A New Design for Lattice-Based Cryptographic Constructions

Charles F. de Barros[1], **L. Menasché Schechter**[2]

[1]Graduate Program in Informatics (PPGI)
Federal University of Rio de Janeiro (UFRJ) – Rio de Janeiro – RJ – Brazil

[2]Department of Computer Science (DCC)
Federal University of Rio de Janeiro (UFRJ) – Rio de Janeiro – RJ – Brazil

charles.barros@ppgi.ufrj.br, luisms@dcc.ufrj.br

Abstract. *In this paper, we propose a new type of construction for a secure and efficient public-key cryptosystem, which is based on a new problem from the theory of lattices.*

1. Introduction

There is no doubt that lattice-based cryptography is very promising. Constructions can be quite efficient, simple and some of them enjoy very strong security proofs. But things are not always as good as they seem. Known lattice-based cryptographic constructions are in the middle of a tradeoff between efficiency and provable security, and the task of achieving both is not simple.

There are a lot of well known lattice-based cryptographic constructions. It all began back in the 90's, with [Ajtai and Dwork 1997], but GGH [Goldreich et al. 1997] was the first practical proposal. After extensive attacks [Nguyen 1999, Lee and Hahn 2010], GGH has been regarded as a deprecated system [Goldreich 1999], even after some improvements were proposed [Micciancio 2001, Yoshino and Kunihiro 2012].

A more efficient and secure lattice-based cryptosystem called NTRU was proposed by Hoffstein, Pipher and Silverman [Hoffstein et al. 1998], but the state of the art in terms of strong security proofs is represented by the constructions based on the Learning With Errors (LWE) problem [Regev 2005, Lyubashevsky et al. 2013].

We are currently developing a study on a new lattice problem, which may be used as an underlying problem to an efficient public-key cryptosystem. The advantage of our proposal lies in its simplicity, since it only deals with integer arithmetic and simple linear algebra computations.

2. The Basics

In this section, we present some notations and conventions. We also provide basic definitions, which may be useful as a theoretical background.

2.1. Notations and Conventions

Vectors will be considered *row vectors* and represented by lowercase boldface letters, such as $\mathbf{u}$. The j-th entry of the vector $\mathbf{u}$ will be represented by $\mathbf{u}_j$. Matrices will be represented by uppercase letters. The entry at the i-th row and j-th column of the matrix

A will be denoted by $A_{i,j}$. The notation $A > 0$ (conversely $A < 0$) implies that all the entries of A are greater (or less) than 0.

We also employ the symbols $[\pm a]^n$, where a is an integer, to denote the vector $\mathbf{a} \in \mathbb{Z}^n$ whose entries are either a or $-a$, and $\lceil \mathbf{u} \rfloor$ to denote the vector in which each entry is the nearest integer to the corresponding entry of $\mathbf{u}$.

2.2. Definitions

Lattices play an essential role in our cryptosystem. Algebraically speaking, a lattice is a discrete additive subgroup of $\mathbb{R}^n$. We provide a definition in terms of a basis of vectors.

Definition 1 *Given a matrix $A \in \mathbb{R}^{n \times n}$, the lattice generated by A is the set*

$$\mathcal{L}(A) = \{\mathbf{u}A : \ \mathbf{u} \in \mathbb{Z}^n\}. \tag{1}$$

In other words, the set $\mathcal{L}(A)$ corresponds to all the integer linear combinations of the rows of A. In particular, if $A \in \mathbb{Z}^{n \times n}$, we say that $\mathcal{L}(A)$ is an integral lattice. The matrix A is said to be a *basis* of the lattice. Like a vector space, a lattice admits infinitely many bases.

Definition 2 *A matrix U is unimodular if its entries are integers and $|\det(U)| = 1$.*

Definition 3 *Given two real matrices A, B, we say that $\mathcal{L}(A)$ is a sublattice of $\mathcal{L}(B)$ if there exists an integer matrix R such that $A = RB$.*

In particular, if R is a unimodular matrix, it can be proven that $\mathcal{L}(A) = \mathcal{L}(B)$. As a matter of fact, $\mathcal{L}(A) = \mathcal{L}(B)$ iff there exists a unimodular matrix U such that $A = UB$.

Definition 4 *A real matrix S is said to be an M-matrix if it can be written as $S = \gamma I + Q$, where $Q \leq 0$ and $\gamma \geq \rho(Q)$*[1].

A useful property of such M-matrices is known as *inverse-positivity*. We state this property in the form of a theorem. For the proof, [Berman and Plemmons 1987] can be consulted.

Theorem 1 *If $S = \gamma I + Q$ is an M-matrix with $\gamma > \rho(Q)$, then S^{-1} exists and $S^{-1} \geq 0$ (S is an inverse-positive matrix).*

The following definition is nonstandard, but it will be helpful, since it summarizes information on how an M-matrix was built.

Definition 5 *We say that S is an $M_{\gamma,\lambda}$-matrix, for positive integers γ, λ, if it is an invertible M-matrix and $S = \gamma I + Q$, with $Q_{i,j} \in \{-\lambda, -\lambda + 1, \cdots, -1, 0\}$.*

For completeness, we give three definitions regarding trapdoor functions, which is an essential concept in the design of public-key cryptosystems.

Definition 6 *A function f is a trapdoor-function if it can be efficiently evaluated in its input domain, but its inversion requires the knowledge of some special information, called the trapdoor.*

Definition 7 *An evaluation set for a trapdoor-function f is a set $\mathcal{E}$ of public parameters, required for the evaluation of f in its input domain. In this case, we use the notation $f_{\mathcal{E}}$ to represent the function f with evaluation set $\mathcal{E}$.*

Definition 8 *A trapdoor set for the inverse of a trapdoor-function f is a set $\mathcal{T}$ of secret parameters, required for the evaluation of f^{-1} in its input domain. In this case, we use the notation $f_{\mathcal{T}}^{-1}$ to represent the inverse function f^{-1} with trapdoor set $\mathcal{T}$.*

[1]Here, $\rho(Q)$ denotes the spectral radius of Q, i.e., $\rho(Q) = \max\{|\lambda_1|, \cdots, |\lambda_n|\}$, where $\lambda_1, \cdots, \lambda_n$ are the eigenvalues of Q.

