis. Recall that conceptually, the lattice point $\text{ROUND}_R(\mathbf{u})$ is the signature on the vector $\mathbf{u}$ (though the actual signature is the representation of that point w.r.t. the public basis B). Define the "error vector"

$$\mathbf{e} \stackrel{\text{def}}{=} [R^{-1}\mathbf{u}] - R^{-1}\mathbf{u}$$

that is, the i'th entry ϵ_i in $\mathbf{e}$ is the difference between the i'th coefficient in the representation of $\mathbf{u}$ as a linear combination of the vectors in R and the nearest integer. Then the distance between $\mathbf{u}$ and $\text{ROUND}_R(\mathbf{u})$ is just the Euclidean norm of the vector $R\mathbf{e}$. Clearly, we have $|\epsilon_i| \leq 1/2$ for all i, since this is just the difference between some real number and the nearest integer. This immediately gives us

Claim 5.1: Let R be the private basis used for signing and denote the maximum L_1 norm of any row in R by γ. If we set $\tau = \gamma\sqrt{n}/2$ then the signing algorithm always succeeds (with probability 1).

Proof: Denote $\mathbf{d} \stackrel{\text{def}}{=} R\mathbf{e}$. We can write the i'th entry in $\mathbf{d}$ as $\delta_i = \sum_j \rho_{ij}\epsilon_j$ where ρ_{ij} is the i,j'th entry in R and ϵ_j is the j'th entry in $\mathbf{e}$. Therefore $|\delta_i| \leq \sum_j |\rho_{ij}\epsilon_j| \leq \frac{1}{2}\sum_j |\rho_{ij}| \leq \frac{1}{2}\gamma$ and so

$$\|\mathbf{d}\| = \sqrt{\sum_{i=1}^{n} \delta_i^2} \leq \sqrt{\sum_{i=1}^{n} \left(\frac{1}{2}\gamma\right)^2} = \gamma\sqrt{n}/2$$

□

As for the trapdoor function, we may choose to set a lower value for τ to get a better security. We now describe an "approximate analysis" which enables us to estimate the failure probability for lower values of τ.

(As opposed to the situation with the trapdoor function, however, even these approximate estimates are not very good. Experimentally, we found that we can set the value of τ to be about half the value which we get from the analysis below.)

Recall that the distance between $\mathbf{u}$ and $\text{ROUND}_R(\mathbf{u})$ is the Euclidean norm of the vector $R\mathbf{e}$ where $|\epsilon_i| \leq 1/2$ for all i. Moreover, if $\mathbf{u}$ is chosen uniformly at random from a large enough box in $\mathcal{R}^n$ then the distribution induced over the vector $\mathbf{e}$ is close to the uniform distribution over $(-\frac{1}{2}, +\frac{1}{2}]^n$.

To see that, notice that if we choose $\mathbf{u}$ uniformly from the parallelepiped $\{\sum_i \gamma_i\mathbf{r}_i \ : \ 0 \leq \gamma_i < 1\}$ (where $\mathbf{r}_i$ is the i'th column of R) then the induced distribution on $\mathbf{e}$ is exactly the uniform distribution. Moreover, every large enough box in $\mathcal{R}^n$ can be viewed as union of many disjoint parallelepipeds like that, plus some "left over" volume. As the volume of the box increases, the fraction of this "left over" volume decreases. Thus, the induced distribution of the vector $\mathbf{e}$ gets closer to the uniform distribution.

Thus, to evaluate the distance between $\mathbf{u}$ and $\text{ROUND}_R(\mathbf{u})$ when $\mathbf{u}$ is uniform in some large box in $\mathcal{R}^n$, we need to evaluate the Euclidean norm of the vector $R\mathbf{e}$ when $\mathbf{e}$ is uniform in $(-\frac{1}{2}, +\frac{1}{2}]^n$. We can write each entry in this vector as $\delta_i = \sum_j \rho_{ij}\epsilon_j$ where ρ_{ij} is the i,j'th entry in R and ϵ_j is the j'th entry in $\mathbf{e}$.

