University of Zurich[UZH]

Institute of Mathematics

Master's Thesis

The Ajtai-Dwork Cryptosystem and Other Cryptosystems Based on Lattices

***Author*:**
Michael Hartmann

***Supervisor*:**
Prof. Dr. Joachim Rosenthal

October 29, 2015

Abstract

Since the publication of the seminal result of Miklós Ajtai presented in *Generating Hard Instances of Lattice Problems* lattice based cryptosystems have become a very attractive field in cryptography. For the first time it was possible to base the security of a cryptosystem on the worst-case assumptions rather than on average-case assumption, which is essential for doing cryptography. Another important advantage of cryptosystems based on lattice problems over cryptosystem based on factoring or discrete logarithm is the lack of quantum algorithms solving these problems with nameable computational advantage over classical algorithms.

The goal of this thesis is to compare four cryptosystems based on lattices and to analyze their practicality with today's computational power as well as analyze the feasibility of their known attacks with today's implementations of LLL and BKZ.

Acknowledgements

First and foremost I offer my sincerest gratitude to my supervisor, Prof. Dr. Joachim Rosenthal, for giving me the opportunity to write my Master's thesis while already supervising many other students. I also thank Dr. Urs Wagner for recommending me this interesting topic. A warm thank also goes to Dr. Joop van de Pol who responded promptly to a implementation question via email and to Carsten Rose giving me access to one of the server of the Mathematical Institute and a short introduction on how to use it. I owe many thanks to my friends Kevin and Ben for their time for proofreading and their valuable input. Last but not least I express my thanks to my parents for supporting me a lot while my studies and to my partner Nina, who mental supported me as well as for her understanding and patience.

Contents

List of Figures

Chapter 1

Introduction

Imagine a day in the future, sitting in the train, grabbing a newspaper and reading the headline: "Internet No Longer Secure - New Quantum Computer Able to Break Mostly Used Communication Encryption". Such a headline can unsettle many people and raise questions about security, considering that a large percentage of our communication takes place on the internet. The number of people buying products, accessing sensitive data as bank accounts or insurance data or simply revealing their GPS position periodically is increasing. Such a scenario can turn to reality, since the de facto standard for encrypting data traffic is based on finite field problems such as factoring and discrete logarithm. In 1997, Shor [Sho97] developed polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. This poses the question of the technical possibility of building such a quantum computer. In 2014, the largest number, which could be factored on a quantum computer was 56'153 [DB14], improving the previous record of 143 that was set in 2012 [XZL$^+$12]. In our opinion, it is only a matter of time until quantum computers are capable to break today's cryptosystems, but even if this was totally unrealistic it is undoubtedly desirable to have a big variety of mathematical problems to rely on.

One of the main goals in cryptography is to find a way of generating a hard problem together with a solution to it. The security provided by the use of a specific encryption is dependent on its average-case hardness, rather than the effort it would take to solve the worst-case. The hardness which is desired is either a proof, that the minimum steps needed for finding the solution is a very high number and so infeasible for an attacker to break, or a problem which is NP-complete.

The shortest vector problem and its inhomogeneous counterpart are promising problems in the field of lattices to base trapdoor one-way functions on them. At this time, no quantum algorithm is known that solves the mentioned problems with a nameable advantage. These problems became even more relevant when Ajtai presented his famous discovery, a

fascinating connection between the worst-case complexity and the average-case complexity of a variant of the shortest vector problem. In *Generating Hard Instances of Lattice Problems* [Ajt96] Ajtai demonstrated a way to generate a random instance of a lattice together with a shortest vector in it, where finding this shortest vector is as hard as solving other related problems in the worst-case. Later, Ajtai and Dwork [AD97] presented public-key cryptosystem with worst-case/average-case equivalence. Even if the cryptosystem was not very practical, this discovery encouraged many other scientists to study the field of lattice, and new trapdoor one-way functions were published [GGH97b, Mic01b, HPS98].
In this thesis we examine besides the Ajtai-Dwork cryptosystem, three other cryptosystems based on lattices. Our goal is to implement them together with known attacks and run them on today's computer and with modern implementations of the underlying algorithms. These implementations should also serve as a foundation to run experiments in order to analyze and compare their practicality today.
The structure of this thesis is as follows. After an introduction to lattices, the definitions and results that were used in this work in Section 2, we give an overview of the cryptosystem by Ajtai and Dwork in Section 3, followed by a Section where we show a heuristic attack presented by Nguyen and Stern. Section 5 describes the GGH cryptosystem together with four different attacks, followed by a very related cryptosystem by Micciancio in Section 6. In Section 7 we take a closer look at a cryptosystem, called NTRU, which finds application in today's devices. Finally, in Section 8 we compare the results of the experiments and summarize the facts.

Practical Tests

For every cryptosystem covered in this thesis, we provide some practical tests. All these experiments are implemented in Sage Mathematics Software System 6.1.1 [S$^+$14] and are available under `https://bitbucket.org/MichaelHartmann/lattice-based-cryptosystems`.
All tests were performed on the same machine, a system with two 64-bit CPU's (8-core Intel Xeon 4C E5-2643 3.30 GHz), 10MB cache and 128GB of ram running ubuntu 14.04 LTS. Every test ran only on one core to give more accurate results to compare.
Sage uses many functions of the high-performance C++ Number Theory Library (NTL) for polynomial rings and lattice basis reduction. This library is free available under [Sho14]. Both, Sage and NTL, are under GNU General Public License [GPL07].

Chapter 2

Preliminaries

Notation 1. To clearly differentiate between variables for numbers and vectors, we write numbers in minuscule letter in normal font, e.g. $i \in \mathbb{Z}$ and the vectors in bold minuscule letter, e.g. $\boldsymbol{w_1} \in \mathbb{R}^n$ (not to be confused with w_1 which denotes the first element of the vector $\boldsymbol{w}$). The matrices are written in bold capital letters as $\boldsymbol{W} \in \mathbb{R}^{n \times m}$.

Let $\mathbb{R}$ be the set of real numbers and $\mathbb{Z}$ the set of integer numbers. Further, let $\mathbb{R}^m$ be the real vector space of dimension m. For any ring R, we denote $R^\times$ the unit group of R. For $\boldsymbol{w_1}, \ldots, \boldsymbol{w_k} \in \mathbb{R}^m$ arbitrary vectors, define $\mathbf{W} := [\boldsymbol{w_1}, \ldots, \boldsymbol{w_k}] \in \mathbb{R}^{m \times k}$, the matrix with column vectors $\boldsymbol{w_1}, \ldots, \boldsymbol{w_k}$. The general linear group of degree n over $\mathbb{Z}$, i.e. the set of invertible $n \times n$ matrices over the integers, is denoted by $GL_n(\mathbb{Z})$. The L_2-norm is denoted by $\|\cdot\|$, e.g. $\|\boldsymbol{w}\| := \sqrt{w_1^2 + \cdots + w_n^2}$, where $\boldsymbol{w} = (w_1, \ldots, w_n) \in \mathbb{R}^n$.

Definition 1. The additive subgroup generated by $\boldsymbol{b_1}, \ldots, \boldsymbol{b_k}$ is given by

$$\Lambda := \Lambda(\boldsymbol{b_1}, \ldots, \boldsymbol{b_k}) := \left\{ \sum_{i=1}^{k} \lambda_i \cdot \boldsymbol{b_i} \,\middle|\, \lambda_i \in \mathbb{Z} \, \forall i \in \{1, \ldots, k\} \right\} \subset \mathbb{R}^m$$

If $\boldsymbol{b_1}, \ldots, \boldsymbol{b_k}$ are linearly independent we call $\Lambda(\boldsymbol{b_1}, \ldots, \boldsymbol{b_k})$ a **lattice** of **rank** k and **dimension** m and further, we call $\boldsymbol{b_1}, \ldots, \boldsymbol{b_k}$, respectively $\mathbf{B}$ a **basis** of Λ.
In the literature, the notions rank and dimension are often used interchangeably. To prevent confusion, we use the term dimension only for the dimension of the vector space the lattice is embedded in.
The **length** of a basis $\mathbf{B}$ is the maximum over all L_2-norms of the basis vectors, so

$$\max_{\boldsymbol{b} \in \mathbf{B}} \|\boldsymbol{b}\|$$

Definition 2. The linear space spanned by Λ is defined as follows:

$$\mathbf{span}\,(\Lambda) := \left\{ \sum_{i=1}^{k} \lambda_i \cdot \boldsymbol{b_i} \,\middle|\, \lambda_i \in \mathbb{R} \, \forall i \in \{1, \ldots, k\} \right\}$$

The general linear group $GL_n(\mathbb{F})$ is a group for every field $\mathbb{F}$. We prove in the following lemma, that this is also true for the ring of integers.

Lemma 1. $GL_n(\mathbb{Z})$ is a subgroup of $GL_n(\mathbb{R})$.

Proof. Clearly, $GL_n(\mathbb{Z}) \subset GL_n(\mathbb{R})$. So, it is enough to show, that $GL_n(\mathbb{Z})$ forms a group. $\mathbf{I_n}$ is clearly the neutral element of $GL_n(\mathbb{Z})$ and $GL_n(\mathbb{R})$. So the only thing remaining to prove is that $\mathbf{A} \cdot \mathbf{B^{-1}} \in GL_n(\mathbb{Z})$ for $\mathbf{A}, \mathbf{B} \in GL_n(\mathbb{Z})$. We first prove that $\mathbf{B^{-1}}$ is an integer matrix: For the inverse we have the following formula:

$$\mathbf{B^{-1}} = \frac{1}{\det \mathbf{B}} adj(\mathbf{B}),$$

where $adj(\mathbf{B})$ is the adjugate matrix of $\mathbf{B}$. Since, $\det \mathbf{B} = \pm 1$, we get

$$\mathbf{B^{-1}} = \pm adj(\mathbf{B}).$$

By definition, $adj(\mathbf{B})$ consists of of determinant values of minors of $\mathbf{B}$, which are all integers, since the determinant of an integer matrix is it self an integer (see definition of the determinant, e.g. by Leibniz).
Therefore $\mathbf{B^{-1}}$ is an integer matrix. It is unimodular because:

$$1 = \det \mathbf{I_n} = \det(\mathbf{B}\mathbf{B^{-1}}) = \det \mathbf{B} \cdot \det \mathbf{B^{-1}} = (\pm 1) \cdot \det \mathbf{B^{-1}}$$

So, $\mathbf{B^{-1}}$ is an unimodular integer matrix, i.e. $\mathbf{B^{-1}} \in GL_n(\mathbb{Z})$, and so is $\mathbf{A}\mathbf{B^{-1}}$, because it is an integer matrix and

$$\det(\mathbf{A}\mathbf{B^{-1}}) = \det \mathbf{A} \cdot \det \mathbf{B^{-1}} = (\pm 1) \cdot (\pm 1)$$

Therefore, $GL_n(\mathbb{Z})$ is a subgroup of $GL_n(\mathbb{R})$. □

Proposition 2. Let $\mathbf{M}, \mathbf{M'} \in \mathbb{R}^{n \times k}$ be two matrices with linearly independent columns. $\mathbf{M}$ and $\mathbf{M'}$ generate the same lattice $\iff \exists \mathbf{U} \in GL_k(\mathbb{Z}) : \mathbf{M} = \mathbf{U}\mathbf{M'}$