3. The Problem of Lattice Deformations

Geometrically, a lattice may be viewed as a grid in the euclidean space, with the additive property: if two points are in the grid, their vectorial sum is also in the grid. In $\mathbb{R}^2$, a lattice is a 2D-grid, in $\mathbb{R}^3$ a 3D-grid and so on. The lattice basis forms what is called a *fundamental parallelepiped*, and the lattice itself consists of *copies* of this fundamental parallelepiped distributed along the euclidean space. We may think about the process of deforming such a grid, as shown in Figure 1.

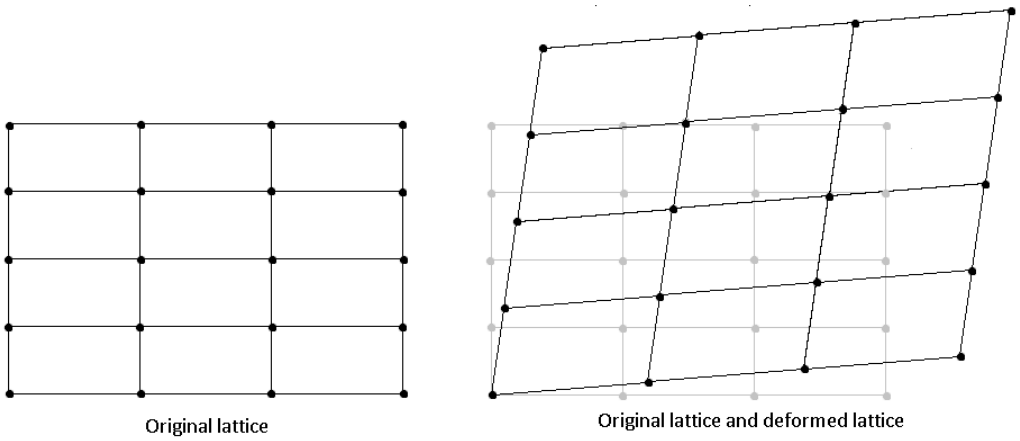

Figure 1. Lattice deformation.

This raises an interesting question: given a *deformed* grid, is it possible to recover the original grid? It seems to be clear that, if no information is given on how the original grid was deformed and what it looked like, it is quite impossible to know the answer to this question. The original grid could be *virtually any* grid.

Before we state this problem formally, we need to rigorously define what a *deformation* is, and what kind of information we aim to recover when trying to solve the problem. Let us denote by $\mathcal{I}_n$ the set of all $n \times n$ integer matrices, and consider three *nonempty* subsets $\mathcal{A}, \mathcal{B}, \mathcal{C} \subseteq \mathcal{I}_n$. Let $S \in \mathcal{A}$ be a lattice basis. A deformation on S consists of two steps:

1. Obtain a basis for a sublattice of $\mathcal{L}(S)$ given by RS (see Definition 3), where $R \in \mathcal{B}$;
2. Add a *deformation matrix* $E \in \mathcal{C}$, obtaining a basis $P = RS + E$.

The deformation matrix contains the vectors that will be added to the vectors of RS, in order to obtain the *deformed basis* P, as shown in Figure 2. We consider the problem of finding matrices $S' \in \mathcal{A}$ and $R' \in \mathcal{B}$ (the solution is not necessarily unique) such that $P = R'S' + E'$, with $E' \in \mathcal{C}$, or equivalently, such that $P - R'S' \in \mathcal{C}$.

Problem 1 (Lattice Deformation Problem (LDP)) *We take as parameters of the problem three subsets $\mathcal{A}, \mathcal{B}, \mathcal{C} \subseteq \mathcal{I}_n$. The input consists of a lattice basis P. The goal is to find $S \in \mathcal{A}$ and $R \in \mathcal{B}$ such that $P - RS \in \mathcal{C}$.*

We shall denote an instance of LDP with parameters $\mathcal{A}, \mathcal{B}, \mathcal{C}$ and input P by $\mathrm{LDP}^{\mathcal{A},\mathcal{B},\mathcal{C}}(P)$. A pair of integer matrices S, R is a solution for $\mathrm{LDP}^{\mathcal{A},\mathcal{B},\mathcal{C}}(P)$ if $S \in \mathcal{A}$, $R \in \mathcal{B}$ and $P - RS \in \mathcal{C}$. We say that P is a *viable input* if such a solution exists.

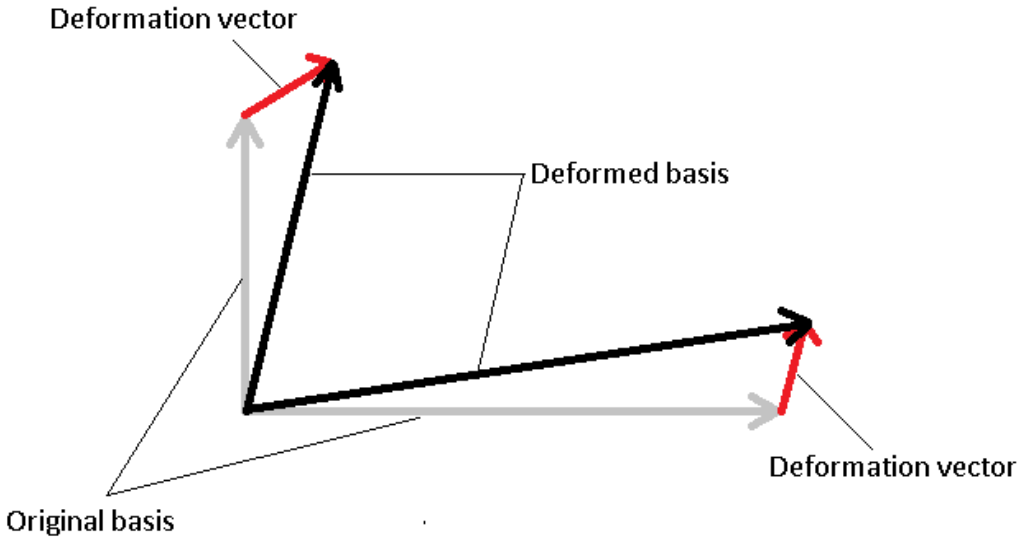

Figure 2. Deformation of a basis.