Denote the largest entry in R by $\rho_{\max}$, then each of the random variables $\rho_{ij}\epsilon_j$ is distributed in the interval $[-\frac{\rho_{\max}}{2}, +\frac{\rho_{\max}}{2}]$ and has zero mean. Using Hoeffding bound we conclude that for any $\delta > 0$

$$\Pr[|\delta_i| > \delta n] < 2 \cdot \exp\left(-\frac{2\delta^2 n}{\rho_{\max}^2}\right)$$

which implies that

$$\Pr\left[\|\mathbf{d}\|^2 > \delta^2 n^3\right] \le \Pr\left[\exists\ i\ s.t.\ \delta_i^2 > \delta^2 n^2\right] \le 2n \cdot \exp\left(-\frac{2\delta^2 n}{\rho_{\max}^2}\right)$$

Therefore, to make the error probability less than ε it is sufficient to set $\delta > (\rho_{\max} \cdot \ln(2n/\varepsilon)/2n)$, which means that the threshold is set to

$$\tau = \sqrt{\delta^2 n^3} \ge \frac{\rho_{\max}}{2} \ln(2n/\varepsilon)\sqrt{n} \tag{2}$$

Typical numeric values. The value of the threshold which we obtain from Equation 2 with $\varepsilon = 2^{-30}$ and $n = 140$ is

$$\tau = \frac{\rho_{\max}}{2} \ln(280 \cdot 2^{30})\sqrt{140} \approx 156\rho_{\max}$$

In our experiments we typically have $\rho_{\max} = 4$, which implies that $\tau \approx 625$. If we are willing to settle for $\varepsilon = 10^{-4}$ the we can make $\tau \approx 350$. As we said above, we found that experimentally we can actually use threshold value which is about half of what we get from these bounds. In particular, for the setting above we can set $\tau = 200$ to get the error probability below 10^{-4}.

5.3.2 Encoding messages as vectors in $\mathcal{R}^n$

Recall that in the above scheme, a lattice-point $\mathbf{p}$ is considered a valid signature on a vector $\mathbf{v}$ if the two vectors are "close enough". This means that the same lattice point $\mathbf{p}$ is valid with respect to many different vectors (in fact, all the vectors in a sphere of radius τ centered at $\mathbf{p}$). This fact has two implications: On one hand, we can allow many "slightly different" representations of the same "logical datum" without effecting the validity of the signature. On the over hand, vectors which represent different "logical datum" must be very different from one another.

Signing analog data. As a simple example of an analog data, consider attaching a digital signature to a FAX document (say, by printing a bar-code containing the signature on the document itself). Clearly, in this case we cannot expect that the senders digital representation of the document will be identical to the representation obtained by the receiver after the document is printed. However, suppose that we could represent the "contents" of the FAXed document using some small set of parameters, in such a way that

- Printing and re-scanning the document does not change its parameters very much; and
- Documents which contains different contents are represented by very different sets of parameters.

If we have such representation, we could use these sets of parameters to represent a document as a vector in $\mathcal{R}^n$. Consequently, it will be very likely that a digital signature on some representation of the document will still be valid even after the document was printed and re-scanned. We will need to assume that such a representation will be sufficiently rich in the sense that documents of interest will results in representations in a sufficiently large box of $\mathcal{R}^n$. (Clearly, signatures are easy to forge if documents of interest are all mapped to a small region of $\mathcal{R}^n$ – and carrying the argument to an extreme, we definitely do not want all documents to be mapped to within distance τ of the same lattice point.) Furthermore, it should be infeasible to obtain a meaningful document which matches a random vector in this large box of $\mathcal{R}^n$.

Signing digital data. When signing digital data, we do not have the "multiple representation" problem as above – there is a unique binary string which represents the logical datum. What we need is an encoding of binary strings as "random" points in $\mathcal{R}^n$. We may assume, without loss of generality, that the string has length n, since shorter and longer strings can be handled using well-known methods (such as padding and collision-free hashing, respectively). So what we need is a mapping of $\{0,1\}^n$ to $\mathcal{R}^n$ which does not map two different strings too close to one another (i.e., to within proximity τ). This is very easy to do. However, we want the range of this mapping to be sufficient "random" so that finding a close lattice point will be hard for these mapping-images.