Proof. "$\Rightarrow$": Let $\Lambda(\mathbf{M}) = \Lambda(\mathbf{M'})$. Each column of $\mathbf{M}$ is in $\Lambda(\mathbf{M'})$, so we can write $\mathbf{M} = \mathbf{U}\mathbf{M'}$ for some $\mathbf{U} \in \mathbb{Z}^{k \times k}$ (by definition of lattice). With the same reasoning we can write $\mathbf{M'} = \mathbf{U'}\mathbf{M}$ for some $\mathbf{U'} \in \mathbb{Z}^{k \times k}$. Combining these two equations, we get:

$$\mathbf{M} = \mathbf{U}\mathbf{M'} = \mathbf{U}\mathbf{U'}\mathbf{M}$$

or equivalently,

$$\mathbf{0} = (\mathbf{U}\mathbf{U'} - \mathbf{I_k})\mathbf{M}$$

Since the columns in $\mathbf{M}$ are linearly independent, it must be $\mathbf{0} = (\mathbf{U}\mathbf{U'} - \mathbf{I_k})$, and we get $\mathbf{U}\mathbf{U'} = \mathbf{I_k}$ and since $\mathbf{U}, \mathbf{U'} \in \mathbb{Z}^{k \times k}$, $\det(\mathbf{U})$ and $\det(\mathbf{U'})$ must be integers. In particular,

$$\det(\mathbf{U}) \det(\mathbf{U'}) = \det(\mathbf{U}\mathbf{U'}) = \det(\mathbf{I_k}) = 1,$$

$$
\begin{array}{c}
\\ \\ \\ \mu \\ \\ \\ \\ \nu \\ \\ \\ \\
\end{array}
\begin{pmatrix}
1 & & & & & & & & & & & \\
 & \ddots & & & & & & & & & & \\
 & & 1 & & & & & & & & & \\
 & & & 0 & & & & 1 & & & & \\
 & & & & 1 & & & & & & & \\
 & & & & & \ddots & & & & & & \\
 & & & & & & 1 & & & & & \\
 & & & 1 & & & & 0 & & & & \\
 & & & & & & & & 1 & & & \\
 & & & & & & & & & \ddots & & \\
 & & & & & & & & & & & 1
\end{pmatrix}
\qquad
\begin{array}{c}
\mu \\
\begin{pmatrix}
1 & & & & & & \\
 & \ddots & & & & & \\
 & & 1 & & & & \\
 & & & -1 & & & \\
 & & & & 1 & & \\
 & & & & & \ddots & \\
 & & & & & & 1
\end{pmatrix}
\end{array}
\qquad
\begin{array}{cc}
 & \mu \\
\nu & \begin{pmatrix}
1 & & & & & \\
 & \ddots & & & & \\
 & & \ddots & & & \\
 & & & \lambda & & \\
 & & & & \ddots & \\
 & & & & & 1
\end{pmatrix}
\end{array}
$$

Figure 2.1: Unitary Matrices Corresponding to the Operations in Corollary 3

so $\det(\mathbf{U}) = \det(\mathbf{U}') \in \{\pm 1\} = \mathbb{Z}^{\times}$. Hence $\mathbf{U} \in GL_k(\mathbb{Z})$.
"$\Leftarrow$": Let $\mathbf{M} = \mathbf{U}\mathbf{M}'$ for some unimodular matrix $\mathbf{U} \in GL_k(\mathbb{Z})$. Since $\mathbf{U} \in GL_k(\mathbb{Z})$ it follows that $\mathbf{U}^{-1}$ exists and it lies also in $GL_k(\mathbb{Z})$ (Lemma 1). So,

$$\mathbf{M}' = \mathbf{U}^{-1}\mathbf{M}, \text{ where } \mathbf{U}^{-1} \in \mathbb{Z}^{k \times k}$$

Hence $\Lambda(\mathbf{M}') \subset \Lambda(\mathbf{M})$ and $\Lambda(\mathbf{M}) \subset \Lambda(\mathbf{M}')$, i.e. $\Lambda(\mathbf{M}') = \Lambda(\mathbf{M})$. □

Corollary 3. The following operations on a lattice basis $\mathbf{B}$ do not change the lattice.

1. Switching two rows.
2. Multiply a row by -1.
3. Adding an integer multiple of a row to another.

Proof. We show that all these operations correspond to a left-multiplication of the k-dimensional basis matrix $\mathbf{B}$ by an unimodular matrix $\mathbf{U} \in GL_k(\mathbb{Z})$ and therefore do not change the lattice (see Proposition 2). Figure 2.1 shows these matrices in the same order as in the claim.

1. Switching the μ-th and ν-th row in $\mathbf{B}$ corresponds to left-multiplying $\mathbf{B}$ by the matrix $\mathbf{I}_{\mu\nu}$, which denotes the identity matrix $\mathbf{I}$ with interchanged μ-th and ν-th row for $\mu \neq \nu$. By definition of determinant one can see, that $\det(\mathbf{I}_{\mu\nu})$ = -1.
2. Multiply the μ-th row in $\mathbf{B}$ by -1 corresponds to multiply $\mathbf{B}$ from the left by the identity matrix $\mathbf{I}$ having a -1 instead of a 1 in the μ-th row. This modified identity matrix has obviously determinant -1, and is therefore unimodular in $\mathbb{Z}$.
3. Adding an integer multiple $\lambda \in \mathbb{Z}$ of the μ-th to the ν-th row ($\mu \neq \nu$), corresponds to left-multiplying $\mathbf{B}$ by the identity matrix $\mathbf{I}$ with λ instead of 0 on the μ-th position

in the ν-th row. By Laplace expansion one can easily see, that its determinant can be computed as follows:

$$1 \cdot \det(\mathbf{I_{k-1}}) + \lambda \cdot \det \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ 0 & & & \dots & & & 0 \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix} = 1 + 0 = 1$$

All the other sub-matrices have also determinant 0.

□

Definition 3. The **parallelepiped** spanned by the $\boldsymbol{b_i}$'s is defined as

$$\mathcal{P}(\mathbf{B}) := \left\{ \sum_{i=1}^{k} r_i \cdot \boldsymbol{b_i} \,\middle|\, r_i \in [0,1) \,\forall i \in \{1,\dots,n\} \right\} \subset \mathbf{span}\,(\Lambda)$$

Definition 4. Let $\Lambda \subset \mathbb{R}^n$ be a lattice with rank d and $\mathbf{B} = (\boldsymbol{b_1}, \dots, \boldsymbol{b_d})$ a basis of Λ, then the volume of Λ is given by:

$$vol(\Lambda) := \sqrt{\det(\mathbf{B}^T\mathbf{B})}$$

If $\mathbf{B}$ is non-singular, $vol(\Lambda)$ is equal to $|\det \mathbf{B}|$.

Remark. While the parallelepiped is basis-dependent, the volume is unique for the lattice. If we have two matrices $\mathbf{B}_1$ and $\mathbf{B}_2$ with $\Lambda(\mathbf{B}_1) = \Lambda(\mathbf{B}_2)$, then there exists a $\mathbf{U} \in GL_k(\mathbb{Z})$ such that $\mathbf{B}_1 = \mathbf{U}\mathbf{B}_2$ (Proposition 2), and therefore

$$\begin{aligned} vol(\Lambda) &= \sqrt{\det(\mathbf{B}_1^T\mathbf{B}_1)} \\ &= \sqrt{\det((\mathbf{U}\mathbf{B}_2)^T(\mathbf{U}\mathbf{B}_2))} \\ &= \sqrt{\det(\mathbf{B}_2^T\mathbf{U}^T\mathbf{U}\mathbf{B}_2)} \\ &= \sqrt{\det(\mathbf{B}_2^T)\det(\mathbf{U}^T\mathbf{U})\det(\mathbf{B}_2)} \\ &= \sqrt{\det(\mathbf{B}_2^T)\det(\mathbf{B}_2)} \\ &= \sqrt{\det(\mathbf{B}_2^T\mathbf{B}_2)} \end{aligned}$$

Definition 5. Let Λ be a lattice with parallelepiped $\mathcal{P}(\boldsymbol{b_1}, \dots \boldsymbol{b_n})$. Define the **width vector** $\boldsymbol{d} := (d_1, \dots, d_n)$, where d_i is the Euclidean distance between $\boldsymbol{b_i}$ and the hyperplane generated by $\{\boldsymbol{b_1}, \dots, \boldsymbol{b_n}\} \setminus \boldsymbol{b_i}$.
Further, we call

$$width\,(\mathcal{P}(\boldsymbol{b_1}, \dots \boldsymbol{b_n})) := \min\{d_1, \dots, d_n\} \tag{2.1}$$

the **width** of the parallelepiped $\mathcal{P}(\boldsymbol{b_1}, \dots \boldsymbol{b_n})$.

Remark. Let $\Lambda := \Lambda(\boldsymbol{b_1}, \dots, \boldsymbol{b_k})$ be a lattice of rank k. Then $\mathbf{span}\,(\Lambda)$ is equal to the disjoint union of the parallelepiped $\mathcal{P}(\Lambda)$ shifted by lattice points $\boldsymbol{u}$:

$$\mathbf{span}\,(\Lambda) = \bigcup_{\boldsymbol{u} \in \Lambda} \boldsymbol{u} + \mathcal{P}(\Lambda)$$

For $\boldsymbol{u} \in \Lambda$ the only lattice point in $\boldsymbol{u} + \mathcal{P}(\Lambda)$ is $\boldsymbol{u}$ itself. This allows us to define a reduce operation on the lattice Λ:

Definition 6. For any point $\boldsymbol{r} := \boldsymbol{u} + \boldsymbol{p} \in \mathbf{span}\,(\Lambda)$, where $\boldsymbol{u} \in \Lambda$ and $\boldsymbol{p} \in \mathcal{P}(\Lambda)$ we define

$$\boldsymbol{p} \equiv \boldsymbol{r} \mod \mathcal{P}(\Lambda)$$

as the **reduction** of $\boldsymbol{r}$ modulo the parallelepiped $\mathcal{P}(\Lambda)$.
More generally, we define an equivalence relation for $\boldsymbol{a}, \boldsymbol{b} \in \mathbf{span}\,(\Lambda)$:

$$\boldsymbol{a} \equiv \boldsymbol{b} \iff \boldsymbol{a} - \boldsymbol{b} \in \Lambda$$

Example 1. Figure 2.2 shows a lattice generated by $\boldsymbol{b_1} = (2, 3)$ and $\boldsymbol{b_2} = (5, 1)$. The black dots demonstrate the lattice points. The shaded area illustrates the parallelepiped $\mathcal{P}(\mathbf{B}) = \mathcal{P}(\boldsymbol{b_1}, \boldsymbol{b_2})$. The gray dots show the points equivalent to

$$\boldsymbol{v} = 3 \cdot \boldsymbol{b_1} - 1 \cdot \boldsymbol{b_2} + (2, 2)$$

and $\boldsymbol{v}'$ is the unique point in the parallelepiped equivalent to $\boldsymbol{v}$:

$$(2, 2) = \boldsymbol{v}' \equiv \boldsymbol{v} \mod \mathcal{P}(\mathbf{B})$$

Definition 7. The **dual** lattice of a lattice $\Lambda := \Lambda(\boldsymbol{b_1}, \dots, \boldsymbol{b_n})$ is defined as

$$\Lambda^\star := \left\{\boldsymbol{x} \in \mathbf{span}\,(\Lambda) \,\middle|\, \boldsymbol{x}^T \boldsymbol{y} \in \mathbb{Z}\, \forall \boldsymbol{y} \in \Lambda\right\}$$

or equivalently as,

$$\Lambda^\star = \left\{\boldsymbol{x} \in \mathbf{span}\,(\Lambda) \,\middle|\, \boldsymbol{x}^T \boldsymbol{b_i} \in \mathbb{Z}\, \forall \boldsymbol{b_i} \in \mathbf{B}\right\}$$

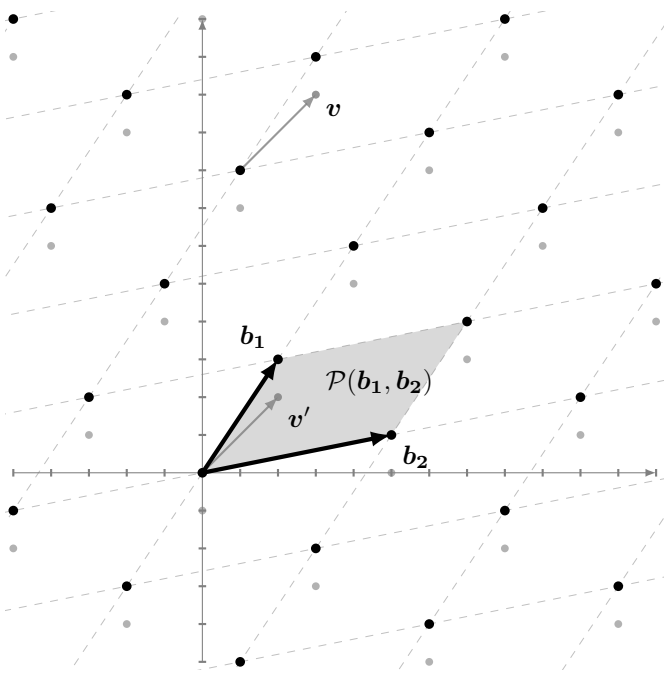

Figure 2.2: Lattice Generated by $\boldsymbol{b_1} = (2, 3)$ and $\boldsymbol{b_2} = (5, 1)$.