We are not aware of any specific method that can be used to solve this problem in the general case. It is clear that the hardness of an instance will depend on the given subsets $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$. These subsets are supposed to be sufficiently large, so that it is hard to perform an exhaustive search for a solution. On the other hand, large subsets may yield more than one solution, maybe an infinite number of solutions. The trivial case is $\mathcal{A} = \mathcal{B} = \mathcal{C} = \mathcal{I}_n$, for which any pair of integer matrices S, R is a valid solution (this remains true for any choice of $\mathcal{A}$ and $\mathcal{B}$ if we choose $\mathcal{C} = \mathcal{I}_n$).

In this paper, we propose a cryptosystem based on the assumed hardness of $\mathrm{LDP}^{\mathcal{A},\mathcal{B},\mathcal{C}}$, where $\mathcal{A}$ is the set of $n \times n$ M-matrices with given parameters, $\mathcal{B}$ is the set of n-dimensional unimodular matrices and $\mathcal{C}$ is the set of $n \times n$ integer matrices with entries bounded by given integers. For this purpose, we shall employ special notation for these subsets. The dimension n will be implicit in the notation:

- $\mathcal{M}_\gamma^\lambda$ will denote the set of all $n \times n$ $M_{\gamma,\lambda}$-matrices;
- $\mathcal{U}$ will denote the set of all n-dimensional unimodular matrices;
- $\mathcal{E}_\alpha^\beta$ will represent the set of all integer $n \times n$ matrices with entries chosen from the set $\{-\alpha, \cdots, 1, 0, 1, \cdots, \beta\}$, for given positive integers α, β.

Hence, we are considering instances of $\mathrm{LDP}^{\mathcal{A},\mathcal{B},\mathcal{C}}$ with $\mathcal{A} = \mathcal{M}_\gamma^\lambda$, $\mathcal{B} = \mathcal{U}$ and $\mathcal{C} = \mathcal{E}_\alpha^\beta$.

4. A New Trapdoor Function

In this section, we provide an overview of a candidate trapdoor function, whose security is based on the Lattice Deformation Problem. We follow the steps below:

- Pick up positive integers $n, \alpha, \beta, \gamma, \delta, \lambda, \sigma$ and positive real numbers ϵ_1, ϵ_2. The values of ϵ_1 and ϵ_2 must be close enough to 0. An heuristic choice for these parameters will be discussed in Section 7.
- Build a matrix $E \in \mathbb{Z}^{n \times n}$, with entries randomly and uniformly chosen from the set $\{-\alpha, -\alpha + 1, \cdots, 0, 1, \cdots, \beta\}$.
- Build a random unimodular matrix U as follows: choose a random upper triangular matrix T_u, a random lower triangular matrix T_l, both filled with 1's along the

main diagonal, and compute $U' = T_l T_u$. Next, choose two random permutation matrices R_1, R_2 and compute $U = R_1 U' R_2$. The entries of T_u and T_l may be chosen from the set $\{-\alpha, -\alpha+1, \cdots, 0, 1, \cdots, \beta\}$.

- Build an $M_{\gamma,\lambda}$-matrix $S = \gamma I + Q$, with $Q_{i,j} \in \{-\lambda, -\lambda+1, \cdots, -1, 0\}$. The inverse of S must satisfy the following conditions:

$$\frac{1}{\gamma} < S^{-1}_{j,j} < \frac{1+\epsilon_1}{\gamma}, \text{ for all } j = 1, \cdots, n \tag{2}$$

and

$$0 < S^{-1}_{i,j} < \frac{\lambda(1+\epsilon_2)}{\gamma^2}, \text{ for all } i \neq j. \tag{3}$$

- Build the matrix $P = US + E$.
- Choose real numbers $\theta_1, \theta_2, \mu_1, \mu_2$ satisfying

$$\theta_2 > \theta_1 > n\sigma\alpha > 0 \tag{4}$$

and

$$\mu_1 < \mu_2 < -n\sigma\beta < 0. \tag{5}$$

In order to build a suitable trapdoor for the inversion of our function, we need to choose these parameters satisfying some other conditions, which will be discussed later when we describe the inversion algorithm.

Let $\mathcal{E} = \{P, n, \sigma, \theta_1, \theta_2, \mu_1, \mu_2\}$ be the evaluation set of our function $f_{\mathcal{E}}$. Define a procedure `generateVector`, which takes as input $n, \theta_1, \theta_2, \mu_1, \mu_2$ and outputs a vector $\mathbf{r} \in \mathbb{Z}^n$. It generates, for all $k = 1, \cdots, n$, a random bit b_k. If $b_k = 0$, it chooses a random and uniform integer value from the interval $]\theta_1, \theta_2[$. If $b_k = 1$, an integer value is randomly and uniformly picked up from the interval $]\mu_1, \mu_2[$. The chosen value is assigned to the k-th entry of the vector $\mathbf{r}$.

Our function takes as input a vector $\mathbf{x} \in \mathbb{Z}^n$, with entries from the set $\{0, 1, \cdots, \sigma\}$, and outputs

$$f_{\mathcal{E}}(\mathbf{x}) = \mathbf{r} + \mathbf{x}P. \tag{6}$$

where $\mathbf{r}$ is the output of `generateVector`$(n, \theta_1, \theta_2, \mu_1, \mu_2)$.

