Public-Key Cryptosystems from Lattice Reduction Problems

Oded Goldreich[⋆] Shafi Goldwasser[⋆⋆] Shai Halevi
Weizmann Institute of Science, ISRAEL MIT, Laboratory for Computer Science

{oded, shafi, shaih}@theory.lcs.mit.edu

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

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 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 (e.g., [20, 19, etc.]) and discrete logarithm computations (e.g.,[8, 9, 7, etc]).

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). From this trapdoor function, we then derive a public-key encryption and digital signature methods.

These methods are asymptotically more efficient than the RSA and ElGamal encryption schemes, 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 these systems. Specifically, for security parameter k, the new system has public key of size $O(k^2)$ and computation time of $O(k^2)$, compared to public key of size $O(k)$ and computation time of $O(k^3)$ for the RSA and ElGamal systems. We believe that, given today's technologies, the increase in size of the keys is more than compensated by the decrease in computation time.

[⋆] This research was done while visiting in the Laboratory for Computer Science, MIT.

[⋆⋆] This research was supported by DARPA grant DABT63-96-C-0018.

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 a small error vector to it). 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.

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*. 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.

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 unimodular 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 [4] 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, in such a way that no "partial information" about the message is leaked by the ciphertext (cf., [13]).

There are several ways to do that, and we discuss some of them in Section 4. One generic way is to use hard core bits of the trapdoor function to embed the bits of the message (e.g., [12]). This approach has the advantage of ensuring that the encryption scheme is as secure as the underlying trapdoor function, but it is inefficient in terms of message expansion.

Another plausible way, which may be more efficient, is to map the message to a lattice point by taking the integer combinations of the public basis vectors which is "specified" by the message bits, 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 a scheme based on algebraic coding that was suggested by McEliece's in [18].

A signature scheme. It is also possible to construct a signature scheme along similar lines: 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 may be of great benefit. However, to get secure signatures in the sense of [14], this property pause a significant problem. 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 [10]). Due to lack of space, we do not discuss that construction in this extended abstract.

1.1 Discussion

Our work was inspired by a remarkable result of Ajtai [1] 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.

Independently of our work, Ajtai and Dwork [2] suggested a public-key encryption scheme whose security is reducible to a variant of SVP. Although exhibiting a trapdoor Boolean predicate (which is sufficient for public-key encryption – see [13]), the Ajtai-Dwork construction does not provide a trapdoor function. That is, given the trapdoor information it is possible to decide whether the predicate evaluates to 0 or 1 but not known how to find an inverse. Also, the variant of SVP used in the security proof of [2], called the "poly(n)-unique shortest vector problem" seems to be considerably easier than the general SVP. Finally, we note that the Ajtai-Dwork construction is less efficient than ours, both in terms of the key-size and in terms of encryption time ($O(n^4)$ vs. $O(n^2)$ for both measures). Thus, it seems that their current construction is not really practical.

In retrospect, our encryption scheme bears much similarity to McEliece's scheme [18]. 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.

1.2 Evaluation of Security

To provide some feeling for the security of our construction, we analyzed a few plausible attacks against it and evaluated their effectiveness. Our analysis, combined with extensive testing, indicate that the work-load of these attacks grows exponentially with the dimension of the lattice. In particular, according to our estimates these attacks should be intractable in practice for dimension 300 or so.

1.3 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, and in Section 4 we describes our encryption scheme. In Section 5 we describe our experimental results.

2 Lattices and Lattice Reduction Problems

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. If β is a real number, we denote the integer closest to β by $\lceil \beta \rfloor$ and the smallest integer which is $\geq \beta$ by $\lceil \beta \rceil$. If $\mathbf{b}$ is a vector in $\mathcal{R}^n$, then $\lceil \mathbf{b} \rfloor$ denotes the vector in $\mathcal{Z}^n$ which is obtained by rounding each entry in $\mathbf{b}$ to the nearest integer. 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 $L(B) \stackrel{\text{def}}{=} \{\sum_i k_i \mathbf{b}_i \ : \ k_i \in \mathcal{Z} \text{ for all } i\}$

We call B a *basis* of the lattice $L(B)$. If the vector $\mathbf{v}$ belongs to the lattice L, then we say that $\mathbf{v}$ is a lattice-vector (or a lattice point). 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 $L(B) = \{B\mathbf{v} : \mathbf{v} \text{ is an integral vector}\}$. Below we briefly mention a few well-known facts about lattices. 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. An important fact about lattices is that all the bases of a given lattice have the same determinant (up to the sign). 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 notion of of the *orthogonality defect* of a basis, which was introduced by Schnorr in [21], 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 $\mathrm{orth\text{-}defect}(B) \stackrel{\text{def}}{=} \frac{\prod_i \|\mathbf{b}_i\|}{|det(B)|}$, where $\|\mathbf{b}_i\|$ is the Euclidean norm of the i'th column in B.