Remark. If $(\boldsymbol{b_1}, \ldots, \boldsymbol{b_n})$ is a basis for a lattice Λ, then $(\boldsymbol{c_1}, \ldots, \boldsymbol{c_n})$ is a basis for $\Lambda^\star$, where

$$\boldsymbol{c_i}^T \boldsymbol{b_j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j. \end{cases}$$

Definition 8. Let Λ be a lattice. The **orthogonal** lattice $\Lambda^\perp$ is defined as the set of points in $\mathbb{Z}^n$ that are orthogonal to all the lattice points, i.e.

$$\Lambda^\perp := \{\boldsymbol{z} \in \mathbb{Z}^n \mid \langle \boldsymbol{v}, \boldsymbol{z} \rangle = 0 \, \forall \boldsymbol{v} \in \Lambda\}$$

Theorem 4. Let $\mathbf{B} \in \mathbb{Z}^{n \times n}$ be non-singular and $d := vol(\mathbf{B})$, then $d \cdot \mathbb{Z}^n \subset \Lambda(\mathbf{B})$.

Proof. Let $\boldsymbol{v} \in d \cdot \mathbb{Z}^n$, i.e. $\boldsymbol{v} = d \cdot \boldsymbol{y}$ for some vector $\boldsymbol{y} \in \mathbb{Z}^n$. We want to show that $\exists \boldsymbol{x} \in \Lambda(\mathbf{B}) : \boldsymbol{v} = \mathbf{B}\boldsymbol{x}$. Since $\mathbf{B}$ is non-singular, such an $\boldsymbol{x}$ exists always in $\mathbb{R}^n$. We now show, that $\boldsymbol{x}$ is in fact a integer vector. Consider $d \cdot \boldsymbol{y} = \mathbf{B}\boldsymbol{x}$, which can be solved using Cramer's rule:

$$\begin{aligned} x_i &= \frac{\det\left([\boldsymbol{b_1}, \ldots, \boldsymbol{b_{i-1}}, d\boldsymbol{y}, \boldsymbol{b_{i+1}}, \ldots, \boldsymbol{b_n}]\right)}{\det \mathbf{B}} \\ &= \frac{d \cdot \det\left([\boldsymbol{b_1}, \ldots, \boldsymbol{b_{i-1}}, \boldsymbol{y}, \boldsymbol{b_{i+1}}, \ldots, \boldsymbol{b_n}]\right)}{\det \mathbf{B}} \\ &= \pm \det\left([\boldsymbol{b_1}, \ldots, \boldsymbol{b_{i-1}}, \boldsymbol{y}, \boldsymbol{b_{i+1}}, \ldots, \boldsymbol{b_n}]\right) \in \mathbb{Z} \end{aligned}$$

□

Remark. Every lattice $\Lambda := \Lambda(\mathbf{B})$ is periodic in $d\mathbb{Z}^n$ for $d := vol(\Lambda)$. This defines an equivalence relation $\boldsymbol{u} \equiv \boldsymbol{v} :\Leftrightarrow \boldsymbol{u} - \boldsymbol{v} \in d\mathbb{Z}^n$ and thus $\boldsymbol{u} \in \Lambda(\mathbf{B}) \Leftrightarrow \boldsymbol{v} \in \Lambda(\mathbf{B})$.

2.1 Public Key Cryptography

Until the publication of Diffie and Hellman in 1976 [DH76] all known cryptosystems were **symmetric**, which means that the sender and the receiver of a message use both the same key for encryption and decryption. This kind of cryptosystems are very impractical in large networks, since every pair of nodes uses their own keys which leads to a difficult key management. Another drawback is, that these keys have to be exchanged in secret.
In contrast to symmetric key cryptography, public key cryptography (or **asymmetric** key cryptography) works with only two keys for each participant, namely a private and a public key. Typically every participant initially creates these two keys. Consider two participants, commonly named Alice and Bob. Alice wants to send a secret message to the receiver Bob over an insecure channel. For that purpose, Alice encrypts the message with the public key of Bob and sends the ciphertext to him. Bob then uses his private key to decrypt the message with his private key. Contrary to symmetric key cryptography, it should not be feasible to decrypt the message with the key used for encrypting. The attacker will be called Eve.
In practice encrypting with an asymmetric key system is extremely slower than encrypting with a symmetric key system. That's why asymmetric key systems are often used to exchange a randomly generated key over an insecure channel, which is then used to continue in a symmetric manner.

Definition 9. A **one-way function** is an injective function $\varphi : X \to Y$ which is easy to compute for every input $x \in X$, but difficult to compute in the opposite direction.
A **trapdoor one-way function** (or simply trapdoor function) is a one-way function $\varphi : X \to Y$, with the specialty that it is easy to invert every $y \in \varphi(X)$ with a special piece of information, called the **trapdoor**. In public key cryptography, the designer of the trapdoor one-way function owns the trapdoor which allows him the computation $\varphi^{-1}(y) \, \forall y \in \varphi(X)$ in a short amount of time.

Remark. The term "difficult to compute" is not very precise. In practice this means, that the designer has strong reasons to believe that the process needs such an amount of time that the information he shares will not longer be of interest after successfully solved the problem. Such a reason ideally would be a proof that the problem cannot be solved under a certain large amount of steps. Another reason could be that the problem is an NP-complete problem, but often one has to be content with the fact that the problem (e.g. factorization) is very famous and the most able scientists were unable to find an

algorithm, which solves it in polynomial time, for a long time, which of course is no proof for its difficulty.

Example 2 (Trapdoor Function)**.** An easy to explain example for a trapdoor function is "factorization", which is used in the RSA cryptosystem. Consider

$$\varphi : \mathbb{P} \times \mathbb{P} \longrightarrow \mathbb{Z}, (p, q) \mapsto p \cdot q,$$

where $\mathbb{P}$ denotes the set of prime numbers. The function is injective and easy to evaluate. The opposite direction, namely find p and q from $n := \varphi(p, q)$ is believed to be computationally difficult if p and q are two randomly picked large primes. There are many recommendations how to choose these primes which we don't want to amplify.

With the notion of one-way function we can define a symmetric key system more formally:

Definition 10. A **symmetric** key system (or secret key system) is defined as a 5-tuple $(\mathcal{M}, \mathcal{C}, \mathcal{K}, e, d)$, where $\mathcal{M}$ denotes the set of messages, $\mathcal{C}$ the ciphertext space and $\mathcal{K}$ the set of keys. Further

$$e : \mathcal{M} \times \mathcal{K} \to \mathcal{C},\ (m, k) \mapsto c$$
$$d : \mathcal{C} \times \mathcal{K} \to \mathcal{M},\ (c, k) \mapsto m$$

are the functions to encrypt or decrypt a message m, respectively, with the key k and the following properties.

1. $\forall m \in \mathcal{M}$, $\forall k \in \mathcal{K}$: $d(e(m, k), k) = m$.
2. For any fixed $m \in \mathcal{M}$, the function $e_m : \mathcal{K} \to \mathcal{C}$ given by $k \mapsto e(m, k)$ is a one-way function.

The second condition guarantees that $e(m, k) = c$ will not reveal k. Famous examples of symmetric key systems are:

- The enigma machine,
- Data Encryption Standard (DES),
- Advanced Encryption Standard (AES).

Definition 11. Similar to a symmetric key system, an **asymmetric** key system is a 6-tuple $(\mathcal{M}, \mathcal{C}, \mathcal{K}_1, \mathcal{K}_2, e, d)$ with two key spaces $\mathcal{K}_1$ and $\mathcal{K}_2$ and

$$e : \mathcal{M} \times \mathcal{K}_2 \to \mathcal{C},\ (m, k_2) \mapsto c$$
$$d : \mathcal{C} \times \mathcal{K}_1 \to \mathcal{K},\ (c, k_1) \mapsto m$$

where e is a one-way trapdoor function with trapdoor k_1. Further $k_2 = \varphi(k_1)$ for a one-way function φ. k_2 is public and used to encrypt messages, while k_1 is kept private and is used to decrypt ciphertexts.

Definition 12. We say a decision problem is in the **complexity class P** if it can be solved by a deterministic Turing machine using a polynomial amount of computation time.

This definition presumes the comprehension of deterministic Turing machine which is an abstract machine in the field of theoretical computer science. We don't want to introduce the theory behind it. If the reader is not familiar with deterministic Touring machines he can think of it as an abstract machine which can perform the same operations as the CPU of a computer. For more detailed information about Turing machines we refer to [MC06].

Example 3. Agrawal et al. [AKS04] proved in 2004 that the decision problem if a number n is prime or composite is in P.

Definition 13. We say a decision problem is in the **complexity class NP** if it can be solved by a non-deterministic Turing machine using a polynomial amount of computation time. Furthermore, if the answer is "yes" it can be verified by a deterministic Turing machine.

More intuitively, NP is the set of all decision problems for which we can efficiently verify its correctness whenever the answer to the problem is "yes".

Example 4. A simple example for a problem in NP is the partition problem: Given a set of natural numbers S, is there a partition in two distinct subsets S_1 and S_2 with $S = S_1 \cup S_2$ and the sum of all elements in S_1 equals the sum of elements in S_2. We don't know an efficient algorithm which solves this question, but if any (possibly not efficient) algorithm returns "yes" and the subsets S_1 and S_2, we can prove its correctness in polynomial time by just computing the sum of S_1 and compare it with the sum of all elements in S_2.

By definition, every problem in P is also contained in NP since for verifying a problem in P we can just solve it in polynomial time, so the set P $\subset$ NP. We can specify the class of NP more in the subclass of NP-complete problems:

Definition 14. A problem p is said to be **NP-hard** if every problem in NP can be reduced in polynomial time to p. If the NP-hard problem p is in NP we call it **NP-complete**.

It is still an open question if P = NP or not. Figure 2.3 shows the relations between the classes P, NP, NP-complete and NP-hard for the case of P $\neq$ NP on the left and for P = NP on the right.