The next step consists of establishing a proper inversion algorithm, which will make use of the trapdoor set $\mathcal{T} = \{U^{-1}, S^{-1}, \delta\}$. From now on, let $\mathbf{c} = f_{\mathcal{E}}(\mathbf{x}) = \mathbf{r} + \mathbf{x}P$, and define the *inversion error vector*

$$\mathbf{e} = (\mathbf{r} + \mathbf{x}E)S^{-1}. \tag{7}$$

Our first goal consists of determining which conditions, besides (4) and (5), the parameters $\theta_1, \theta_2, \mu_1, \mu_2$ must satisfy so that the following holds:

$$\theta_1 < \mathbf{r}_j < \theta_2 \Rightarrow \delta < \mathbf{e}_j < \delta + \frac{1}{2} \tag{8}$$

and

$$\mu_1 < \mathbf{r}_j < \mu_2 \Rightarrow -\delta - \frac{1}{2} < \mathbf{e}_j < -\delta, \tag{9}$$

for all $j = 1, \cdots, n$. In other words, we would like to guarantee that $\lceil \mathbf{e} \rfloor = [\pm\delta]^n$. Let $\mathbf{v} = \mathbf{r} + \mathbf{x}E$. Note that

$$\mathbf{v}_j = \mathbf{r}_j + \sum_{i=1}^{n} \mathbf{x}_i E_{i,j}, \tag{10}$$

for all $j = 1, \cdots, n$. Observe that, since we have $\theta_2 > \theta_1 > 0 > \mu_2 > \mu_1$, it is true that $\mu_1 < \mathbf{r}_j < \theta_2$ for all $j = 1, \cdots, n$. Provided that $0 \leq \mathbf{x}_j \leq \sigma$ and $-\alpha \leq E_{i,j} \leq \beta$, identity (10) allows us to conclude that

$$\mu_1 - n\sigma\alpha < \mathbf{v}_j < \theta_2 + n\sigma\beta. \tag{11}$$

Now we must consider the two cases below:

- Case 1: $\theta_1 < \mathbf{r}_j < \theta_2$
  From (4) and (10) we conclude that $\mathbf{v}_j > \theta_1 - n\sigma\alpha > 0$. By (7) we have

$$\mathbf{e}_j = \mathbf{v}_j S^{-1}_{j,j} + \sum_{i \neq j} \mathbf{v}_i S^{-1}_{i,j}. \tag{12}$$

  Since $S^{-1}_{i,j} > 0$ for all $1 \leq i, j \leq n$ and the lower bound for $\mathbf{v}_i$, for $i \neq j$, is a negative number, a lower bound for the sum above is given by

$$\mathbf{e}_j \geq \mathbf{v}_j S^{-1}_{j,j} + (n-1)\min_{i \neq j}\{\mathbf{v}_i\} \max_{i \neq j}\left\{S^{-1}_{i,j}\right\}. \tag{13}$$

  Using (2), (3) and (11), we conclude that

$$\mathbf{e}_j > \frac{\theta_1 - n\sigma\alpha}{\gamma} + \frac{\lambda(n-1)(\mu_1 - n\sigma\alpha)(1+\epsilon_2)}{\gamma^2}. \tag{14}$$

  In order to guarantee that $\mathbf{e}_j > \delta$, it is sufficient to set

$$\boxed{\theta_1 > \gamma\delta + n\sigma\alpha - \frac{\lambda(n-1)(\mu_1 - n\sigma\alpha)(1+\epsilon_2)}{\gamma}.} \tag{15}$$

  On the other hand, by (7) we have that

$$\mathbf{e}_j \leq \mathbf{v}_j S^{-1}_{j,j} + (n-1)\max_{i \neq j}\{\mathbf{v}_i\} \max_{i \neq j}\left\{S^{-1}_{i,j}\right\}. \tag{16}$$

  By (2), (3), (11) and (16), we get

$$\mathbf{e}_j < (\theta_2 + n\sigma\beta)\left(\frac{1+\epsilon_1}{\gamma} + \frac{\lambda(n-1)(1+\epsilon_2)}{\gamma^2}\right). \tag{17}$$

  Therefore, we obtain $\mathbf{e}_j < \delta + 1/2$ by setting

$$\boxed{\theta_2 < \frac{\gamma^2(2\delta+1)}{2(\gamma(1+\epsilon_1) + \lambda(n-1)(1+\epsilon_2))} - n\sigma\beta.} \tag{18}$$

  Hence, if θ_1 and θ_2 satisfy (4), (15) and (18), then (8) holds.

- Case 2: $\mu_1 < \mathbf{r}_j < \mu_2$
  By (5) and (10), we have $\mathbf{v}_j < \mu_2 + n\sigma\beta < 0$. From (16), we obtain

$$\mathbf{e}_j < \frac{\mu_2 + n\sigma\beta}{\gamma} + \frac{\lambda(n-1)(\theta_2 + n\sigma\beta)(1+\epsilon_2)}{\gamma^2}. \tag{19}$$

  Choosing the following upper bound for μ_2, we ensure that $\mathbf{e}_j < -\delta$:

$$\boxed{\mu_2 < -\gamma\delta - n\sigma\beta - \frac{\lambda(n-1)(\theta_2 + n\sigma\beta)(1+\epsilon_2)}{\gamma}.} \tag{20}$$

  Finally we have $\mathbf{v}_j > \mu_1 - n\sigma\alpha$. Once again, a lower bound for $\mathbf{e}_j$ is given by (13). Taking into account that $\mathbf{v}_j < 0$, we obtain

$$\mathbf{e}_j > (\mu_1 - n\sigma\alpha)\left(\frac{1+\epsilon_1}{\gamma} + \frac{\lambda(n-1)(1+\epsilon_2)}{\gamma^2}\right). \tag{21}$$

  In order to guarantee that $\mathbf{e}_j > -\delta - 1/2$, it is sufficient to choose μ_1 satisfying

$$\boxed{\mu_1 > \frac{-\gamma^2(2\delta+1)}{2(\gamma(1+\epsilon_1) + \lambda(n-1)(1+\epsilon_2))} + n\sigma\alpha.} \tag{22}$$

Conditions (4), (5), (15), (18), (20) and (22) ensure that, whenever $0 < \theta_1 < \mathbf{r}_j < \theta_2$, we have $\lceil \mathbf{e}_j \rfloor = \delta$. Similarly, when $\mu_1 < \mathbf{r}_j < \mu_2 < 0$, we have $\lceil \mathbf{e}_j \rfloor = -\delta$.