Clearly, $\mathrm{orth\text{-}defect}(B) = 1$ if and only if the columns of B are orthogonal to one another, and $\mathrm{orth\text{-}defect}(B) > 1$ 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).

Another important notion in our scheme is the dual lattice. If $B = \mathbf{b}_1, \cdots, \mathbf{b}_n$ is a basis for some lattice in $\mathcal{R}^n$ (where we think of B as an $n \times n$ matrix whose columns are the $\mathbf{b}_i$'s) then the dual lattice of $L(B)$ is the lattice which is spanned by the rows of the matrix B^{-1}. In Section 3.4 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 some natural 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 $\mathrm{orth\text{-}defect}^*(B) \stackrel{\text{def}}{=} \prod_i \|\hat{\mathbf{b}}_i\| / |det(B^{-1}|) = |det(B)| \cdot \prod_i \|\hat{\mathbf{b}}_i\|$, where $\hat{\mathbf{b}}_i$ is the 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 Closest Vector Problem (CVP). 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 [6] to be $\mathcal{NP}$-hard for any l_p norm. Also, Arora et al. [3] proved that approximating the CVP to within any constant factor is also NP-hard.

No polynomial-time algorithm is known for approximating the CVP in $\mathcal{R}^n$ to within a polynomial factor in n. The best polynomial time algorithms for approximating CVP are based on the LLL algorithm [17] and its variants. Babai [4] proved that the CVP in $\mathcal{R}^n$ can be approximated in polynomial time to within a factor of $2^{n/2}$. This was later improved by Schnorr [21] to a factor of $(1+\varepsilon)^n$ for any $\varepsilon > 0$. We note, however, that these bounds refer to worst-case instances, and these algorithms "typically" perform much better than the above upper-bounds.

As we explain in Section 3, an attack against our trapdoor function amounts to finding an exact solution for some random 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. For this problem too there are no known polynomial-time algorithms, and the best polynomial-time approximation algorithms for it are variants of the LLL algorithms, which achieve an approximation ratio of $2^{O(n^2)}$ in the worst case for SBP instances in $\mathcal{R}^n$.

In our public key constructions, finding the private-key from the public-key requires solving some random SBP instances.

3 A Candidate Trapdoor Function

In this section we define our candidate trapdoor function and analyze a few possible attacks against it. Informally, a collection of trapdoor functions consists of four algorithms, GENERATE, SAMPLE, EVALUATE and INVERT, where GENERATE outputs a description of a function and the associate trapdoor information, SAMPLE picks an element in the domain of the function, EVALUATE evaluates the function on that element and INVERT uses the trapdoor information to inverts the function. Below we describe 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 3.2. 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 some pairs of vectors $\mathbf{v}, \mathbf{e} \in \mathcal{R}^n$ (see below).

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[3] from the range $\{-n, \ldots, +n\}$. The vector $\mathbf{e}$ is chosen by setting each entry in it to either $+\sigma$ or $-\sigma$, 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 $p_\sigma = \frac{\sigma^2}{2\lceil\sigma\rceil^2}$, and 0 with probability $1 - 2p_\sigma$.)

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

[3] We do not know if the size of this range has any influence on the security of the construction. The value n is rather arbitrary, and was only chosen to get integers of about 8 bits for the parameters which we work with.

INVERT. Given R and $\mathbf{c}$, we use Babai's Round-off algorithm [4] 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 $\mathbf{v} \leftarrow T\left\lceil R^{-1}\mathbf{c}\right\rfloor$ and $\mathbf{e} \leftarrow \mathbf{c} - B\mathbf{v}$.

3.1 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 4. *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 $\left\lceil R^{-1}\mathbf{e}\right\rfloor \neq \bar{\mathbf{0}}$.*

Proof. Let T be the unimodular transformation matrix $T = B^{-1}R$. Then the inversion algorithm is $\mathbf{v} = T\left\lceil R^{-1}\mathbf{c}\right\rfloor$ 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

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

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 $\left\lceil T^{-1}\mathbf{v} + R^{-1}\mathbf{e}\right\rfloor = T^{-1}\mathbf{v} + \left\lceil R^{-1}\mathbf{e}\right\rfloor$, and therefore

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

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

Theorem 5. *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 omitted.