Notation 2. For analyzing the efficiency of time or space usage of an algorithm, we use the big-O notation. We say a function f grows not essentially faster than a function g ($f \in \mathcal{O}(g)$) if for sufficiently large x: $|f(x)| \leq k \cdot g(x)$ for some $k > 0$. Further we call f

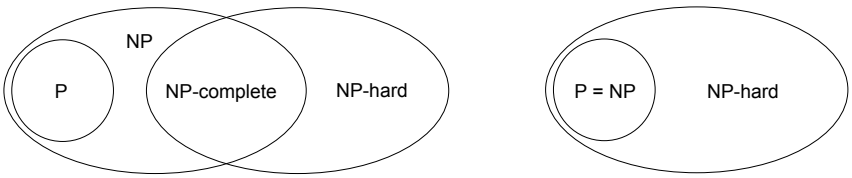

Figure 2.3: Schematic Diagram of Problem Classes

- **constant** if $f = \mathcal{O}(1)$
- **logarithmic** if $f = \mathcal{O}(\log(x))$
- **linear** if $f = \mathcal{O}(x)$
- **polynomial** if $f = \mathcal{O}(x^c)$ for a $c > 0$
- **exponential** if $f = \mathcal{O}(c^x)$ for a $c > 0$

Related to the big-O notation is the little-o notation. A function f is negligible compared to g ($f \in o(g)$) if for every constant $k > 0$ one can find an a such that $|f(x)| \leq k \cdot g(x)$ holds for all $x > a$. It holds that every function $f \in o(g)$ is also in $\mathcal{O}(g)$, but not generally otherwise.

2.2 Hermite Normal Form

In this section, we introduce the **Hermite normal form** (HNF), which we use later in Chapter 6. Various authors define this term differently, so we decided to follow the notion of Cohen [Coh93].

Definition 15. We say that an $(m \times n)$-matrix $\mathbf{M} = (m_{i,j})$ with integer coefficients is in Hermite normal form (HNF) if there exists $r \leq n$ and a strictly increasing map

$$f : \{r+1, \ldots, n\} \longrightarrow \{1, \ldots, m\}$$

satisfying the following properties.

1. For $r + 1 \leq j \leq n$, $m_{f(j),j} \geq 1$, $m_{i,j} = 0$ if $i > f(j)$ and $0 \leq m_{f(k),j} < m_{f(k),k}$ if $k < j$.
2. The first r columns of $\mathbf{M}$ are equal to 0.

Remark. So generally, a matrix in HNF has the following shape:

$$\begin{pmatrix} 0 & \ldots & 0 & \star & \star & \ldots & \star \\ 0 & \ldots & 0 & 0 & \star & \ldots & \star \\ \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \ldots & 0 & 0 & \ldots & 0 & \star \end{pmatrix}$$

Definition 16 (Special Case)**.** In this thesis we have mostly the case where $n = m$ and linearly independent columns (i.e. $\det(\mathbf{M}) \neq 0$), for which we can simplify the previous definition: $\mathbf{M} = (m_{i,j})$ is in HNF if it satisfies the following conditions.

1. $\mathbf{M}$ is an upper triangular matrix, i.e. $m_{i,j} = 0$ if $i > j$.
2. For every i, we have $m_{i,i} > 0$.
3. For every $j > i$ we have $0 \leq m_{i,j} < m_{i,i}$.

Theorem 5. Let $\mathbf{A}$ be an $(m \times n)$-matrix with coefficients in $\mathbb{Z}$. Then, there exists a unique $(m \times n)$-matrix $\mathbf{B} = (b_{i,j})$ in HNF of the form

$$\mathbf{B} = \mathbf{AU}$$

with $\mathbf{U} \in GL_n(\mathbb{Z})$.

Proof. Existence. Consider Algorithm 1. The algorithm terminates since $|a_{i,k}|$ is strictly decreasing each time we return to step 2 from step 4. First note that $\mathbf{W}$ is in HNF by construction, and since it is obtained from $\mathbf{A}$ by elementary column operation, $\mathbf{W}$ is the HNF of $\mathbf{A}$.
Uniqueness. Suppose there is another HNF of $\mathbf{A}$, say $\mathbf{G} \neq \mathbf{H}$. Since both, $\mathbf{G}$ and $\mathbf{H}$ are obtained from $\mathbf{A}$ by elementary column operation, they generate the same lattice $\Lambda = \Lambda(\mathbf{G}) = \Lambda(\mathbf{H})$. We choose now i, j such that $h_{i,j} \neq g_{i,j}$, where i is the smallest of all i's satisfying this property. Without loss of generality $h_{i,j} > g_{i,j}$. Since $\mathbf{H}_j$ and $\mathbf{G}_j$ are vectors in Λ, $\mathbf{H}_j - \mathbf{G}_j \in \Lambda$. This means, that there exists a linear combination of columns of $\mathbf{H}$ for $\mathbf{H}_j - \mathbf{G}_j$. By choice of i, the first $i - 1$ components of $\mathbf{H}_j - \mathbf{G}_j$ are zeros, so it is actually a linear combination of columns of $i, \ldots, n$ of $\mathbf{H}$. But among these columns, only the i-th column has a non-zero entry in the i-th component, which implies that $h_{i,j} - g_{i,j} = z \cdot h_{i,i}$ for some $z \in \mathbb{Z}$. But $h_{i,j} < h_{i,i}$ and $g_{i,j} < g_{i,i} < h_{i,i}$, i.e. $|h_{i,j} - g_{i,j}| < h_{i,i}$. This works only for $z = 0$, which is a contradiction to $h_{i,j} \neq g_{i,j}$. $\square$

Algorithm 1 Hermite Normal Form

Input: $(m \times n)$-matrix $\mathbf{A} = (a_{i,j})$, $a_{i,j} \in \mathbb{Z}$
Output: Hermite normal form $\mathbf{W}$ of $\mathbf{A}$

[Step 1 - Initialize]:
$i := m$
$k := n$
if $m \leq n$ **then**
  $l := 1$
else
  $l := m - n + 1$
end if

[Step 2 - Row finished?]:
if $a_{i,j} = 0$ for $j < k$ **then**
  if $a_{i,k} < 0$ **then**
    Replace column $\mathbf{A}_k$ by $-\mathbf{A}_k$ and go to Step 5
  end if
end if

[Step 3 - Choose non-zero entry]:
Pick among the non-zero $a_{i,j}$ for $j \leq k$ one with the smallest absolute value, say a_{i,j_0}.
if $j_0 < k$ **then**
  exchange column $\mathbf{A}_k$ with column $\mathbf{A}_{j_0}$
end if
if $a_{i,k} < 0$ **then**
  replace column $\mathbf{A}_k$ by $-\mathbf{A}_k$.
end if
$b := a_{i,k}$

[Step 4 - Reduce]:
for $j = 1 \to k - 1$ **do**
  $q := \lfloor a_{i,j}/b \rceil$
  $\mathbf{A}_j - q\mathbf{A}_k$
end for
go to Step 2

[Step 5 - Final reductions]:
$b := a_{i,k}$
if $b = 0$ **then**
  $k := k + 1$
  go to Step 6
else
  for $j > k$ **do**
    $q := \lfloor a_{i,j}/b \rceil$
    $\mathbf{A}_j - q\mathbf{A}_k$
  end for
end if

[Step 6 - Finished?]:
if i=l **then**
  for $j = 1 \to n - k + 1$ **do**
    $\mathbf{W}_j := \mathbf{A}_{j+k-1}$
  end for
  terminate
else
  $i := i - 1$
  $k := k - 1$
  go to step 2
end if

Remark. Note that although $\mathbf{B}$ is unique, the matrix $\mathbf{U}$ will not be unique. Furthermore, if we have two different bases generating the same lattice, the HNF of both will be equal. So the HNF gives a unique basis for every lattice. We will prove this fact in the following

Lemma 6. Let $\mathbf{A}$ and $\mathbf{B}$ be two $(n \times n)$-matrices with full rank generating the same lattice Λ, and let $\mathbf{G}$, and $\mathbf{H}$ respectively, be their Hermite normal form, then $\mathbf{G} = \mathbf{H}$.

Proof. Since $\Lambda(\mathbf{G}) = \Lambda(\mathbf{A}) = \Lambda = \Lambda(\mathbf{B}) = \Lambda(\mathbf{H})$, there exists an unimodular matrix $\mathbf{U}$ such that $\mathbf{G} = \mathbf{UH}$. Since $\mathbf{G}$ and $\mathbf{H}$ are upper triangular matrices with non-zero determinant, we know that $\mathbf{U}$ is an upper triangular matrix. We now look to which elementary row operations $\mathbf{U}$ corresponds to:

1. $\mathbf{U}$ cannot switch two rows, since then $\mathbf{G}$ will be not longer upper triangular.
2. $\mathbf{U}$ cannot multiply a row by -1, since this would make an entry of $\mathbf{G}$ in the diagonal to be less than zero, which contradicts the HNF.
3. Since both $\mathbf{G}$ and $\mathbf{H}$ are upper triangular matrices, $\mathbf{U}$ can a priori just add column j to i where $j < i$ to hold its triangularity, but also this is a contradiction since adding a column j to i where $j < i$ leads to $h_{k,j} > h_{k,i}$ which would also violate the hermit normal form.

We conclude that $\mathbf{U}$ does not make any elementary row operation, i.e. $\mathbf{U} = \mathbf{I_n}$ and thus $\mathbf{G} = \mathbf{H}$. □

2.3 (d, M)-Lattices

Definition 17. Assume $n \in \mathbb{N}$, $M, d \in \mathbb{R}_{>0}$, and $\Lambda \subset \mathbb{Z}^n$ is a lattice which has an $(n-1)$-dimensional sublattice Λ' with the following properties:

1. Λ' has a basis of length at most M.
2. If H is the $(n-1)$-dimensional subspace of $\mathbb{R}^n$ containing Λ' and $H' \neq H$ is a coset of H intersecting Λ, then the distance of H and H' is at least d.

Then, we say that Λ is a (d, M)**-lattice**. The minimum distance between H and a coset of H intersecting Λ will be denoted by d_Λ.
If $d > M$, then Λ' is unique. In this case, Λ' will be denoted by $\Lambda^{(d,M)}$.

Example 5. We consider the lattice $\Lambda := \Lambda(\boldsymbol{b_1}, \boldsymbol{b_2})$, where $\boldsymbol{b_1} = (2, 3)$ and $\boldsymbol{b_2} = (6, 1)$. As Λ' we choose the lattice generated by $\boldsymbol{b_1}$, which has length $\sqrt{13} \approx 3.61$. Then, H clearly equals $\mathbf{span}\,(\boldsymbol{b_1})$. The distance to the nearest coset (H'), called d_Λ is approximately 4.44. So, Λ is a (d, M)-lattice with $d = 4.4$ and $M = 3.7$. Since $d > M$, Λ' is unique and we call it $\Lambda^{(d,M)}$. This example is shown in Figure 2.4, the sublattice Λ' is highlighted.