Now remind that $\mathbf{c} = f_{\mathcal{E}}(\mathbf{x}) = \mathbf{r} + \mathbf{x}P$. Define the *rounding difference vector* as follows:

$$\mathbf{d} = \mathbf{c}S^{-1} - \lceil \mathbf{c}S^{-1} \rfloor. \tag{23}$$

Since the bounds $\mu_1, \mu_2, \theta_1, \theta_2$ for the entries of $\mathbf{r}$ satisfy (4), (5), (15), (18), (20) and (22), the vector $\mathbf{d}$ allows us to decide whether $\lceil \mathbf{e}_j \rfloor = \delta$ or $\lceil \mathbf{e}_j \rfloor = -\delta$, even without prior knowledge of the vector $\mathbf{r}$. In fact,

$$\mathbf{d} = (\mathbf{r} + \mathbf{x}P)S^{-1} - \lceil (\mathbf{r} + \mathbf{x}P)S^{-1} \rfloor. \tag{24}$$

Replacing P by $US + E$, we obtain

$$\mathbf{d} = (\mathbf{r} + \mathbf{x}E)S^{-1} + \mathbf{x}U - \lceil (\mathbf{r} + \mathbf{x}E)S^{-1} + \mathbf{x}U \rfloor. \tag{25}$$

Since $\mathbf{x}U$ is an integer vector, we have that $\lceil (\mathbf{r}+\mathbf{x}E)S^{-1} + \mathbf{x}U \rfloor = \lceil (\mathbf{r}+\mathbf{x}E)S^{-1} \rfloor + \mathbf{x}U$. Hence:

$$\mathbf{d} = (\mathbf{r} + \mathbf{x}E)S^{-1} - \lceil (\mathbf{r} + \mathbf{x}E)S^{-1} \rfloor = \mathbf{e} - \lceil \mathbf{e} \rfloor. \tag{26}$$

We know that either $\delta < \mathbf{e}_j < \delta + 1/2$ or $-\delta - 1/2 < \mathbf{e}_j < -\delta$, for all $j = 1, \cdots, n$. In the first case, $\mathbf{e}_j > \lceil \mathbf{e}_j \rfloor$, while in the second we have $\mathbf{e}_j < \lceil \mathbf{e}_j \rfloor$. Hence, we conclude that, if $\mathbf{d}_j > 0$, then $\delta < \mathbf{e}_j < \delta + 1/2$, which means that $\lceil \mathbf{e}_j \rfloor = \delta$. Similarly, if $\mathbf{d}_j < 0$, we know for sure that $-\delta - 1/2 < \mathbf{e}_j < -\delta$ and, consequently, $\lceil \mathbf{e}_j \rfloor = -\delta$.

In face of these facts, we can build an inversion algorithm to our trapdoor function f, using the trapdoor set $\mathcal{T} = \{U^{-1}, S^{-1}, \delta\}$. It takes as input a vector $\mathbf{c}$, retrieves the rounded inversion error vector $\lceil \mathbf{e} \rfloor$ from the rounding difference vector, and outputs

$$f_{\mathcal{T}}^{-1}(\mathbf{c}) = (\lceil \mathbf{c}S^{-1} \rfloor - \lceil \mathbf{e} \rfloor)U^{-1}. \tag{27}$$

The correctness of $f_{\mathcal{T}}^{-1}$ can be easily proven. In fact, since the rounding difference vector $\mathbf{d}$ allows to correctly recover the rounded inversion error vector $\lceil \mathbf{e} \rfloor$, we have:

$$f_{\mathcal{T}}^{-1}(f_{\mathcal{E}}(\mathbf{x})) = (\lceil (\mathbf{r} + \mathbf{x}P)S^{-1} \rfloor - \lceil \mathbf{e} \rfloor)U^{-1} = (\lceil (\mathbf{r} + \mathbf{x}E)S^{-1} + \mathbf{x}U \rfloor - \lceil \mathbf{e} \rfloor)U^{-1}. \quad (28)$$

Once again, using the fact that $\mathbf{x}U$ is an integer vector, we may write the previous identity as follows:

$$f_{\mathcal{T}}^{-1}(f_{\mathcal{E}}(\mathbf{x})) = (\lceil (\mathbf{r} + \mathbf{x}E)S^{-1} \rfloor + \mathbf{x}U - \lceil \mathbf{e} \rfloor)U^{-1}. \quad (29)$$

From the definition of $\mathbf{e}$ given by (7), we finally obtain

$$f_{\mathcal{T}}^{-1}(f_{\mathcal{E}}(\mathbf{x})) = (\lceil (\mathbf{r} + \mathbf{x}E)S^{-1} \rfloor + \mathbf{x}U - \lceil (\mathbf{r} + \mathbf{x}E)S^{-1} \rfloor)U^{-1} = \mathbf{x}, \quad (30)$$

for all $\mathbf{x}$ in the domain of $f_{\mathcal{E}}$.

5. From a Trapdoor Function to a Cryptosystem

Now that we have a full description of our trapdoor function, it is time to use it to build a public-key cryptosystem. It is clear that the public key will consist of the evaluation set, and the secret key will be the trapdoor set. A full cryptosystem from our function requires procedures for key generation, encryption and decryption. The key generation procedures are the following:

- `MmatrixGenerator`: on input $n, \gamma, \lambda, \epsilon_1, \epsilon_2$, outputs an $M_{\gamma,\lambda}$-matrix S of dimension n, satisfying conditions (2) and (3).
- `unimodularGenerator`: on input n, α, β, outputs a random n-dimensional unimodular matrix U, as described in Section 4.
- `randomMatrixGenerator`: on input n, α, β, outputs an n-dimensional matrix E whose entries are uniformly chosen from the set $\{-\alpha, \cdots, \beta\}$.
- `parameterGenerator`: it takes as input the values of $n, \alpha, \beta, \gamma, \delta, \lambda, \sigma, \epsilon_1, \epsilon_2$ and outputs the parameters $\theta_1, \theta_2, \mu_1, \mu_2$ satisfying (4), (5), (15), (18), (22) and (20).