Although Theorem 5 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 6. *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*

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

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}. We 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. □

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 $n = 120$, $\varepsilon = 10^{-5}$. For a certain setting of the parameters which we tested (in which the entries in R were chosen from the range ± 4), the maximal Euclidean norm of the rows in R^{-1} is about $1/30$. Evaluating the expression above for $\gamma = 1/30$ yields $\sigma \leq \left(\frac{1}{30}\sqrt{8\ln\left(\frac{2\cdot 120}{10^{-5}}\right)}\right)^{-1} \approx \frac{30}{11.9} \approx 2.5$.

3.2 The Generate Algorithm

In this section we discuss various aspects of the Generate algorithm. We described in Section 3.1 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

- The choice of the private basis R induces a distribution on the lattices in $\mathcal{Z}^n$.
- For any private basis R, the process of "mixing" R to get the public basis B induces some distribution on the bases of $L(R)$.

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

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 [22]). We speculate that working in dimensions about 250-300 should be good enough with respect to this algorithm.

Distribution of the private bases. We considered two possible distributions for choosing the private basis.

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).

Choosing an "almost rectangular lattice": We start from the box $k \cdot I$ in $\mathcal{R}^n$ (for some number k), and add "noise" to each of the box vectors. Namely, we 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, so it may be possible to choose a larger value of σ. On the other hand, it may also allow an attacker to obtain a basis with smaller dual orthogonality factor by reducing the public basis. Our experiments show that we get the best parameters when k is about $\sqrt{n} \cdot l$.

Generating the public basis. Once we have the private basis R, we should pick the public basis B according to some distribution on the bases of the lattice $L(R)$. We tested two methods for generating B from R:

In the *first method*, we transform R into B via a sequence of many "mixing steps", in which we take one basis vector and add to it a random integer linear combination of the other vectors.

In our experiments, we went through the basis vectors one at a time, to make sure that we replace them all. The coefficients in the linear combination were chose 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. Our experiments indicate that using $2n$ mixing steps was sufficient to prevent LLL from recovering the original basis.

In the *second method* we multiply R by a few "random" unimodular matrices to get B, namely $B = R \cdot T_1 \cdot T_2 \cdots$. Each of these unimodular transformation matrices is chosen as a product of and upper- and lower-triangular matrices, $T_i = L_i U_i$, where the diagonal entries in L_i, U_i are ± 1. In our experiments, we chose the other non-zero entries in L_i, R_i from $\{-1, 0, 1\}$. We found that we need to multiply R by at least four transformation matrices to prevent LLL from recovering the original basis. Also, our experiments show that this process generates public matrices with larger entries than using $2n$ mixing steps according to the previous method. Thus, we chose to use the first method for most of our experiments.

3.3 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. Although it is possible to store the exact values for R^{-1}, the entries in R^{-1} may have hundreds of bits of precision, which makes working with them rather inefficient. 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 of R^{-1}.

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 (modified) 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 $\left\lceil R^{-1}\mathbf{e} + E(B\mathbf{v} + \mathbf{e}) \right\rfloor = \bar{\mathbf{0}}$, 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 3.1, 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 entries of error vector $\mathbf{e}$ are all $\pm\sigma$ (for our choice of parameters, we have $\sigma \approx 3$). Thus the contribution of the vector $E\mathbf{e}$ can be ignored. 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 8 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 $200 \cdot 2^{16+8-64} \approx 2^{-32}$. Thus, a sufficient condition for correct inversion is that each entry in $R^{-1}\mathbf{e}$ is less than $\frac{1}{2} - 2^{-32}$ 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.

3.4 Security Analysis

In this section we provide some initial analysis for the security of the suggested trapdoor function by considering several possible attacks and trying to analyze their work-load. An obvious pre-processing step in just about every attack on our construction is to reduce the public basis B to get a better basis B' which can then be used for the attack. For the sake of simplicity, we therefore assume that the public basis itself is already reduced via a "good lattice-reduction algorithm".

Our numerical estimates for the work-load of the various attacks are based on experiments reported in Section 5. In these experiments we used the implementation of the LLL lattice-reduction algorithm from the LiDIA project [16]. The bottom line of our experiments is that all the attacks below become infeasible in dimensions above 150. We do not have data about the performance of these attack using better lattice-reduction algorithms (such as the ones described in [22]. We speculate that when using these better algorithms, the attacks will become infeasible in dimensions about 250-300.