5.4 Security of the Signature Scheme

To get some initial indication for the security of this scheme, we consider what happens when we try to execute the signing algorithm using the public basis B. Here we do not even have an approximate analysis. Instead we conducted experiments to evaluate how close to the threshold we can get when using the public basis for signing. For the same setting as the "typical numeric values" in Section 5.3.1, ($n = 140$, max-entry in $R = 4$), we got distances which were all above 520 (we tried 5 different LLL-reduced bases, 10000 "signatures" for each basis). This suggests that for these parameters, picking the threshold at $\tau = 200$ may be good enough to counter this attack, at least when using LLL as our lattice-reduction algorithm.

6 Experimental Results

We performed several experiments in order to measure the effect of various parameters in the basis generation process on the security of our scheme. Since, as we described in Section 3.5, the security of the scheme is related to the dual-orthogonality-defect of the bases involved, we view the ratio between the dual-orthogonality-defect of the public and private bases as our "measure of security".

Testing methods. For our experiments we used an implementation of the LLL lattice reduction algorithm due to the LiDIA group [Li95]. In each experiment, we chose a private and public bases-pair and evaluated the ratio between their dual-orthogonality-defects. We generated the public basis from the private one by mixing it (as described in Section 3.3) and LLL-reducing the result. To gain some confidence in our results, we repeated this experiment several times for each setting of the parameters.

- For each private basis, we generated five public bases and used the ratio between the minimum dual-orthogonality-defect of these public bases and the dual-orthogonality-defect of the private basis as the "security-level" of this private basis.
- For each setting of the parameters, we generated seven private bases with these setting and considered the median "security-level" of these seven bases.

Parameters. The parameters which we tested are

1. The dimension of the lattice, denoted by n. We performed most of the tests in dimensions 80-120.
2. The range of integers ($\{-l, \cdots, +l\}$) from which we choose the entries in the private basis. Below we refer to this range as the 'l-parameter' of the private basis.

3. How "cube-like" is the private basis. Namely, we generated the private basis as $R = k \cdot I + rand(\pm l)$ for several values of k. (Where I is the identity matrix and $rand(\pm l)$ is a random matrix with entries in $\{-l, \cdots, +l\}$.) Below we refer to this parameter as the 'k-parameter' of the private basis.

4. How many "mixing steps" are used to generate the public basis from the private one.

6.1 Generating the Private Basis

We first measured the effects of the parameters involved in choosing the private basis, namely lattice dimension (n), range of integers (l) and "cube-likeness" (k). For each setting of k, l, we tested dimensions 80 through 120 (in increments of 10).

Entry size (l). We tested the l-parameter settings of 1, 4 and 10, working with both "random lattices" ($k = 0$) and "cube-like lattices" ($k = l\lceil 1 + \sqrt{n}\rceil$). The results of these experiments are summarized in Figure 1. In all these experiments, we applied $2n$ "elementary mixing mixing-steps" to the private bases and LLL-reduced the result to obtain the public basis (See Section **??**. As can be seen from these results, the l-parameter had no effect on the "security-level" of the bases which we obtained.

"Cube-like" parameter (k). The settings of the k-parameter which we tested are $k = 0, k = \frac{1}{2}l\lceil 1 + \sqrt{n}\rceil$ and $k = l\lceil 1 + \sqrt{n}\rceil$. The reason that we express k in "units" of $l\sqrt{n}$ is that the expected length of a random vector in $\{-l \cdots + l\}^n$ is $O(l\sqrt{n})$. We tested these settings with $l = 1$ and $l = 4$. The results are summarized in Figure 2. Varying the value of k had the following effects

- Increasing the value of k increases the dimensions in which LLL can recover the private basis. For example, LLL could recover the private basis in dimension 80 when we set $k = l\lceil 1 + \sqrt{n}\rceil$, but failed for the smaller values of k.

- When the dimensions increase beyond some threshold, the ratio of the dual-orthogonality-defect becomes much larger for large values of k. The reason is that the dual-orthogonality-defect of the private basis becomes smaller (since the private basis is more "cube-like"). In fact, for $k = l\lceil 1 + \sqrt{n}\rceil$, the dual-orthogonality-defect of the private basis is already very close to one. On the other hand, the dual-orthogonality-defect of the corresponding public basis is not affected by this change (since beyond some threshold dimension, LLL fails to take advantage of the "cube-likeness" of the lattice). Thus, the ratio between the dual-orthogonality-defect of the public and private basis increases considerably.