The public key, as already mentioned, consists of the evaluation set of our function, and the secret key is the trapdoor set. Denoting the public and secret keys respectively by `PK` and `SK`, we have `PK` = $\{P, n, \sigma, \theta_1, \theta_2, \mu_1, \mu_2\}$ and `SK` = $\{S^{-1}, U^{-1}, \delta\}$.

Encryption will be randomized. In other words, the same message encrypted twice is likely to generate different ciphertexts. For the encryptiom step, we need three algorithms:

- `messageEncoder`: takes as input a binary stream b and outputs $\mathbf{x} \in \mathbb{Z}^n$, with entries in $\{0, \cdots, \sigma\}$, or $\mathbf{r} \in \mathbb{Z}^n$, with entries in $]\mu_1, \mu_2[\cup]\theta_1, \theta_2[$, which encodes b.
- `ephemeralKeyGenerator`: if `messageEncoder` outputs $\mathbf{x}$, then this function outputs a random vector $\mathbf{r} \in \mathbb{Z}^n$, with entries in $]\mu_1, \mu_2[\cup]\theta_1, \theta_2[$. If `messageEncoder` outputs $\mathbf{r}$, then this function outputs a random vector $\mathbf{x} \in \mathbb{Z}^n$, with entries in $\{0, \cdots, \sigma\}$.
- `encryptionAlgorithm`: takes as input the vectors $\mathbf{x}$ and $\mathbf{r}$, generated by the previous procedures, and outputs $\mathbf{c} = \mathbf{r} + \mathbf{x}P$.

Note that the encryption resembles GGH, except that in our case the vector $\mathbf{x}P$ is not the vector of $\mathcal{L}(P)$ closest to $\mathbf{c}$. Furthermore, as in the GGH construction, the message can be encoded into $\mathbf{x}$ or $\mathbf{r}$.

The last part of our cryptosystem is the decryption procedure, which consists of three steps, performed by the following algorithms:

- `retrieveErrorVector`: builds the rounding difference vector $\mathbf{d}$ defined in (26) and reconstructs the rounded inversion error vector $\lceil \mathbf{e} \rfloor$ from (7) as follows: if $\mathbf{d}_j > 0$, then $\lceil \mathbf{e} \rfloor_j = \delta$, otherwise $\lceil \mathbf{e} \rfloor_j = -\delta$.
- `retrieveMessageVector`: computes $\mathbf{x} = (\lceil \mathbf{c}S^{-1} \rfloor - \lceil \mathbf{e} \rfloor)U^{-1}$. If the message was encoded into $\mathbf{r}$, the algorithm computes $\mathbf{c} - \mathbf{x}P$.
- `decodeMessage` decodes the vector $\mathbf{x}$ or $\mathbf{r}$ in order to obtain the original binary message.

Correctness of the decryption procedure comes immediately from the correctness of the inversion procedure for our trapdoor function.

Note that the secret key consists of three parts: S^{-1}, U^{-1} and δ. Apparently, neither of these parts isolated allows full decryption of a ciphertext. For example, if the value of δ leaks to an attacker, he still will not be able to decrypt any ciphertext. The same is true if one of the secret matrices leak. Hence, our cryptosystem seems to be suitable for scenarios of secret sharing, where the secret key must be distributed amongst a group of participants, in such a way that individual parts of the secret key are of no use on their own.

6. Security Analysis

In this section, we provide a security analysis of our proposal, describing the most common attacks and countermeasures to avoid them.

Before we proceed, let us precisely state the relation between breaking our cryptosystem and solving an instance of LDP. First, we make some remarks on the parameters $\gamma, \lambda, \alpha, \beta$, which are used to build the secret key. Since they are not necessary for encryption, there's no need to make them publicly available. However, since the possible choices for them must be publicly known, it is desirable to conceive a threat model in which the adversary has access to the values of these parameters.

Under the assumption that $\gamma, \lambda, \alpha, \beta$ are publicly known, we may establish a clear relation between our cryptosystem and LDP as follows:

1. Reduction from $\mathrm{LDP}^{\mathcal{M}^{\lambda}_{\gamma},\mathcal{U},\mathcal{E}^{\beta}_{\alpha}}$ to finding the secret matrices:
   (a) Let P be any viable input of $\mathrm{LDP}^{\mathcal{M}^{\lambda}_{\gamma},\mathcal{U},\mathcal{E}^{\beta}_{\alpha}}$. We map this input to a public matrix of our cryptosystem. In fact, P itself is a public matrix for some instance of a public key.
   (b) Assume the existence of an oracle that, given a public matrix and the parameters $\gamma, \lambda, \alpha, \beta$, returns the secret matrices S and U.
   (c) Feed the oracle with P and get the secret matrices S, U.
   (d) The pair S, U is a solution for $\mathrm{LDP}^{\mathcal{M}^{\lambda}_{\gamma},\mathcal{U},\mathcal{E}^{\beta}_{\alpha}}(P)$.
2. Reduction from finding the secret matrices to $\mathrm{LDP}^{\mathcal{M}^{\lambda}_{\gamma},\mathcal{U},\mathcal{E}^{\beta}_{\alpha}}$:

(a) Let P be a public matrix from a generic instance of a public key. We map P to a viable input of $\mathrm{LDP}^{\mathcal{M}_\gamma^\lambda,\mathcal{U},\mathcal{E}_\alpha^\beta}$. Because of the way P is built, it is also such a viable input.
(b) Assume the existence of an oracle that solves $\mathrm{LDP}^{\mathcal{M}_\gamma^\lambda,\mathcal{U},\mathcal{E}_\alpha^\beta}$, for any viable input.
(c) Feed the oracle with P and get a solution pair S, U.
(d) This solution pair serves as a pair of secret matrices. The parameter δ can be found by exhaustive search (since the space of all possible choices for δ will eventually be publicly known as well).

Hence, by assuming the hardness of $\mathrm{LDP}^{\mathcal{M}_\gamma^\lambda,\mathcal{U},\mathcal{E}_\alpha^\beta}$, we conclude that finding the secret matrices from the public matrix is hard. But some precautions must be taken in order to avoid other kinds of attacks.