The Round-off Attack The most obvious attack on our scheme (other than a brute-force search for the error vector $\mathbf{e}$) is to try and use the public basis B for inverting the function in the same manner as we use the private basis R. Namely, given the output of the function $\mathbf{c} = B\mathbf{v} + \mathbf{e}$, 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 $E[\delta_i] = 0$ and $\mathrm{Var}[\delta_i] = \sum_j \beta_{ij}^2 E[\epsilon_j^2] = (\sigma||\hat{\mathbf{b}}_i||)^2$, 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 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

$$2^{h(\mathbf{d})} = (\pi e)^{n/2} \cdot \sigma^n \cdot \prod_i ||\hat{\mathbf{b}}_i|| = (\pi e)^{n/2} \cdot \sigma^n \cdot \frac{\text{orth-defect}^*(B)}{|det(B)|}$$

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 Subsection 5.2 we describe experiments which we performed in dimension 80 through 160. Upto dimension 80, the LLL algorithm is capable of reconstructing a "good" basis, so that the work-load of this attack is essentially 1. In higher dimensions, however, LLL fails to provide a good basis, and consequently the work load of the Round-off attack grows by a factor of about 8000 per dimension. Thus, already in dimension 100 this attack is worse than the trivial brute-force search for the error vector $\mathbf{e}$.

The Nearest-plane Attack One rather obvious improvement to the Round-off attack from 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 [4]. 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). More precisely, the Nearest-plane algorithm works as follows: It is given a point $\mathbf{c}$ and an LLL reduced basis $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ (in the order induced by the LLL reduction). It then considers all the affine spaces

$$H_k = \left\{ k\mathbf{b}_n + \sum_{i=1}^{n-1} \alpha_i \mathbf{b}_i \ : \alpha_i \in \mathcal{R} \right\}$$

for all $k \in \mathcal{Z}$, finds the hyperplane H_k which is closest to the point c, and projects the point $c - k\mathbf{b}_n$ onto the $(n-1)$-dimensional space which is spanned by $\{\mathbf{b}_1, \ldots, \mathbf{b}_{n-1}\}$. This yields a new point c' and a new basis $B' = \{\mathbf{b}_1, \ldots, \mathbf{b}_{n-1}\}$, and the algorithm now proceeds recursively to find a point p' in this $(n-1)$-dimensional lattice which is close to c'. Finally, the algorithm sets $p = p' + k\mathbf{b}_n$.

It was pointed to us by Don Coppersmith that the Nearest-plane 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}. An analysis similar to Subsection 3.1 shows that this choice locally maximizes the probability that the hyperplane H_k is the correct one (this analysis is omitted from this extended abstract).
- We can "peel off" more than one vector in each level of the recursion, if there are several vectors for which the corresponding rows of B^{-1} have small norm.
- We can apply a lattice-reduction procedure to the remaining basis vectors in each level of the recursion. This improvement is particularly useful since the performance of the lattice-reduction algorithm improves rapidly as the dimension decreases.
- If all the rows in B^{-1} have a large Euclidean norm, we can apply an exhaustive search to the few entries which has the smallest Euclidean norm. That is, instead of just trying the closest H_k, we can also try the second closest one, etc.

The work-load of the Nearest-plane attack can be analyzed and tested in a similar manner to that of the Round-off attack: We can describe this attack as consisting of an off-line phase, in which we construct from the public basis B another matrix $\hat{B}$, and an on-line phase in which $\hat{B}$ used in a manner similar to the way B^{-1} is used in the Round-off attack. An estimate for the work-load of this attack can be computed from the Euclidean norm of the rows in $\hat{B}$. Due to lack of space, the analysis is omitted from this extended abstract.

Experiments reported in Subsection 5.3 indicate that the Nearest-plane attack has a much lower work-load than the Round-off attack. Nonetheless, its work-load also grows exponentially with the dimension of the lattice. Our experiments show that when using LLL as our lattice-reduction algorithm, some amount of search is needed starting from dimensions 110-120, and the attack becomes infeasible in dimensions 140-150.

The Embedding Attack Finally, another heuristic which is often used to approximate CVP (and which was brought to our attention by Clause Schnorr and Don Coppersmith) is to embed the n basis-vectors and the point $\mathbf{c}$ for which we want to find a close lattice point in an $(n+1)$-dimensional lattice like so

$$B' = \begin{pmatrix} | & | & | & & | \\ c & b_1 & b_2 & \cdots & b_n \\ | & | & | & & | \\ 1 & 0 & 0 & & 0 \end{pmatrix}$$

Then we use a lattice reduction algorithm to search for the shortest non-zero vector in $L(B')$, in the hope that the first n entries in this vector will be the closest point to c. As opposed to the other attacks, we do not know how to use the output of this attack as a starting point for an exhaustive search (in the case where the output is not the "right lattice point"). Thus the only thing that we can measure about this attack is whether it works or not. Some experiments which we made with this attack (using LLL as our tool for finding shortest vectors) indicate that this heuristic works up to dimensions about 110-120. Recall that the Round-off attack becomes worse than the simple exhaustive search already in dimension 100.