6.2 How Many Mixing Steps

We also tested the number of "elementary mixing steps" which we apply to the private basis in order to get the public basis. In each elementary mixing step, we pick one of the basis vectors and add to it a random integral linear combination of the other vectors. In our experiments we chose the coefficients of this linear combination from $\{-1, 0, 1\}$ with $\Pr[1] = \Pr[-1] = 1/7$. To make sure that we replace all the vectors in the private basis, we must make at least n mixing steps. To make sure that we hit them all, we chose a random permutation over $\{1, \cdots n\}$ and picked the vectors according to the order in that permutation.

To evaluate how "secure" is the resulting public basis, we LLL-reduced it and compared the dual-orthogonality-defect of the result with that of the private basis. In our experiments we tried

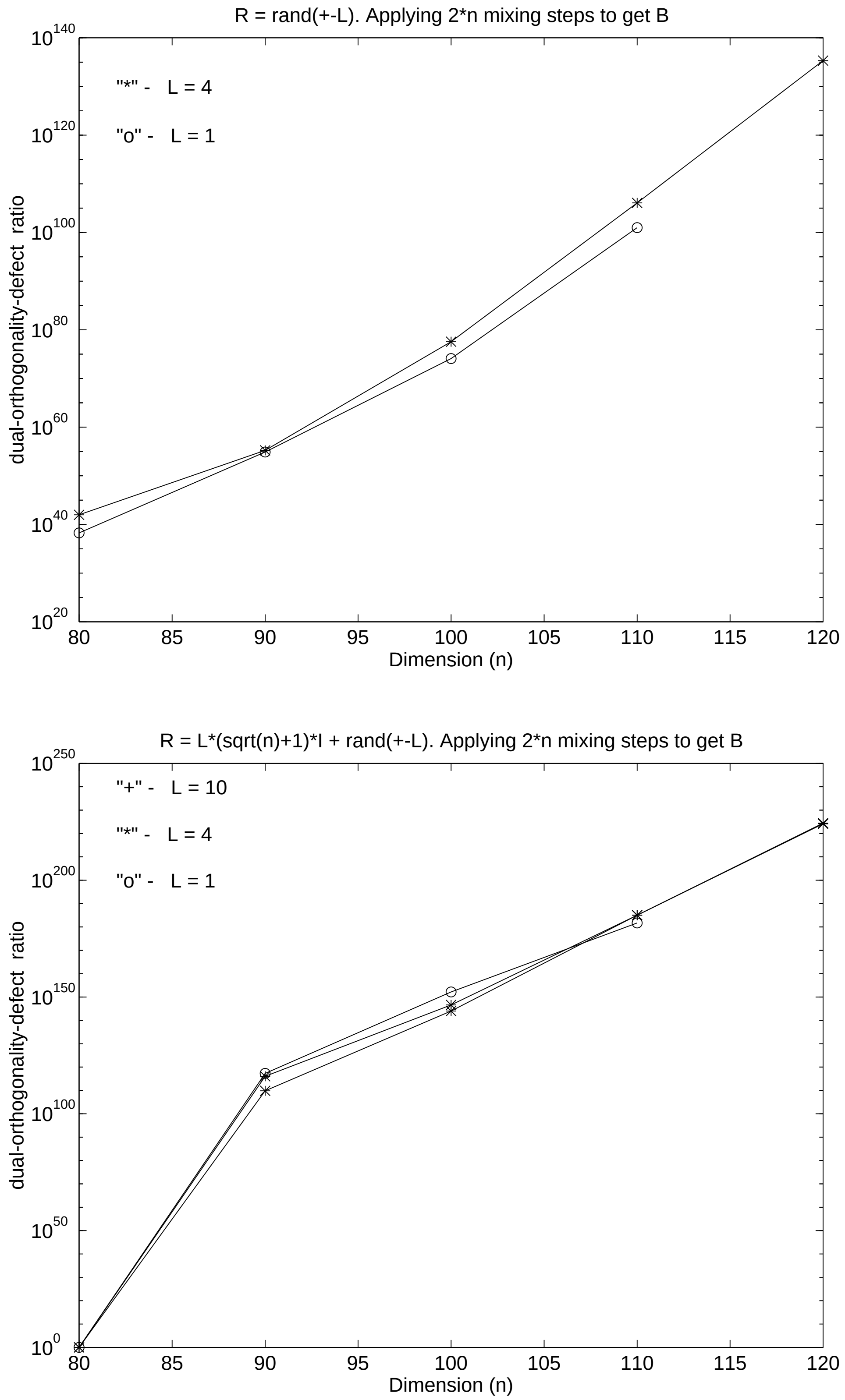

Figure 1: The effect of varying the entry size l for $k = 0$ (upper figure) and $k = \lceil 1 + \sqrt{n} \rceil \cdot l$ (lower figure).