6.1. Exhaustive Search

The first attack we would like to avoid consists of an exhaustive search on the key space, in which the attacker literally tries to *guess* the secret key. As we mentioned before, we work under the assumption that any adversary has access to the parameters $\gamma, \lambda, \alpha, \beta$. In order to avoid this kind of attack, these values must be chosen in such a way that the search space becomes sufficiently large.

If an attacker performs an exhaustive search for the secret matrices, he only needs to try guessing two matrices, for example U and S. The first alternative is trying to guess S and E and checking whether $(P - E)S^{-1}$ is a unimodular matrix.

For each γ, the search space for the matrix $S = \gamma I + Q$ is equal to the search space for the matrix Q, which is given by $(\lambda + 1)^{n^2}$. For the matrix E, the search space is given by $(\alpha+\beta+1)^{n^2}$. Assuming that all the matrices are uniformly and independently chosen, the total size of the search space (for the secret matrices) is given by $(\lambda+1)^{n^2}(\alpha+\beta+1)^{n^2}$.

If the attacker tries to guess U and S, he needs to check whether $P - US$ is a matrix with entries in the set $\{-\alpha, \cdots, \beta\}$. In order to compute the search space for the matrix U, we must take into account the search spaces for the matrices T_u, T_l, R_1, R_2, which are equal to $(\alpha + \beta + 1)^{n^2/2-n}$, $(\alpha + \beta + 1)^{n^2/2-n}$, $n!$ and $n!$ respectively. Hence, assuming that the choices are random and uniform, the total size of the search space is given by $(\alpha + \beta + 1)^{n^2-2n}(n!)^2$.

6.2. Babai's Algorithm

As we already mentioned, the encryption method of our cryptosystem is similar to GGH. Hence, one of the most natural types of attack consists of applying Babai's algorithm [Babai 1986] on the ciphertext $\mathbf{c} = \mathbf{r} + \mathbf{x}P$. The attack works as follows:

- The attacker applies some lattice reduction technique, such as LLL [Lenstra et al. 1982] or BKZ [Schnorr and Euchner 1994], on the public matrix P, obtaining a reduced basis $P' = U'P$, where U' is a unimodular matrix;
- The attacker computes $\lceil \mathbf{c}P'^{-1} \rfloor = \lceil \mathbf{r}P'^{-1} \rfloor + \mathbf{x}U'^{-1}$;
- If the vector $\mathbf{r}$ is sufficiently short, the attacker may expect that the vector above is equal to $\mathbf{x}U'^{-1}$. Multiplying by P', he obtains $\mathbf{x}P$ and solves a linear system to obtain $\mathbf{x}$.

This attack may work whenever $\mathbf{r}$ is a short vector, i.e., if $\mathbf{c}$ is sufficiently close to the lattice point $\mathbf{x}P$. In order to avoid this, the parameter δ must be sufficiently large. In our experiments, we set δ between 256 and 384, which was sufficient to avoid this kind of attack.

6.3. Shortening the Perturbation Vector

The previous attack does not work basically because the vector $\mathbf{r}$ is too large (if δ is sufficiently large), which means that the vector $\mathbf{c}$ is not close to $\mathbf{x}P$. However, an attacker may try to *shorten* the vector $\mathbf{r}$, in order to obtain a vector which is closer to $\mathbf{x}P$.

In fact, let $\mathbf{c} = \mathbf{r} + \mathbf{x}P$ be the ciphertext, and consider a vector $\mathbf{s}$, with entries in $]\mu_1, \mu_2[\cup]\theta_1, \theta_2[$, such that $\mathbf{r}_j\mathbf{s}_j > 0$ for all $j = 1, \cdots, n$. It is clear that the vector $\mathbf{r}' = \mathbf{r} - \mathbf{s}$ has much smaller entries than the vector $\mathbf{r}$. The attack consists of applying Babai's algorithm on the vector $\mathbf{c} - \mathbf{s}$, which is much closer to the vector $\mathbf{x}P$ than $\mathbf{c}$.

This attack will succeed if the positive (respectively negative) entries of $\mathbf{s}$ match the positive (respectively negative) entries of $\mathbf{r}$. Provided that the signal for each entry of $\mathbf{r}$ is uniformly chosen, we expect the number of positive entries to be close to the number of negative entries, which gives the attacker at least $n!/((n/2)!)^2$ possibilities to try.

We raise a natural question: how does the attacker know whether he chose the right vector $\mathbf{s}$? After applying Babai's algorithm on $\mathbf{c} - \mathbf{s}$, he expects to obtain the vector $\mathbf{x}$. Therefore, he may check whether his answer is a vector with integer entries in the interval $[0, \sigma]$. Our experiments suggest that this attack yields a unique valid solution. Curiously, if a random integer matrix is used instead of the unimodular U to build the public matrix P, this attack produces many valid solutions. The attacker has no way of knowing which one of them is the right one, because encryption is randomized. In this case, the security of the system would be based on the assumed hardness of $\mathrm{LDP}^{\mathcal{M}_\gamma^\lambda, \mathcal{I}_n, \mathcal{E}_\alpha^\beta}$.

7. Choice of Parameters

In this section, we discuss a possible choice of parameters to our cryptosystem, providing an analysis of the key size for this choice and the hardness of the aforementioned attacks. All values suggested here were tested using the NTL C++ Library [Shoup 2015], running on Linux Mint 17 (64 bits) with an AMD E-350 1600MHz processor and 4GB of RAM.

The choices for $\sigma, \delta, \gamma, \lambda, \alpha, \beta, \epsilon_1, \epsilon_2$ were heuristically determined in order to produce suitable values for $\mu_1, \mu_2, \theta_1, \theta_2$ and generate valid $M_{\gamma,\lambda}$-matrices satisfying conditions (2) and (3).

From now on, let $\mathrm{random}(x)$ be a function that outputs a random integer from the interval $[0, x[$. The suggested parameters are given below:

- $\sigma = 256$;
- $\delta = 256 + \mathrm{random}(128)$;
- $\gamma = n^5 + \mathrm{random}(n^3)$;
- $\lambda = n/2 + \mathrm{random}(n/2)$;
- $\alpha = \beta = n + \mathrm{random}(n)$.
- $\epsilon_1 = 10^{-6}$.
- $\epsilon_2 = 10^{-4}$.