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 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 can think of this process as 'encrypting a lattice vector' by adding to it a small error vector, and we can think of the resulting vector in $\mathcal{R}^n$ as the ciphertext. To encrypt arbitrary messages, we must specify an (easily invertible) encoding which maps messages into lattice vectors which are then encrypted as above. Describing such an encoding is the focus of this section.

To obtain a semantically secure encryption scheme [13], we need an encoding scheme such that seeing the ciphertext does not help a polynomial time adversary in getting "any information" about the message. Other parameters which need to be considered (besides security) are the efficiency of encoding and decoding, and the message expansion. Below we describe two possible encoding methods.

4.1 A Generic Encoding

The first method 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, [12]) 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 of this scheme is the message expansion, 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.

4.2 Encoding via the low-order bits in $\mathbf{v}$

Another approach is to embed the bits of the message directly in an integer vector $\mathbf{v}$, and then compute the ciphertext as $\mathbf{c} = B\mathbf{v} + \mathbf{e}$, where B is the public basis and $\mathbf{e}$ is an error vector.

The main problem with this approach is that the adversary can in fact use $\mathbf{c}$ to obtain an estimate on each entry in $\mathbf{v}$. To see that, notice that $B^{-1}\mathbf{c} = \mathbf{v} + B^{-1}\mathbf{e}$, and so each entry in $B^{-1}\mathbf{c}$ is equal to the corresponding entry in $\mathbf{v}$ plus some "noise" from $B^{-1}\mathbf{e}$. Below we denote $\mathbf{d} \stackrel{\text{def}}{=} B^{-1}\mathbf{e}$. Also, the i'th entry in $\mathbf{d}$ is denoted by δ_i and the i'th entry of $\mathbf{v}$ is denoted ν_i.

We saw in Section 3.1 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 can be obtained simply by rounding the i'th entry in $B^{-1}\mathbf{c}$ to the nearest integer. Thus, an attacker could focus on the rows of B^{-1} which have low Euclidean norm, and compute the corresponding entries in $\mathbf{v}$. 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 $\sigma||\hat{\mathbf{b}}_i||$).

Remark. 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 $\mathbf{v}$. This suggests that in this encryption scheme, it may be useful to publish public basis which is not LLL reduced.

Embedding the message in the vector $\mathbf{v}$. From the above discussion, it is clear that if we are to embed the message in the vector $\mathbf{v}$ itself, then it should be embedded in the least significant bits of $\mathbf{v}$'s entries. Also, we should not put any bits of the message in entries of $\mathbf{v}$ which correspond to rows with small Euclidean norm in B^{-1}. 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 $\sigma||\hat{\mathbf{b}}_i||$. 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 the message. If we have a row of B^{-1} with very high Euclidean norm, 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 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. The statistical advantage then is exponentially small in $\sigma \cdot \sum_i ||\hat{\mathbf{b}}_i||$ (where the sum is taken over the XOR'ed entries).

4.3 Additional Properties

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 all the entries in the error vector $\mathbf{e}' = \mathbf{c} - \mathbf{p}'$ are $\pm\sigma$. Thus as long as the lattice does not contain a point in which all the entries are exactly $\pm 2\sigma$, 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).

5 Experimental Results

Throughout this work, we used experimental data to guide us through the choices of various parameters in our construction, and to help us evaluate the effectiveness of some of the attacks. In this section we describe our testing methods and sketch a few of the main results. A full report on these tests will be available in the full version of this paper. For these experiments we used the implementation of the LLL lattice-reduction algorithm from the LiDIA project [16].

5.1 Choosing Parameters for the Key-Generation

The tests which we performed to determine the parameters of the key-generation process are omitted from this extended abstract. These tests are described in the TR version of this work [11].

5.2 Evaluation of the Round-Off Attack

We used the analysis from Subsection 3.4, in conjunction with our tests, to evaluate the performance of the Round-off attack in dimensions 80 through 160 (in increments of 10). We performed the following experiments:

1. In each dimension n we generated *five private bases*. Each basis was chosen as $R_i = 4\lceil\sqrt{n}\rceil \cdot I + \mathrm{rand}(\pm 4)$, where I is the identity matrix and $\mathrm{rand}(\pm 4)$ is a square matrix whose entries are selected uniformly from the range $\{-4, \ldots, +4\}$.