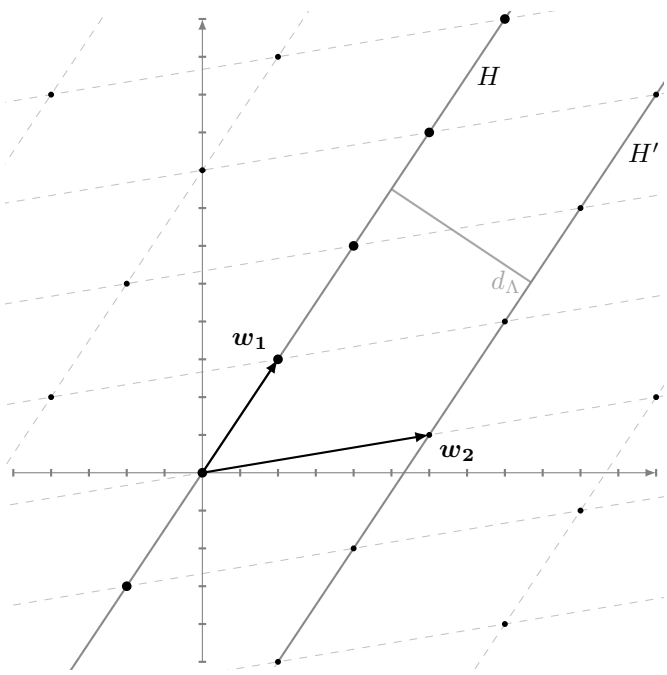

Figure 2.4: $(4.4, 3.7)$-Lattice

Definition 18 (Minimum Distance)**.** The **minimum distance** of a lattice Λ, denoted by $\lambda_1(\Lambda)$, is the minimum distance between any two distinct lattice points, and equals the length of the shortest nonzero lattice vector:

$$\begin{aligned}\lambda_1(\Lambda) :&= \min\{\|\boldsymbol{x}-\boldsymbol{y}\| \mid \boldsymbol{x} \neq \boldsymbol{y} \in \Lambda\}\\ &= \min\{\|\boldsymbol{x}\| \mid \boldsymbol{0} \neq \boldsymbol{x} \in \Lambda\}\end{aligned}$$

This definition can be generalized to define the i-th **successive minimum** as the smallest λ_i such that the ball $\mathcal{B}_{\lambda_i} = \{\boldsymbol{x} \mid \|\boldsymbol{x}\| \leq \lambda_i\}$ contains at least i linearly independent lattice points. We denote it by $\lambda_i(\Lambda)$.

Definition 19. Another useful invariant for lattices is the **lattice gap** defined in [NS01] as follows: Let Λ be a lattice. The lattice gap is a real number defined as

$$\alpha(\Lambda) := \frac{\lambda_2(\Lambda)}{\lambda_1(\Lambda)} \tag{2.2}$$

2.4 Lattice Problems

Lattices have already been studied by Gauss, Dirichlet, Hermite and Minkowski, among others early in the 19th century [NS98]. From an algorithmic point of view, two problems

in the field of lattices have evolved (the Shortest Vector Problem and the Closest Vector Problem). In this section we discuss these problems and the complexity for solving them. Generally lattice are defined over $\mathbb{R}$. However, from a computational point of view, we can only define them over $\mathbb{Q}$, which is equivalent to defining them over $\mathbb{Z}$, since every lattice over the rationals can be transformed to a lattice over $\mathbb{Z}$ by multiplying the basis by a suitable integer.

2.4.1 Shortest Vector Problem

Definition 20 (Shortest Vector Problem (SVP))**.** Given a norm $\|\cdot\|$ (often L_2) and a basis B for a non-trivial lattice $\Lambda \subset \mathbb{R}^n$ (i.e. Λ contains a non-zero vector), find a vector $\boldsymbol{u}$ such that

$$\|\boldsymbol{u}\| = \min \{\|\boldsymbol{v}\| \mid \boldsymbol{v} \in \Lambda, \boldsymbol{v} \neq \mathbf{0}\}$$

Remark. The **shortest vector** $\boldsymbol{u}$ is not unique. Obviously if $\boldsymbol{u}$ is a short vector, so is $-\boldsymbol{u}$, but there can also be more vectors in Λ with the same norm as $\boldsymbol{u}$ and $-\boldsymbol{u}$, which are linearly independent of $\boldsymbol{u}$.
In 1981, P. van Emde-Boas [vEB81] showed that the exact version of the problem is NP-hard under the L_∞-norm. However, most of the applications are based on the L_2-norm. In 1998, Ajtai [Ajt98] showed that SVP under L_2-norm is also NP-hard for randomized reductions.
In 2009, Daniele Micciancio and Panagiotis Voulgaris [MV10] presented a deterministic algorithm to solve SVP in a n-dimensional lattice in $2^{3.199n}$ time and $2^{1.325n}$ space.

There are many applications which do not need an exact short vector. In this case an approximation is enough. This brings out the following problem:

Approximate Shortest Vector Problem

Definition 21 (Approximate Shortest Vector Problem (γ-SVP))**.** Given an approximation factor γ, a norm $\|\cdot\|$ and a basis B for a non-trivial lattice $\Lambda \subset \mathbb{R}^n$, find a vector $\boldsymbol{u}$ such that

$$\|\boldsymbol{u}\| \leq \gamma \cdot \lambda_1 (\Lambda)$$

Unique Shortest Vector Problem

The unique shortest vector problem has became famous because the first cryptosystems of Ajtai and Dwork [AD97] were built on its hardness. It is not a different problem as such, but on the requirements to the lattice itself. It restricts the lattice to have a unique shortest vector in the following sense: Let $\gamma = n^{\mathcal{O}(1)}$, where n is the rank of the lattice. Λ has a γ-unique shortest vector when $\lambda_2 (\Lambda) > \gamma \lambda_1 (\Lambda)$.

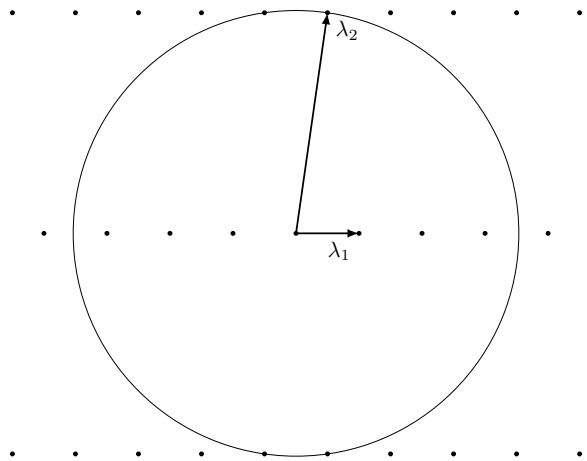

Figure 2.5: A Lattice with a Unique Shortest Vector

Definition 22 (Unique Shortest Vector Problem (uSVP))**.** Given a lattice Λ with a γ-unique shortest vector, find a vector $\boldsymbol{v} \in \Lambda$ with $\|\boldsymbol{v}\| = \lambda_1(\Lambda)$.

Remark. The term unique is due to the fact that in such lattices the shortest vector $\boldsymbol{u}$ is unique in the sense, that any vector $\boldsymbol{v}$ of length less than $\gamma\lambda_1(\Lambda)$, is parallel to $\boldsymbol{u}$. This implies that $\boldsymbol{v} = c \cdot \boldsymbol{u}$ for a $c \in \mathbb{Z}$. So, for such a lattice γ-SVP and uSVP are computationally equivalent. If one has found a γ-approximation $\boldsymbol{v}$ of an exact shortest vector $\boldsymbol{u}$, then we know $\boldsymbol{v} = c \cdot \boldsymbol{u}$ and can therefore check $\boldsymbol{v}/c$ for $c \in \mathbb{N}_{<\gamma}$. Since γ is polynomially bounded, there are only polynomially many c's which are such candidates.
In 2008, Vadim Lyubashevsky [Lyu08] showed, that there is a reduction from the approximate decision version of the shortest vector problem (GapSVP) to the unique shortest vector problem.

2.4.2 Closest Vector Problem

The second main problem in the field of lattices is the Closest Vector Problem. It can be seen as the inhomogeneous version of SVP.

Definition 23 (Closest Vector Problem (CVP))**.** Given a norm $\|\cdot\|$, a basis B for a lattice $\Lambda \subset \mathbb{R}^n$ and a vector $\boldsymbol{v} \in \mathbb{R}^n$ (not necessary in the lattice), find a vector $\boldsymbol{u}$ such that

$$\|\boldsymbol{u} - \boldsymbol{v}\| = \min\{\|\boldsymbol{u} - \boldsymbol{v}\| \mid \boldsymbol{v} \in \Lambda\}$$

Remark. CVP is NP-hard. There are several proofs of this result. The first proof was made by Emde-Boas [vEB81] in 1981 (where the author called it Nearest Vector Problem). In 1986, Kannan [Kan87] gave a new simpler proof by reduction from three dimensional matching (3DM), which is NP-complete (see [GJ90]). In 2001, Micciancio [Mic01a] showed

an even simpler proof of the same NP-hardness result by reduction from the Subset Sum Problem.
The fastest deterministic algorithm known for solving CVP runs in $n^{(n/2)+o(n)}$. For an overview of the state of the art algorithms for CVP and SVP we refer to [HPS11].

Approximate Closest Vector Problem (γ-CVP)

Analogous to the approximate variant of SVP, there exists one for CVP which is not NP-hard.

Definition 24 (Approximate Closest Vector Problem)**.** Given a norm $\|\cdot\|$, a basis B for a lattice $\Lambda \subset \mathbb{R}^n$ and a vector $\boldsymbol{v} \in \mathbb{R}^n$, find a vector $\boldsymbol{u}$ such that

$$\|\boldsymbol{u} - \boldsymbol{v}\| < \gamma \min \{\|\boldsymbol{u} - \boldsymbol{v}\| \mid \boldsymbol{v} \in \Lambda\}$$

Remark. In 1993, Arora et al. [ABSS93] showed that even approximating CVP within any constant factor is also NP-hard.

In 1986, Babai [Bab86] gave two algorithms for approximating the CVP in polynomial time within an exponential factor, known as the round-off algorithm and the nearest plane algorithm. These algorithms do both not solve the CVP in every case, but nevertheless, they are very useful in many cryptosystems and attacks. We use these algorithms in GGH (Section 5), Micciancio's Cryptosystem (Section 6) and in some attacks, so we introduce it here.

Babai's Round-Off Algorithm

Algorithm 2 Babai's Round-Off Algorithm

Input: basis $\mathbf{R} = (\boldsymbol{r_1}, \ldots, \boldsymbol{r_n}) \in \mathbb{Z}^n$, target vector $\boldsymbol{c} \in \mathbb{R}^n$
Output: approximate closest lattice point of $\boldsymbol{c}$ in $\Lambda(\mathbf{R})$

```
procedure RoundOff
    Compute inverse of R: R^{-1} ∈ Q^n
    v := R ⌊R^{-1} c⌉
    return v
end procedure
```

Remark. The costly part in the algorithm is clearly the inversion of the matrix $\mathbf{R}$, which is $\mathcal{O}\left(n^3\right)$. The two matrix-vector multiplications to compute $\boldsymbol{v}$ costs only $\mathcal{O}\left(n^2\right)$. The inversion has only to be done once if $\mathbf{R}$ stays the same, so it can be stored and the future applications need only $\mathcal{O}\left(n^2\right)$ time complexity.
The method depends hardly on the quality of the basis, it achieves better approximations

if the basis is LLL-reduced (see Section 2.5), which is often stated as a separate step in Algorithm 2, before inverting $\mathbf{R}$. The complexity of the algorithm stays the same even with preceded LLL-reduction.
The computation can also be done without inverting $\mathbf{R}$ at all, by solving the equation $\mathbf{R}\boldsymbol{x} = \boldsymbol{c}$ for $\boldsymbol{x}$. $\boldsymbol{v}$ would then be computed by $\mathbf{R}\lfloor \boldsymbol{x} \rceil$.

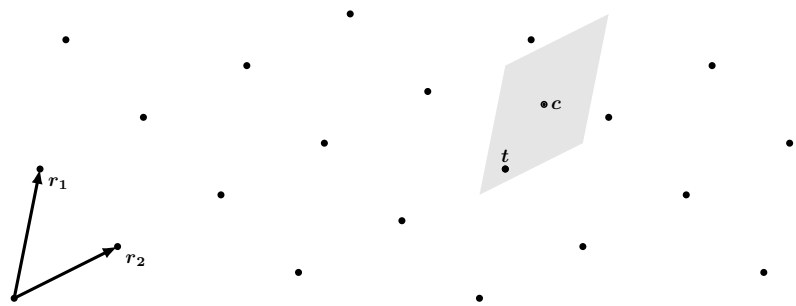

Figure 2.6: Babai's Round-Off Algorithm

Geometrically, Babai's round-off algorithm rounds the point $\boldsymbol{c} \in \mathbb{R}^n$ to its unique point $\boldsymbol{t} \in \Lambda$ in parallelepiped centered at $\boldsymbol{c}$. Figure 2.6 shows an example in dimension 2.