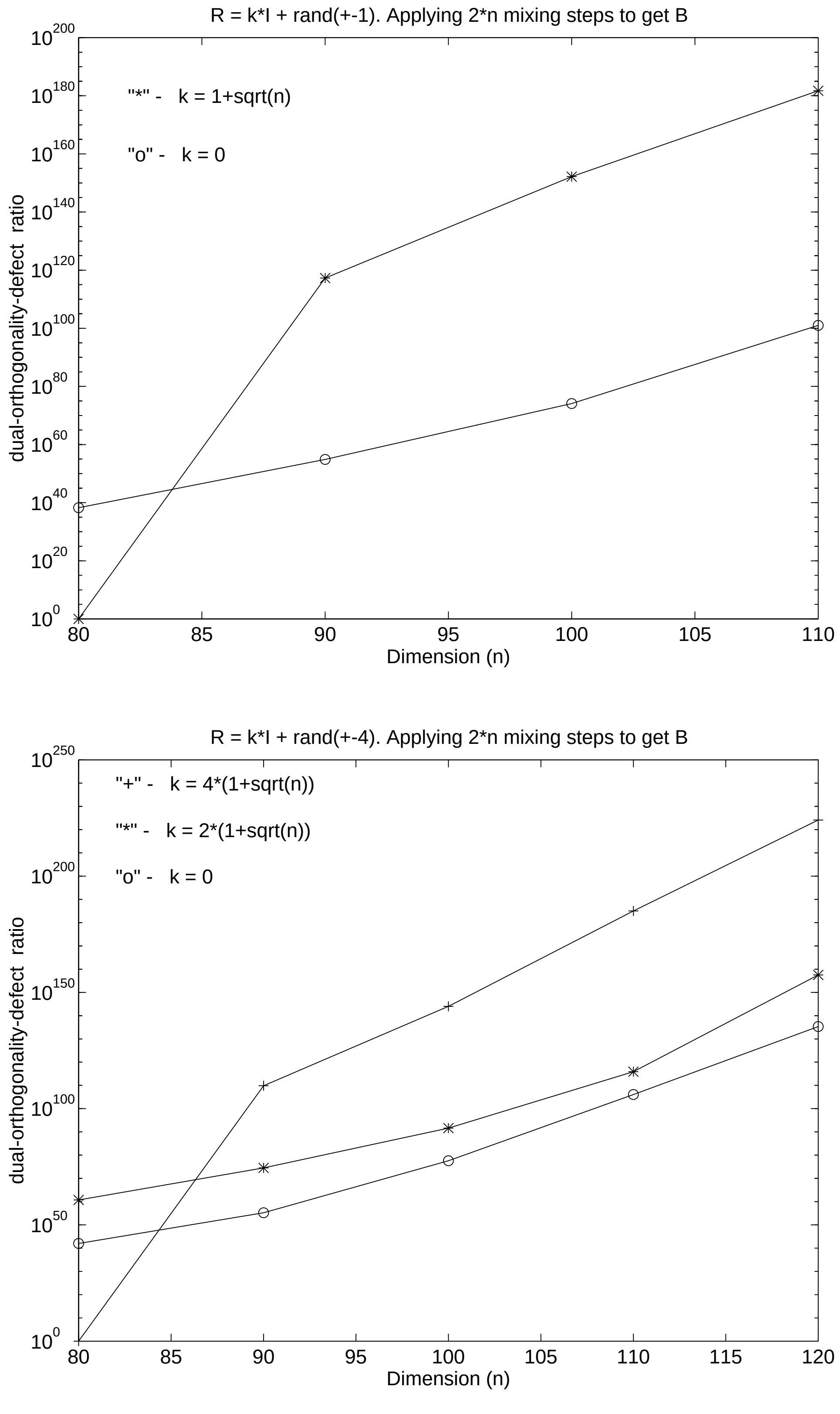

Figure 2: The effect of varying the parameter k for $l = 1$ (upper figure) and $l = 4$ (lower figure). We tessted the values $k = 0, k = \frac{1}{2} l \lceil 1 + \sqrt{n} \rceil$ and $k = l \lceil 1 + \sqrt{n} \rceil$.