For each basis R_i we computed the value σ_i (which is used for the error vector in our construction) as $\sigma_i = (\gamma_i \sqrt{8\ln(2n/\varepsilon)})^{-1}$, where γ_i is the Euclidean norm of the largest row in R_i^{-1}, and $\varepsilon = 10^{-5}$. (By the Remark at the end of Subsection 3.1, we estimate that the probability of decryption errors using the private basis R_i and this value of σ_i is about 10^{-5}.)

2. For each private basis R_i we generated *five public bases*. Each public basis B_{ij} was generated by first applying $2n$ "mixing steps" to R_i and then LLL-reducing the resulting basis. As explained in Subsection 3.4, we evaluated the work-load of the Round-off attack using the public basis B_{ij} as

$$\text{work-load}_{ij} = (\pi e)^{n/2} \cdot \sigma_i^n \cdot \frac{\text{orth-defect}^*(B_{ij})}{|det(B_{ij})|}$$

3. We evaluated the work-load of the Round-off attack against the private basis R_i by the minimum of the work-loads for the corresponding private bases $B_{i\star}$. Namely we set $\text{work-load}_i = \min_j \text{work-load}_{ij}$.

4. We evaluated the "typical work-load" in dimension n, by the median of the work-loads for the five private bases in this dimension.

The results of these tests in dimensions 80-160 are summarized in Figure 1. It can be seen in the figure that this attack falls apart once the dimension grows above 90, where the work-load increases by an amazing multiplicative factor of about 8000 per dimension (!!). Clearly, in dimensions 100 and above it is already easier to perform an exhaustive search for the value of the error vector $\mathbf{e}$ than to use the Round-off attack.

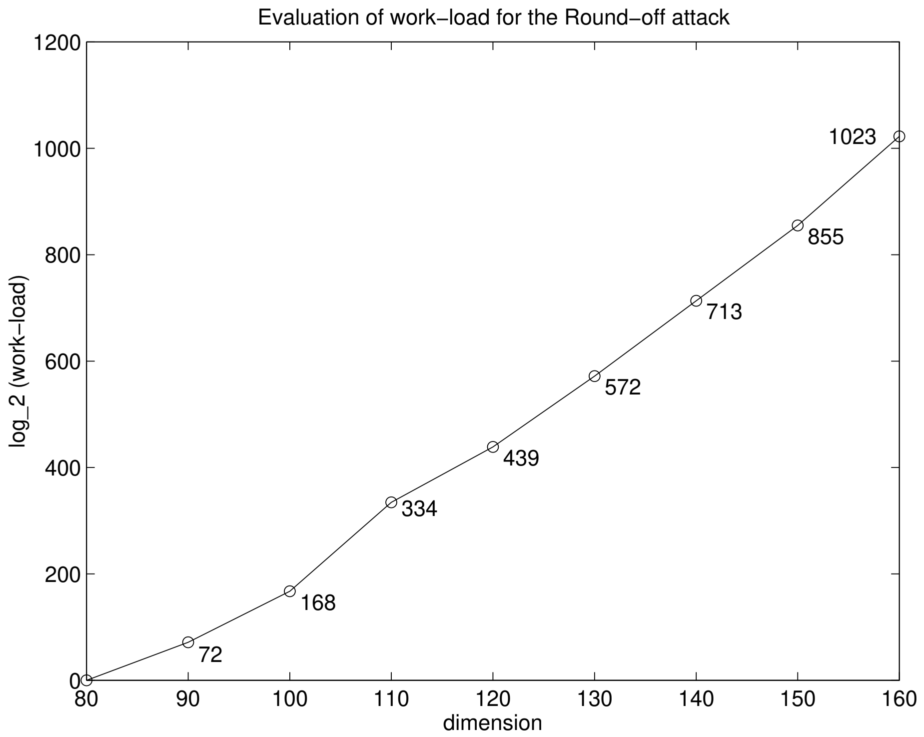

Fig. 1. Evaluation of the work-load of a Round-off attack in dimensions 80-160.

5.3 Evaluation of the Nearest-plane Attack

To evaluate the Nearest-plane attack, we used the same private bases R_i and public bases B_{ij} as for the Round-off attack. For each of the public bases B_{ij}, we carried out the off-line phase in the Nearest-plane attack, thereby generating the transformed matrices $\hat{B}_{ij}$. We then used the $\hat{B}_{ij}$'s to evaluate the work-load of the attack.