Theorem 7. Let $\mathbf{R}$ be an LLL-reduced basis (with respect to the Euclidean norm and with factor $\delta = 3/4$) for a lattice Λ. Then, the output $\boldsymbol{v}$ of the Babai's rounding method on input $\mathbf{W} \in \mathbb{R}^n$ satisfies

$$\|\boldsymbol{w} - \boldsymbol{v}\| \leq \left(1 + 2n(9/2)^{n/2}\right) \|\boldsymbol{w} - \boldsymbol{u}\|$$

for all $\boldsymbol{u} \in \Lambda$.

Proof. The proof can be found in [Bab86]. □

Babai's Nearest Plane Algorithm

In contrast to the round-off algorithm, which computes all coordinates at once, the nearest plane algorithm is iterative. It uses the Gram-Schmidt basis, which we introduce in Section 2.5. Like Babai's round-off method, also this method provide much better results if the basis is LLL-reduced.

Theorem 8. Let $\mathbf{R}$ be an LLL-reduced basis (with respect to the Euclidean norm and with factor $\delta = 3/4$) for a lattice Λ. Then the output $\boldsymbol{v}$ of the Babai's nearest plane method on input $\boldsymbol{w} \in \mathbb{R}^n$ satisfies

$$\|\boldsymbol{w} - \boldsymbol{v}\| \leq \left(2^{n/2}\right) \|\boldsymbol{w} - \boldsymbol{u}\|$$

for all $\boldsymbol{u} \in \Lambda$.

Proof. The proof can be found in [Bab86]. □

Algorithm 3 Babai's Nearest Plane Algorithm

Input: basis $\mathbf{R} = (\boldsymbol{r_1}, \ldots, \boldsymbol{r_n}) \in \mathbb{Z}^n$, target vector $\boldsymbol{c} \in \mathbb{R}^n$

Output: approximate closest lattice point of $\boldsymbol{c}$ in $\Lambda(\mathbf{R})$

```
procedure NearestPlane
    Compute GramSchmidt of R: R* = (r*_1, ..., r*_n) ∈ Z^n
    for i = n → 1 do
        x_i := ⌊⟨c, r*_i⟩ / ||r*_i||⌉
        c := c − x_i r_i
    end for
    return v := R(x_1, ..., x_n)
end procedure
```

Remark. In [DK12] Daniel Dadush and Gábor Kun presented a deterministic algorithm for solving aCVP on any n-dimensional lattice and any norm in $2^{\mathcal{O}(n)}(1 + 1/\epsilon)^n$ time and $2^n poly(n)$ space.

2.4.3 Relationship of SVP and CVP

There are two fundamental differences between SVP and CVP. As mentioned earlier, CVP can be seen as the inhomogeneous version of SVP, this means, that SVP asks for a lattice point near to zero, while the solution to CVP is a lattice point close to an arbitrary point in the space. The other main difference is, that the solution of SVP must not be the all zero vector, while in CVP, this vector can be a possible answer. So one can not ask CVP for the shortest vector close to the origin to solve the SVP, since it would return the zero vector in every lattice. In this section we first show a method by Goldreich et al. to solve SVP with a CVP-oracle and then a heuristic converse, i.e. how to solve CVP by an SVP-oracle.

Reducing SVP to CVP

Goldreich et al. found a method to reduce SVP to CVP described in [GMSS99], which we show here. The method is dimension preserving, so the CVP will have the same dimension as the originally SVP. We first start with a

Lemma 9. Let $\boldsymbol{v} = \sum_{i=1}^{n} c_i \boldsymbol{bi}$ be a shortest non-zero vector in a lattice Λ. Then, there exists an i such that c_i is odd.

Proof. Assume the contrary, then $\boldsymbol{v}$ can be written as $\sum_{i=1}^{n} 2c_i'\boldsymbol{b_i}$, where all c_i' are integers. This contradicts the shortness of $\boldsymbol{v}$, since $\boldsymbol{v}' = \sum_{i=1}^{n} c_i'\boldsymbol{b_i}$ is also in the lattice Λ and its norm is only half as large. □

Now, the reduction goes as follows: Assume that we have the basis $\mathbf{B} = (\boldsymbol{b_1}, \ldots, \boldsymbol{b_n})$ for the lattice Λ.

Definition 25. We define n different bases (these bases don't generate the same lattice):

$$\mathbf{B}^{(j)} := (\boldsymbol{b_1}, \ldots, \boldsymbol{b_{j-1}}, 2\boldsymbol{b_j}, \boldsymbol{b_{j+1}} \ldots, \boldsymbol{b_n})$$

and n corresponding instances of CVP:

$$CVP^{(j)} := CVP(\mathbf{B}^{(j)}, \boldsymbol{b_j})$$

We now call all CVP instances $CVP^{(j)}$ and denote the result by $\boldsymbol{v_j}$. The shortest vector of Λ is the shortest vector of $(\boldsymbol{b_1} - \boldsymbol{v_1}, \ldots, \boldsymbol{b_n} - \boldsymbol{v_n})$. We prove this in Theorem 12, but first we need some preparatory work:

Proposition 10. Let $\boldsymbol{v} = \sum_{i=1}^{n} c_i\boldsymbol{b_i}$ be a lattice vector in Λ such that c_j is odd. Then $\boldsymbol{u} = \frac{c_j+1}{2}(2\boldsymbol{b_j}) + \sum_{i\neq j} c_i\boldsymbol{b_i}$ is a lattice vector in $\Lambda(\mathbf{B}^{(j)})$ and the distance of $\boldsymbol{u}$ from the target $\boldsymbol{b_j}$ equals the length of $\boldsymbol{v}$.

Proof. Note that $\boldsymbol{u} \in \Lambda(\mathbf{B}^{(j)})$ since $\frac{c_j+1}{2}$ is an integer (as c_j is odd). Also observe that

$$\boldsymbol{u} - \boldsymbol{b_j} = \frac{c_j+1}{2}(2\boldsymbol{b_j}) + \sum_{i\neq j} c_i\boldsymbol{b_i} - \boldsymbol{b_j} = c_j\boldsymbol{b_j} + \sum_{i\neq j} c_i\boldsymbol{b_i} = \boldsymbol{v}$$

and the proposition follows. □

Proposition 11. Let $\boldsymbol{u} = c_j'(2\boldsymbol{b_j}) + \sum_{i\neq j} c_i\boldsymbol{b_i}$ be a lattice vector in $\Lambda(\mathbf{B}^{(j)})$. Then $\boldsymbol{v} = (2c_j' - 1)\boldsymbol{b_j} + \sum_{i\neq j} c_i\boldsymbol{b_i}$ is a non-zero lattice vector in $\Lambda = \Lambda(\mathbf{B})$ and the length of $\boldsymbol{v}$ equals the distance of $\boldsymbol{u}$ from the target $\boldsymbol{b_j}$.

Proof. Note that $\boldsymbol{v}$ is non-zero since $(2c_j' - 1)$ is an odd integer. Further observe that

$$\boldsymbol{v} = (2c_j' - 1)\boldsymbol{b_j} + \sum_{i\neq j} c_i\boldsymbol{b_i} = c_j'(2\boldsymbol{b_j}) + \sum_{i\neq j} c_i\boldsymbol{b_i} - \boldsymbol{b_j} = \boldsymbol{u} - \boldsymbol{b_j}.$$

□

Theorem 12. A shortest vector of $(\boldsymbol{b_1} - \boldsymbol{v_1}, \ldots, \boldsymbol{b_n} - \boldsymbol{v_n})$ is indeed a shortest vector of Λ.

Proof. Let $\boldsymbol{v}$ be a shortest vector in Λ. We first recall, that $\boldsymbol{v}$ has an odd coefficient (by Lemma 9). We can now apply Proposition 10, which states that there is an instance $CVP^{(j)}$ with solution $\boldsymbol{u}$ of distance at most $\|\boldsymbol{u} - \boldsymbol{b_j}\| = \|\boldsymbol{v}\|$ to $\boldsymbol{b_j}$. That means, that there exists at least one instance $CVP^{(j)}$ over all the CVP-instances in the method, whose solution vector has distance to the target vector that is equal to the the length of a shortest vector in Λ.
On the other hand we have Proposition 11 which states, that every solution $\boldsymbol{v_j}$ of the $CVP^{(j)}$ instance leads to a vector $\boldsymbol{u}$ in the lattice Λ with norm equals the distance of $\boldsymbol{v_j}$ to $\boldsymbol{b_j}$, i.e. $\|\boldsymbol{u}\|$ can not be smaller than a shortest vector in Λ. This completes the proof. □

This means that the SVP can be reduced to the CVP, so the computational complexity of the CVP is at least as hard as the complexity of the SVP. In [GMSS99] the authors also prove that this method preserves the approximation factor.

Reducing CVP to SVP

In this section we give a method to solve CVP by an SVP-oracle. Although the dimension of the corresponding SVP-oracle is just one dimension larger than the CVP to solve, this is not a reduction in terms of computational complexity, since the method is heuristic. It is also known as the "embedding technique" [Kan87], and it works as follows: Let $\mathbf{B}$ be a basis for an n-dimensional lattice Λ, and $\boldsymbol{c}$ a point in $\mathbb{Z}^n$ (generally not in the lattice). We want to find the closest vector $\boldsymbol{t}$ to $\boldsymbol{c}$ in Λ. Consider the lattice Λ' generated by

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

The closest vector $\boldsymbol{t}$ is obviously in the lattice Λ' and also the vector $\boldsymbol{c}$. Thus, the vector $\boldsymbol{e} := \boldsymbol{t} - \boldsymbol{c}$ is also in the lattice and furthermore it is very short in comparison to the other vectors in the lattice. We can now solve the SVP for Λ' with LLL or any other lattice reduction algorithm to get a short vector, in the hope, that its first n entries are the vector $\boldsymbol{e}$ and so $\boldsymbol{t} = \boldsymbol{c} - \boldsymbol{e}$. $\boldsymbol{m}$ can then be computed by multiplying $\boldsymbol{t}$ by $\mathbf{B}^{-1}$.

Remark. This method is very efficient, since LLL runs in polynomial time as a function of the rank of the lattice (see Section 2.5) and it often performs much better in practice than expected. This is why the embedding technique is a good tool for the cryptanalysis of lattice based cryptosystem with underlying CVP problems. Although, it is not a reduction of CVP to SVP in terms of complexity theory.
The use of the BKZ algorithm would allow to find an exact shortest vector, and therefore

the error vector. However, in the exact version BKZ is exponential in time, and therefor no reduction of CVP to SVP.

2.4.4 Hidden Hyperplane Problem

An equivalent formulation of the uSVP is the so called Hidden Hyperplane Problem, which is the problem the Ajtai-Dwork cryptosystem is directly based on. We give here the definition from [AD97]:

Definition 26 (Hidden Hyperplane Problem (HHP))**.** Let $1 < c \in \mathbb{R}$ and let $\mathcal{L}$ be a distribution on the set of all (d, M)-lattices where $d > n^c M$ and $d \leq d_\Lambda \leq 2d$. The Hidden Hyperplane Problem for $\mathcal{L}$ is the following: given a basis for a random (d, M)-lattice $\Lambda \in \mathcal{L}$, compute the unique distance $\Lambda^{(d,M)}$.