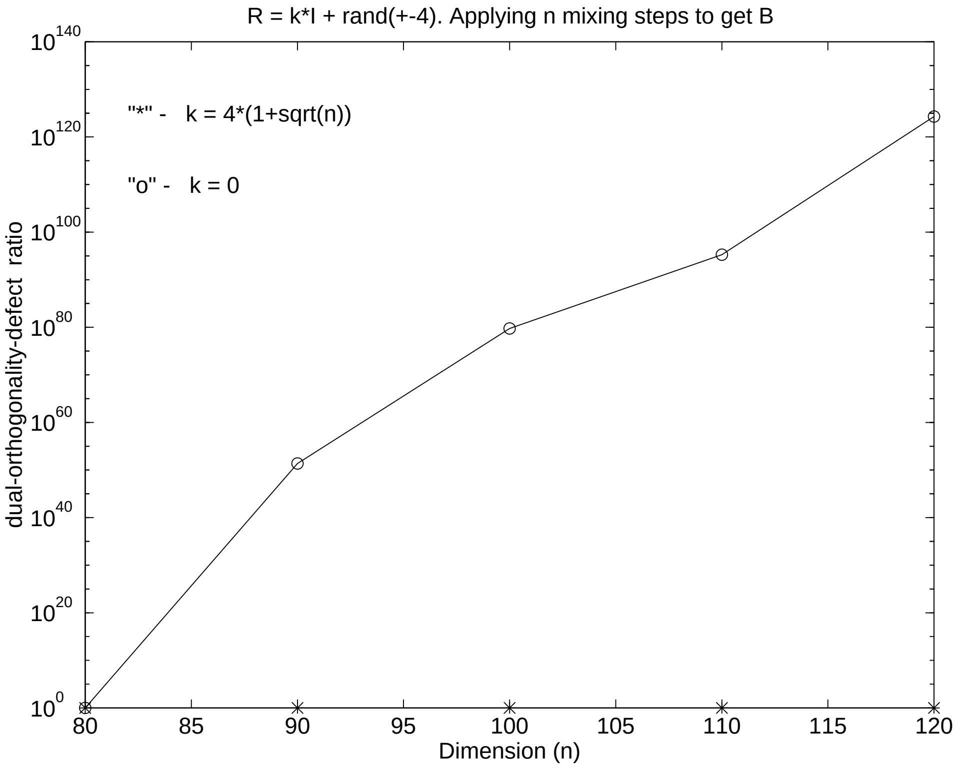

Figure 3: Making only n mixing-steps. Notice that for a cube-like lattice, we were able to recover the private basis in all the dimensions.

to make n and $2n$ mixing steps before the LLL-reduction. The results for $2n$ mixing steps (with various parameters of the private basis) are presented in Figures 1 and 2. The results we get when we only make n mixing steps (for $l = 4$ and $k = 0, k = l\lceil 1 + \sqrt{n}\rceil$) are summarized in Figure 3.

It can be seen that when making only n mixing steps on a cube-like lattice, LLL was always able to recover the private basis. Another problem with making so few mixing steps (which is not reflected in Figure 3) is that the variance which we get for each setting of the parameters is much larger than what we get for $2n$ mixing steps. In fact, although the median results for $k = 0$ seem to increase exponentially with the dimension, the minimum results are very close to one even in dimension 120.

Acknowledgments.

The authors thank Dan Boneh, Don Coppersmith, Claus Schnorr and Jacques Stren for several enlightening conversations.

References

[Aj96] M. Ajtai. Generating hard instances of lattice problems. In Proceedings of the 28th Annual ACM Symposium on Theory of Computing, pages 99-108, Philadelphia, PA, 1996.

[Ba86] L. Babai, On Lovász lattice reduction and the nearest lattice point problem. in *Combinatorica*, vol. 6, 1986, pp. 1-13.

[BG84] M. Blum and S. Goldwasser. An Efficient Probabilistic Public-Key Encryption Scheme which Hides All Partial Information. in *Proceedings of CRYPTO '84*, Springer-Verlag, 1985, pp. 289-299.