As before, the work-load for a private basis R_i is the minimum work-load for all the $\hat{B}_{ij}$'s, and the "typical" work-load for dimension n is the median work-load of the private bases in this dimension.

The results of our tests in dimensions 100-170 are summarized in Figure 2. As the figure clearly demonstrates, this attack is far better than the Round-off attack. Nonetheless, once the dimension grows above 110, the work-load monotonically increases by a multiplicative factor of about 4 per dimension. In dimensions 140-150 this attack is already infeasible. Extrapolating from this line, we estimate that in dimensions higher than 200, it would be easier to perform an exhaustive search for the value of the error vector $\mathbf{e}$ than to use the Nearest-plane attack.

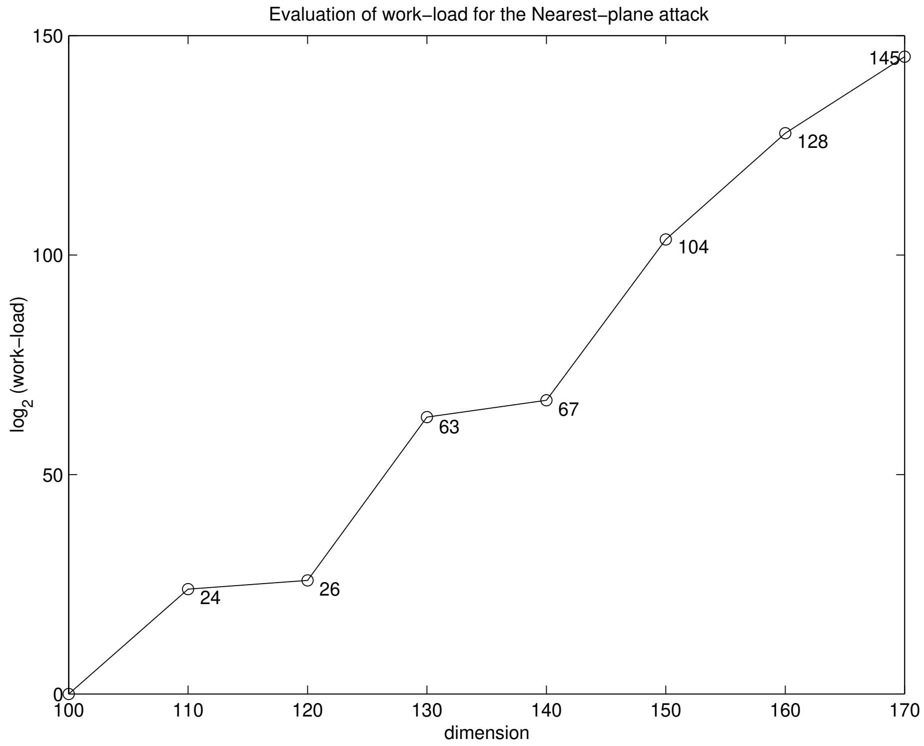

Fig. 2. Evaluation of the work-load of a Nearest-plane attack in dimensions 100-170.

5.4 Evaluation of the Embedding Attack

As we said in Subsection 3.4, we do not know how to turn a failed run of the Embedding attack into a starting point for some exhaustive search, and so we cannot talk about the "work-load" of this attack. Instead, we only measured what is the maximum value of σ (the bound on the error vector) for which this attack works.

For these experiments we used the same private bases R_i and public bases B_{ij} as for the previous two attacks. We then used each public basis to evaluate the function on a few points using a few different values of σ, and tested whether the Embedding attack recovers the encrypted message.

In our experiments we tested several values of σ between 1 and 3. For each setting of σ, we encrypted five messages and declared the attack successful if it recovered at least one of them. For each private basis R_i we computed the highest value of σ for which one of the B_{ij} was successful. For any dimension n we then computed the median among the σ values of the private bases in this dimension.

In Figure 3 we draw these values of σ for dimensions 80-130. These value are compared to the values of σ which we suggest to use in our construction to obtain a probability of 10^{-5} for decryption errors. It can be seen from this figure that for this choice of σ, the Embedding attack stops working around dimensions 110-120.