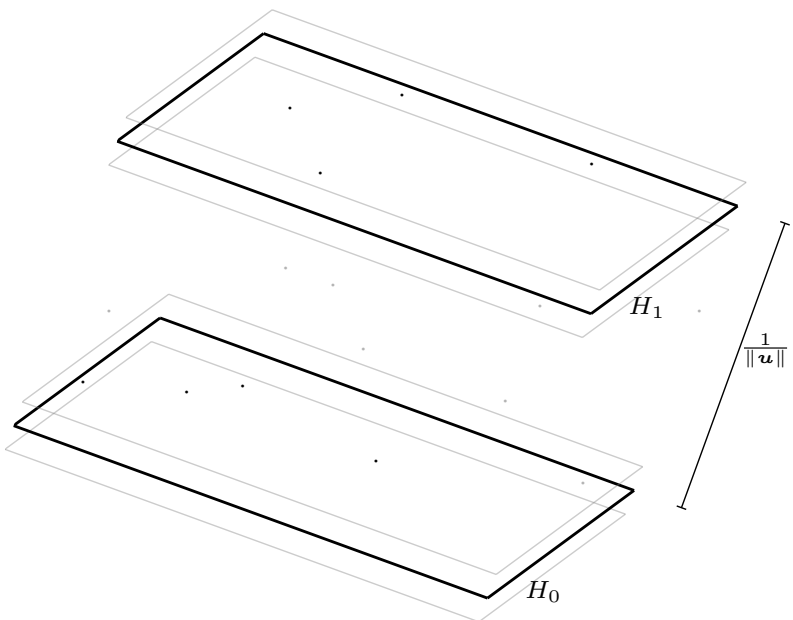

Figure 2.7: Hidden Hyperplane Problem

Remark (Connection to uSVP)**.** The HHP is related to the uSVP as follows: If Λ is a lattice with an n^c-unique shortest vector $\boldsymbol{u}$, then $L := \Lambda^\star$ is a (d, M)-lattice for some $d \geq n^c M$ and $d_L = \|\boldsymbol{u}\|^{-1}$. Let H_0 be the $n-1$-dimensional hyperplane of $\mathbb{R}^n$ containing $L^{(d,M)}$, and let $H_i := \{\boldsymbol{v} \mid \langle \boldsymbol{u}, \boldsymbol{v} \rangle = i\}$. The vector $\boldsymbol{u}$ is clearly orthogonal to H_0, because its scalar product with any vector in H_0 is 0. Further, the distance between two adjacent hyperplanes is $\|\boldsymbol{u}\|^{-1}$.
So, on the one hand by knowing H_0 one can first compute the direction of $\boldsymbol{u}$ and then the

length of $d_L = \|\boldsymbol{u}\|^{-1}$ by simply take the gcd of the distances of random points in L to H_0. This can be done in probabilistic polynomial time. On the other hand, if one knows the n^c-unique shortest vector $\boldsymbol{u}$ of Λ, one can directly compute the hyperplanes H_i.

2.4.5 Short Basis Problem

Definition 27 (Short Basis Problem (SBP))**.** Let Λ be a n-dimensional lattice. Find a basis $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ for Λ, such that for any other basis $(\boldsymbol{b}'_1, \ldots, \boldsymbol{b}'_n)$

$$bl(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) < \gamma \cdot bl(\boldsymbol{b}'_1, \ldots, \boldsymbol{b}'_n)$$

where $\gamma = n^{\mathcal{O}(1)}$ is called approximation factor. The variant, where $\gamma = 1$ is referred to as exact version.

In the next section we introduce two algorithms (LLL and BKZ) solving the short basis problem in the approximate variant as well as in the exact variant.

2.5 Lattice Basis Reduction

Lattice basis reduction is one of the key elements in the field of lattices. Since lattice are generally represented by a basis and lattice problems as SVP or CVP are much easier to solve with reduced basis, lattice basis reduction algorithms are often used as a preparation part of solving these lattice problems.

In $\mathbb{R}^n$, every basis can be transformed to an orthogonal basis using the Gram-Schmidt

Algorithm 4 Gram-Schmidt

```
procedure GramSchmidt(basis B = (b_1, ..., b_n))
    for i = 1 → n do
        b*_i := b_i
    end for
    for i = 2 → n do
        for j = 1 → i − 1 do
            μ_ij := ⟨b_i, b*_j⟩ / ⟨b*_j, b*_j⟩
            b*_i := b*_i − μ_ij b*_j
        end for
    end for
    return b*_1, ..., b*_n
end procedure
```

process. It would be interesting to study orthogonal basis also in lattices, but generally it is not possible to transform a lattice basis to an orthogonal one generating the same

lattice. However, we can consider equivalent lattice basis that are nearly orthogonal. A good measure of nearly orthogonal for a lattice Λ with basis $\mathbf{B} = [b_1, \ldots, b_n]$ is the

Definition 28 (Orthogonality Defect)**.**

$$\delta(\Lambda) = \frac{\prod_{i=1}^{n} \|b_i\|}{vol(\Lambda)} = \frac{\prod_{i=1}^{n} \|b_i\|}{\sqrt{\det(\mathbf{B}^T\mathbf{B})}} \geq 1$$

Remark. Equality holds if and only if the basis $\mathbf{B}$ is orthogonal. Furthermore, one can see, that reducing the length of the vectors b_i also reduces the the orthogonality defect of the lattice.

Minimizing the orthogonality defect (i.e. finding the smallest possible lattice basis) is NP-complete. However, there exist polynomial time algorithms which approximate a short basis. In the following we introduce two such algorithms.

2.5.1 The LLL Algorithm

The Lenstra-Lenstra-Lovász (LLL) lattice basis reduction algorithm invented by Arjen Lenstra, Hendrik Lenstra and László Lovász [LLL82] in 1982 is probably the most famous algorithm for reducing a lattice basis and was a real breakthrough. It runs in polynomial time (as a function of the rank of the lattice) and gives a reduced basis of a lattice. As a side product it also returns an approximation of the short vector. With an exponential approximation factor γ of $2^{(n-1)/2}$ it seems not very useful at a first glance, but it turned out that it behaves much better in practice than in theory, especially for low dimensions and for many lattice problems an approximation of the shortest vector or a reduced basis suffice. In this section we give a brief overview of the LLL algorithm. Since 1982 many variations evolved, so we cannot give the full theory about LLL. For more details we refer to [NV09].

Definition 29. Let $\mathbf{B} = (\boldsymbol{b_1}, \ldots, \boldsymbol{b_n})$ be a basis for a lattice, define its Gram-Schmidt orthogonal basis as

$$\mathbf{B}^* = (\boldsymbol{b_1^*}, \ldots, \boldsymbol{b_n^*})$$

and the **Gram-Schmidt coefficients** as

$$\mu_{i,j} = \frac{\langle \boldsymbol{b_i}, \boldsymbol{b_j^*} \rangle}{\langle \boldsymbol{b_j^*}, \boldsymbol{b_j^*} \rangle}$$

for $1 \leq j < i \leq n$.

Remark. Let $\mathbf{M}$ be the lower triangular matrix $\begin{pmatrix} 1 & & & \\ \mu_{2,1} & 1 & & \\ \vdots & \ddots & \ddots & \\ \mu_{n,1} & \ldots & \mu_{n,n-1} & 1 \end{pmatrix}$, then

$$\mathbf{B} = \mathbf{B}^*\mathbf{M}^T$$

and so

$$vol(\Lambda) = \sqrt{\det(\mathbf{B}^T\mathbf{B})} = \sqrt{\det((\mathbf{B}^*)^T\mathbf{B}^*)} = \prod_{i=1}^{n} \|\boldsymbol{b}_i^*\|$$

since $(\mathbf{B}^*)^T\mathbf{B}^*$ is a diagonal matrix and $\det \mathbf{M} = 1$.

Definition 30 (LLL-reduced)**.** Let $\delta \in (\frac{1}{4}, 1]$. A basis $\mathbf{B}$ is called δ-LLL-reduced if the following two conditions hold:

- (size-reduced): $1 \leq j < i \leq n$:
$$|\mu_{i,j}| \leq \frac{1}{2}$$
- (Lovász condition): For $k = 2, \ldots, n$:
$$\delta\|\mathbf{b}_{k-1}^*\|^2 \leq \|\mathbf{b}_k^*\|^2 + \mu_{k,k-1}^2\|\mathbf{b}_{k-1}^*\|^2$$

We give here the LLL algorithm in a very high-level form. As input, the algorithm expects a lattice basis in matrix form as well as a $\delta \in (\frac{1}{4}, 1)$ and returns a δ-reduced lattice basis.

Algorithm 5 LLL algorithm

Input: basis $\mathbf{B} = (\boldsymbol{b_1}, \ldots, \boldsymbol{b_n}) \in \mathbb{Z}^n$, $\delta \in (\frac{1}{4}, 1]$
Output: δ-lll-reduced basis of $\Lambda(\mathbf{B})$

```
procedure LLL(B, δ)
    for i = 1 → n do
        for j = i − 1 → 1 do
            c_ij := ⌊⟨b_i, b_j*⟩ / ⟨b_j*, b_j*⟩⌉
            b_i := b_i − c_ij b_j
        end for
    end for
    if δ ‖π_i(b_i)‖² > ‖π_i(b_{i+1})‖² for some i then
        swap b_i and b_{i+1}, and go to step 1.
    end if
    return B
end procedure
```

Remark. The stated algorithm is for integer lattices. It is also possible to apply the same algorithm for rational numbers by previously multiplying the basis by a certain factor $z \in \mathbb{Z}$ to get pure integers and afterwards divide the resulting basis by z. In practice, this original version is too inefficient in large dimensions, especially because of the fact that the Gram-Schmidt orthogonalization for long-integer arithmetic is much slower than that for floating-point arithmetic [NS05]. That is why often floating point variants, as this from

Stehlé presented in [Ste10], are used.
The complexity of this version of LLL is cubic. In 2009, Nguyen and Stehlé presented an LLL algorithm with quadratic complexity, which they called L^2-algorithm [NS09].

2.5.2 The BKZ Algorithm

The BKZ or Block Korkine-Zolotarev lattice reduction algorithm can somehow be seen as a generalization of the LLL algorithm. It was introduced 1987 by Schnorr in [Sch87], where he introduced a whole hierarchy of new reduction algorithms. This algorithm reduces a basis in the following sense.

Definition 31 (BKZ-reduced)**.** Let $\mathbf{B}$ be a basis for a lattice $\Lambda(\mathbf{B})$ of rank d, let $\delta \in (\frac{1}{4}, 1]$ and $\beta \in \mathbb{Z}$ such that $2 \leq \beta \leq d$. $\mathbf{B}$ is called δ-BKZ-reduced with block size β if the following two conditions hold:

- (size-reduced): $1 \leq j < i \leq n$:
$$|\mu_{i,j}| \leq \frac{1}{2} \tag{2.3}$$
- (BKZ-condition): For $1 \leq i \leq d-1$:
$$\delta \|\mathbf{b}_i^*\|^2 \leq \lambda_1 \left(\pi_i(\Lambda(\boldsymbol{b_1}, \ldots, \boldsymbol{b_{\min\{i+\beta-1,d\}}}))\right)^2 \tag{2.4}$$

Remark. BKZ uses so called blocks of vectors and performs an SVP in the sublattice spanned by these blocks. The quantity of blocks is given by the parameter β. So if $\beta = d$, we can be sure, that a shortest vector is in the reduced basis. Increasing the block size lowers the orthogonality defect of the reduced basis but also rises the running time exponentially. Determine the optimal block size for a given problem can be really hard. For $\beta = 2$ the δ-BKZ reduction is equivalent to the δ-LLL reduction [SE94].