[Bo81] P. van Emde Boas, Another $\mathcal{NP}$-complete problem and the complexity of computing short vectors in a lattice. Reprot 81-04, Mathematische Instituut, University of Amsterdam, 1981.

[BL96] D. Boneh and R. Lipton, Algorithms for Black-Box Fields and Their Application to Cryptography. in *Proccedings of CRYPTO '96*, Lecture Notes in Computer Science No. 1109, Springer-Verlag, 1996. pp. 283-297.

[DSS] Digital Signature Standard (DSS). FIPS PUB 186, 1994.

[DH76] W. Diffie and M.E. Hellman. New Directions In Cryptography. *IEEE Transactions on Information Theory*, Vol IT-22, 1976, pp. 644-654.

[El85] T. El-Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Trans. Information Theory*, vol. 31, 1985, pp. 469-472

[G86] O. Goldreich. Two Remarks Concerning the Goldwasser-Micali-Rivest Signature Scheme. in *Proceedings CRYPTO 86*, Lecture Notes in Computer Science No. 263. Springer-Verlag, 1987. pp. 104-110.

[GKL] O. Goldreich, H. Krawczyk and M. Luby. On the existence of pseudorandom generators. *SIAM J. on Computing*, Vol. 22(6), 1993. pp. 1163-1175

[GL84] O. Goldreich and L.A. Levin A Hard-Core Predicate for All One-Way Functions *Proceedings of the 21st ACM Symposium on Theory of Computing*, 1989, pp. 25-32

[GM82] Goldwasser, S. and Micali, S., Probabilistic Encryption. *Journal of Computer and System Sciences*, Vol. 28, 1984, pp. 270-299.

[GMR85] S. Goldwasser, S. Micali and R.L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen Message Attack. *SIAM Journal on Computing*, Vol. 17, no. 2, 1988, pp. 281-308. See comment regarding efficiency in [G86].

[IN89] R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. In *30th Annual Symposium on Foundations of Computer Science*, IEEE. 1989. pp. 236-241

[IR88] S. Rudich and R. Impagliazzo, Limits on the Provable consequences of One-Way Permutations. in *Proceedings of CRYPTO '88*, S. Goldwasser, editor. Lecture Notes in Computer Science, volume 403, Springer-Verlag, 1988. pp. 8-26

[Ka] R. Kannan. Unpublished manuscript.

[Li95] The LiDIA project software-package and user-manual. Available from `ftp://crypt1.cs.uni-sb.de/pub/systems/LiDIA`

[LLL] A.K. Lenstra, H.W. Lenstra, L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen* 261, 515-534 (1982).

[Mc79] R.J. McEliece, A Public-Key Cryptosystem Based on Algebraic Coding Theory. DSN Progress Report 42-44, Jet Propulsion Laboratory

[NY89] M. Naor, and M. Yung, Universal One-Way Hash Functions and their Cryptographic Applications. in *Proc.* 21*st ACM Symp. on Theory of Computing*, 1989, pp. 33-43

[Ra79] M.O. Rabin, Digital Signatures and Public-Key Functions as Intractable as Factorization. Technical Report MIT/LCS/TR-212, M.I.T., 1978.

[Ro90] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, 1990, pp. 387-394.

[RSA] R.L. Rivest, A. Shamir and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, Vol. 21, 1978, pp. 120-126.

[Sc87] C.P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. in *Theoretical Computer Science*, vol. 53, 1987, pp. 201-224

[Sc95] C.P. Schnorr and H.H. Horner, Attacking the Chor-Rivest Cryptosystem by Improved Lattice Reduction. in *Proceedings of EUROCRYPT '95*, Louis C. Guillou and Jean-Jacques Quisquater, editors. Lecture Notes in Computer Science, volume 921, Springer-Verlag, 1995. pp. 1-12

[Wi84] H.C. Williams, Some Public Key Crypto-Functions as Intractable as Factorization. in *Proceedings of CRYPTO 84*, G. R. Blakley and D. C. Chaum, editors. Lecture Notes in Computer Science, volume 196, Springer-Verlag, 1985. Pages 66-70.

[Ya82] A.C. Yao. Theory and Applications of Trapdoor Functions. in *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, 1982, pp. 80-91.