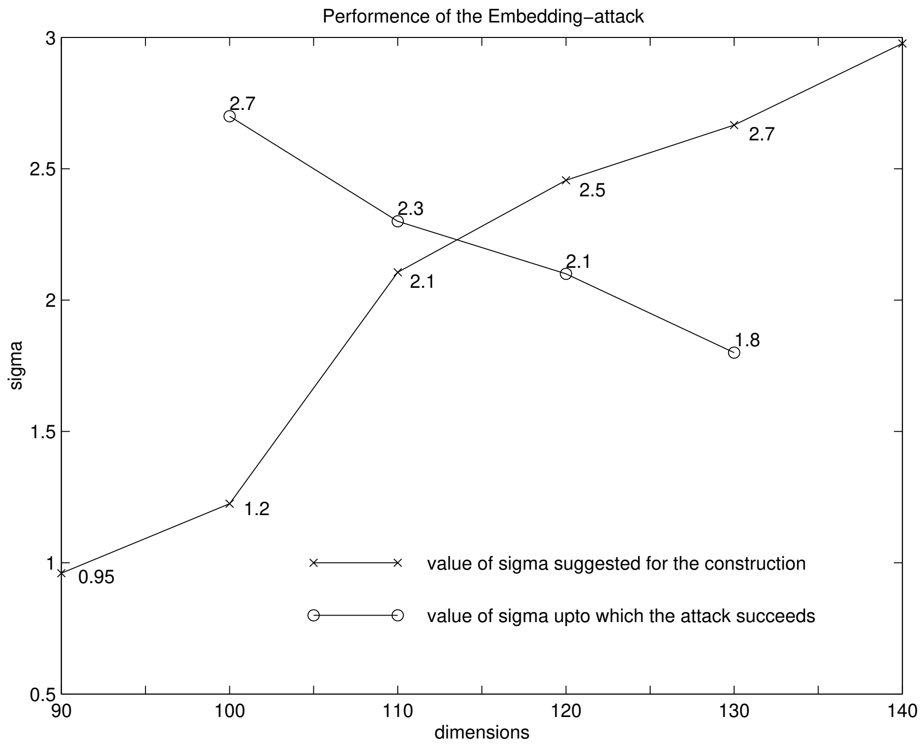

Fig. 3. Performance of the Embedding attack in dimensions 100-130

Acknowledgments.

We thank Svetoslav Tzvetkov for participating in the implementation of the experiments reported in Section 5. We also thank Dan Boneh, Don Coppersmith, Claus Schnorr and Jacques Stren for several very helpful conversations.

References

1. M. Ajtai. Generating hard instances of lattice problems. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 99-108, 1996.
2. M. Ajtai and C. Dwork. A Public-Key Cryptosystem with Worst-Case/Average-Case Equivalence, In *29th ACM Symposium on Theory of Computing*, pages 284-293, 1997.
3. S. Arora, L. Babai, J. Stern, and Z. Sweedyk. The hardness of approximate optimia in lattices, codes, and systems of linear equations. In *Journal of Computer and System Sciences*, 54(2), pages 317-331, 1997.
4. L. Babai, On Lovász lattice reduction and the nearest lattice point problem. in *Combinatorica*, vol. 6, 1986, pp. 1-13.
5. 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.

6. 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.
7. Digital Signature Standard (DSS). FIPS PUB 186, 1994.
8. W. Diffie and M.E. Hellman. New Directions In Cryptography. *IEEE Transactions on Information Theory*, Vol IT-22, 1976, pp. 644-654.
9. 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
10. O. Goldreich, S. Goldwasser and S. Halevi Collision-Free Hashing from Lattice Problems. Theory of Cryptography Library: Record 96-09. Available from `http://theory.lcs.mit.edu/~tcryptol/1996/96-09.html`
11. O. Goldreich, S. Goldwasser and S. Halevi Public-Key Cryptosystems from Lattice Reductions Problems. ECCC Report TR96-056. Available from `http://www.eccc.uni-trier.de/eccc-local/Lists/TR-1996.html`
12. 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
13. S. Goldwasser and S. Micali, Probabilistic Encryption. *Journal of Computer and System Sciences*, Vol. 28, 1984, pp. 270-299.
14. 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.
15. R. Kannan. Algorithmic Geometry of Numbers. in *Annual Review of Computer Science*, vol. 2, 1987, Annual Reviews Inc.
16. The LiDIA project software-package and user-manual. Available from `http://www.informatik.th-darmstadt.de/TI/LiDIA/`
17. A.K. Lenstra, H.W. Lenstra, L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen* 261, 515-534 (1982).
18. R.J. McEliece, A Public-Key Cryptosystem Based on Algebraic Coding Theory. DSN Progress Report 42-44, Jet Propulsion Laboratory
19. M.O. Rabin, Digital Signatures and Public-Key Functions as Intractable as Factorization. Technical Report MIT/LCS/TR-212, M.I.T., 1978.
20. 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.
21. C.P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. in *Theoretical Computer Science*, vol. 53, 1987, pp. 201-224
22. 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

This article was processed using the LaTeX macro package with LLNCS style