The values of $\mu_1, \mu_2, \theta_1, \theta_2$ were computed as follows:

- $\mu_1 = \dfrac{-\gamma^2(2\delta+1)}{2(\gamma(1+\epsilon_1)+\lambda(n-1)(1+\epsilon_2))} + n\sigma\alpha + \text{random}(64).$
- $\theta_1 = \gamma\delta + n\sigma\alpha - \dfrac{\lambda(n-1)(\mu_1 - n\sigma\alpha)(1+\epsilon_2)}{\gamma} + \text{random}(64).$
- $\theta_2 = \dfrac{\gamma^2(2\delta+1)}{2(\gamma(1+\epsilon_1)+\lambda(n-1)(1+\epsilon_2))} - n\sigma\beta - \text{random}(64).$
- $\mu_2 = -\gamma\delta - n\sigma\beta - \dfrac{\lambda(n-1)(\theta_2 + n\sigma\beta)(1+\epsilon_2)}{\gamma} - \text{random}(64).$

We suggest the value of n to be between 128 and 256. Table 1 shows the public key size for some suggested values of n.

Table 1. Public key sizes (in kB) for several parameters.

Dimension (n)	Public Key Size
128	109
150	152
200	278
256	503

Table 2 shows the average time (in seconds) for key generation, encryption and decryption.

Table 2. Time (in seconds) for key generation, encryption and decryption.

Dimension (n)	Key generation	Encryption	Decryption
128	16.6617	0.0054	0.0441
150	31.3466	0.0078	0.0724
200	102.6070	0.0115	0.1729
256	285.3920	0.0191	0.3723

The minimum parameters yield a search space greater than $2^{7.2^{15}}$ for the exhaustive search attack. As we have already mentioned, the attack based on lattice reduction was unsuccessful for the suggested value of δ. For the third attack, the number of possibilities for the signals of the vector $\mathbf{s}$ is equal to 2^{128}. Considering that the signals are chosen uniformly, there are at least 2^{124} possibilities, corresponding to those in which the number of positive entries is equal to the number of negative entries.

For the maximum parameters ($n = 256$), the search space for the exhaustive search exceeds $2^{2^{20}}$. The number of possible choices for the vector $\mathbf{s}$ in the third attack is greater than 2^{251}. We estimate that these parameters offer high security against all known attacks.

8. Conclusion and Final Remarks

We proposed a new public-key cryptosystem from a new lattice problem, which we have reasons to believe is a hard one. It is not yet ideal in terms of key size, but current research suggests that this aspect can be substantially improved. Such improvements are part of our plans for future works, and further research is highly encouraged.

Future work must also include a formal proof of security for our cryptosystem, which means finding a suitable reduction from some classic hard problem to LDP, or even a clear relation between the security of our proposal and some other problem, for which the hardness has already been theoretically established. We also propose future study on the possibility of using random integer matrices instead of unimodular matrices to build the public key, as we briefly mentioned at the end of Section 6.3. This study would include an assessment on the hardness of $\mathrm{LDP}^{\mathcal{M}_\gamma^\lambda, \mathcal{I}_n, \mathcal{E}_\alpha^\beta}$.

Acknowledgements

We thank Professors Severino Collier Coutinho and Luziane Ferreira de Mendonça for helpful discussions, and the anonymous referees for useful comments and suggestions. We also thank CAPES and CNPq for the financial support.

References

Ajtai, M. and Dwork, C. (1997). A public-key cryptosystem with worst-case/average-case equivalence. In *Proc. 29th Annual ACM Symp. on Theory of Computing (STOC)*, pages 284–293.

Babai, L. (1986). On Lovász' lattice reduction and the nearest lattice point problem. volume 1 of *Combinatorica*, pages 1–13.

Berman, A. and Plemmons, R. J. (1987). *Nonnegative Matrices in the Mathematical Sciences*. Classics in Applied Mathematics. SIAM.

Goldreich, O. (1999). Private communication.

Goldreich, O., Goldwasser, S., and Halevi, S. (1997). Public-key cryptosystems from lattice reduction problems. In *Crypto'97, Lecture Notes in Computer Science*, volume 1294, pages 112–131.

Hoffstein, J., Pipher, J., and Silverman, J. H. (1998). NTRU: a ring based public-key cryptosystem. In *Proceedings of ANTS-III (LNCS)*, volume 1423, pages 267–288.

Lee, M. S. and Hahn, S. G. (2010). Cryptanalysis of the GGH cryptosystem. In *Mathematics in Computer Science*, volume 3, pages 201–208.

Lenstra, A. K., Lenstra, H. W., and Lovász, L. (1982). Factoring polynomials with rational coefficients. In *Mathematische Annalen*, volume 261, pages 515–534.

Lyubashevsky, V., Peikert, C., and Regev, O. (2013). On ideal lattices and learning with errors over rings. *Journal of the ACM*, 60(6):1–43.

Micciancio, D. (2001). Improving lattice based cryptosystems using the Hermite Normal Form. In *CaLC - Lecture Notes in Computer Science*, volume 2146, pages 126–145.

Nguyen, P. Q. (1999). Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from Crypto'97. In *Crypto'99, Lecture Notes in Computer Science*, volume 1666, pages 288–304.

Regev, O. (2005). On lattices, learning with errors, random linear codes, and cryptography. In *Proc. 37th ACM Symp. on Theory of Computing (STOC)*, pages 84–93.

Schnorr, C. P. and Euchner, M. (1994). Lattice basis reduction: improved practical algorithms and solving subset sum problems. In *Math. Programming*, volume 66, pages 181–199.

Shoup, V. (2015). Number Theory C++ Library (NTL) version 9.0.2. Available at `http://www.shoup.net/ntl/`.

Yoshino, M. and Kunihiro, N. (2012). Improving GGH cryptosystem for large error vector. In *International Symposium on Information Theory and its Applications*, pages 416–420.