Algorithm 6 shows a high-level form of the BKZ algorithm. We see that i goes cyclically through the set $\{1, \ldots, d-1\}$. For each i, $\boldsymbol{b_i}^{\text{new}}$ is the shortest vector in the sublattice $\Lambda(\pi_i(\boldsymbol{b_i}^*), \ldots, \pi_i(\boldsymbol{b_{\min\{i+\beta-1,d\}}}))$. If $d-1$ times no new vector was added, this means, that the BKZ-condition (2.4) is fulfilled. After every step a kind of LLL-algorithm is applied to $\mathbf{B}$ to ensure that the basis is size-reduced (2.3).

Remark. Both algorithms, LLL and BKZ, are implemented in the high-performance, portable C++ library [Sho14].

2.6 Worst-Case/Average-Case Equivalence

In this section we give an overview of the results of the famous paper "Generating Hard Instances of Lattice Problems" by Ajtai [Ajt96], which forms the basis for the security of

Algorithm 6 BKZ algorithm

Input: basis $\mathbf{B} = (\boldsymbol{b_1}, \ldots, \boldsymbol{b_d}) \in \mathbb{Z}^n$, $\delta \in (\frac{1}{4}, 1]$, $\beta \in \{2, \ldots, d\}$
Output: δ-BKZ-reduced basis of $\Lambda(\mathbf{B})$ with block size β

```
procedure BKZ(B, δ)
    Apply LLL to B
    i = 0
    repeat
        i := (i mod (d − 1)) + 1
        Find b_i^new such that ‖b_i^new‖ = λ_1 (Λ(π_i(b_i*), ..., π_i(b_{min{i+β−1,d}})))
        if δ ‖b_i‖^2 > ‖b_i^new‖^2 then
            Replace b_i by b_i^new
        end if
        Apply LLL to B
    until No vector was replaced in the last d − 1 steps
    return B
end procedure
```

the Ajtai-Dwork cryptosystem. The author gives a random class of lattices in $\mathbb{Z}^n$ together with a short vector in them and a proof of the following fact: If there is a probabilistic polynomial time algorithm which finds a short vector in the random lattice with probability at least $\frac{1}{2}$, then there is also a probabilistic polynomial time algorithm able to solve the following three lattice problems in *every* lattice in $\mathbb{Z}^n$ with probability exponentially close to one: γSVP, uSVP, and SBP.

Definition 32. Two lattice vectors $u = (u_1, \ldots, u_n)$ and $v = (v_1, \ldots, v_n)$ are **congruent modulo** $q \in \mathbb{N}$ ($u \equiv v \mod q$) if $u_1 \equiv v_1, \ldots, u_n \equiv v_n$ modulo q.

Definition of the Random Class

The definition of the random class will depend on two absolute constants c_1 and c_2. Let $n \in \mathbb{N}$ be given, define m and q as follows:

$$m := \lfloor c_1 n \log n \rceil \qquad\qquad q := \lfloor n^{c_2} \rceil$$

The random class consists of lattices with integer coordinates and are defined modulo q. So if $u, v \in \mathbb{Z}^n$ with $u \equiv v \mod q$, then either both or none of them belongs to the lattice. The elements of the lattices of the random class are integer sequences of length m which are orthogonal to a given sequence of vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m \in \mathbb{Z}^n$. More precisely: Let $\boldsymbol{\nu} = \boldsymbol{u}_1, \ldots, \boldsymbol{u}_m$, with $\boldsymbol{u}_i \in \mathbb{Z}^n$ for all $i \in \{1, \ldots, m\}$, then $\Lambda(\nu, q)$ is defined as the lattice consisting of all vectors $\boldsymbol{h} = (h_1, \ldots, h_m) \in \mathbb{Z}^m$ satisfying the following equation:

$$\sum_{i=1}^{m} h_i \boldsymbol{u}_i \equiv \mathbf{0} \mod q \tag{2.5}$$

For each n, we will define a single random variable λ so that $\Lambda = \Lambda(\lambda, q)$ is a lattice with dimension m. The problem of this definition of λ is, that it comes without a construction for a short vector in the lattice Λ. This is why we slightly change the definition of λ to overcome this issue: We randomize the vectors $v_1, \ldots, v_{m-1}$ independently with uniform distribution on the set $\mathbb{Z}^n$. Independently of this, we chose a sequence $(\delta_1, \ldots, \delta_{m-1}) \in \{0,1\}^{m-1}$ and define

$$v_m \equiv -\sum_{i=1}^{m-1} \delta_i \boldsymbol{v}_i \mod q$$

Remark. It can be shown, that the distribution of this new definition of λ is exponentially close to the originally definition. A proof of this can be found in [Ajt96].

Remark. The crucial point in this new definition is, that it comes together with a short vector in Λ, namely the vector $(\delta_1, \ldots, \delta_{m-1}, 1)$. This vector lies in the lattice by definition, and its L_2-norm is $\leq \sqrt{m}$.

Notation 3. If $k \in \mathbb{Z}$, we denote $size(k)$ the number of bits in the binary representation of k.

Theorem 13. There are absolute constants c_1, c_2, c_3 so that the following holds. Suppose that there is a probabilistic polynomial time algorithm $\mathcal{A}$ which given a value of the random variable λ_{n,c_1,c_2} as an input, with a probability of at least $\frac{1}{2}$ outputs a nonzero vector of $\Lambda(\lambda_{n,c_1,c_2},, \lfloor n^{c_2} \rceil)$ of length at most n. Then, there is a probabilistic algorithm $\mathcal{B}$ with the following properties: if the linearly independent vectors $a_1, \ldots, a_n \in \mathbb{Z}^n$ are given as an input, then $\mathcal{B}$, in time polynomial in $\sigma = \sum_{i=1}^n size(a_i)$, gives the outputs $z, u, (d_1, \ldots, d_n)$ so that, with probability of greater than $1 - 2^{-\sigma}$, the following three requirements are met.

- If v is a shortest vector in $\Lambda(a_1, \ldots, a_n)$ then $z \leq \|v\| \leq n^{c_3} z$.
- If v is an n^{c_3}-unique shortest vector in $\Lambda(a_1, \ldots, a_n)$ then $u = v$ or $u = -v$.
- $d_1, \ldots, d_n$ is a basis with $\max_{i=1}^n \|d_i\| \leq n^{c_3} bl(\Lambda)$.

Proof. The proof is very technical, so we leave it out. Interested readers can find the complete proof in [Ajt96]. □

One-Way Function

Assume that c_1, c_2 are given, and define m, q as above. For each $n \in \mathbb{N}$ we define a function $f = f^{(n)}$ in the following way:

$$\begin{aligned} f : \mathbb{Z}_q^n \times \cdots \times \mathbb{Z}_q^n \times \{0,1\} \times \cdots \times \{0,1\} &\longrightarrow \mathbb{Z}_q^n \times \cdots \times \mathbb{Z}_q^n \\ (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{m-1}, \delta_1, \ldots, \delta_{m-1}) &\mapsto (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m) \end{aligned} \tag{2.6}$$

where v_m is defined as follows:

$$v_m \equiv -\sum_{i=1}^{m-1} \delta_i \boldsymbol{v}_i \mod q \tag{2.7}$$

Remark. Assume now, $\boldsymbol{y} = f(\boldsymbol{x})$ for a random element $\boldsymbol{x} \in domain(f)$. If an algorithm is able to find $\boldsymbol{x}'$ with $f(\boldsymbol{x}') = \boldsymbol{y}$, then it has to find a short vector in $\Lambda(\lambda_{n,c_1,c_2})$.

Corollary 14. If for any of the mentioned worst-case problems in Theorem 13 there is *no* polynomial time probabilistic solution then f is a one-way function.

Remark. In 1999, Ajtai improved his work [Ajt99] by giving a method to generate a random lattice of essentially the same random class together with a short basis of the lattice.

Chapter 3

Ajtai-Dwork Cryptosystem

Even if eventually it can be proven that factoring is a hard problem, i.e. not in polynomial time solvable (in the worst-case), there is absolutely no guarantee that the number used in a RSA-system as public key is also hard to factor. There are several papers which tells us which parameters not to use, since they are easy to factor an therefore make the instance of RSA insecure (see e.g. [Mol02]). For a reliable cryptosystem one should be able to ensure that the chosen instance is as hard, so the underlying problem should be hard in the average-case. Exactly this is the foundation of Ajtai-Dwork cryptosystem: Every instance of the cryptosystem is essentially as hard to solve as uSVP in the worst-case, which is conjectured to be extremely hard.
Further, to our knowledge, there is no quantum algorithm for reducing lattices with a nameable advantage, while there are quantum algorithms able to solve the underlying problems of many other cryptosystems used today, e.g. Shors algorithm for prime-factoring and discrete logarithm in polynomial time with a quantum computer [Sho97]. These two facts make Ajtai-Dwork cryptosystem extremely interesting in a theoretical point of view. Unfortunately the original proposed cryptosystem is extremely inefficient.

3.1 Description of the System

Let $n \in \mathbb{N}$. We call n the **security parameter** because it determines the dimension of the vector space we are using for the lattice and also the precision of the binary expansion for real numbers. Given n, define $m := n^3$ and $\rho_n := 2^{n \log_2 n} = n^n$.

Notation 4. Let $\alpha < \frac{1}{2}$. In the following, $\mathbb{Z}_{\pm\alpha}$ denotes the set

$$\bigcup_{z \in \mathbb{Z}} [z - \alpha, z + \alpha].$$

Definition 33. The (big) n-dimensional cube of side-length ρ_n is defined by

$$B_n := \left\{ \boldsymbol{x} \in \mathbb{R}^n \,\middle|\, \forall i \in \{1, \ldots, n\} : -\frac{\rho_n}{2} \leq x_i \leq \frac{\rho_n}{2} \right\}$$

and the (small) n-dimensional ball of radius n^{-c} by

$$S_n := \left\{ \boldsymbol{x} \in \mathbb{R}^n \,\middle|\, \|\boldsymbol{x}\| \leq n^{-c} \right\}$$

for an integer $c > 0$ ($c = 8$ in the original paper [AD97]).

Private Key

The private key is a uniformly chosen vector $\boldsymbol{u}$ in the n-dimensional unit ball

$$U_n := \{\boldsymbol{x} \in \mathbb{R}^n \mid \|\boldsymbol{x}\| < 1\}$$

Public Key

Given the private key $\boldsymbol{u}$, we denote by $\mathcal{H}_u$ the *distribution on points* in B_n induced by the following construction:

- Pick a point $a \in \{\boldsymbol{x} \in B_n \mid \langle \boldsymbol{x}, \boldsymbol{u} \rangle \in \mathbb{Z}\}$ uniformly at random
- Select $\boldsymbol{\delta_1}, \ldots, \boldsymbol{\delta_n}$ uniformly at random from S_n
- Output the point $\boldsymbol{v} = \boldsymbol{a} + \sum_{i=1}^{n} \boldsymbol{\delta_i}$

So, $\mathcal{H}_u$ can be seen as the set $\{\boldsymbol{a} + \sum_{i=1}^{n} \boldsymbol{\delta_i} \mid \langle \boldsymbol{a}, \boldsymbol{u} \rangle \in \mathbb{Z}, \boldsymbol{\delta_i} \in S_n\}$. The public key is obtained by picking points $(\boldsymbol{w_1}, \ldots, \boldsymbol{w_n})$ and $(\boldsymbol{v_1}, \ldots, \boldsymbol{v_m})$ independently at random from the distribution $\mathcal{H}_u$ with constraint that the width of the parallelepiped $\mathcal{P}(\boldsymbol{w_1}, \ldots, \boldsymbol{w_n})$ is at least $n^{-2}\rho_n$.

Encryption

The encryption is bit-wise and depends whether it currently deals with a 0 or a 1. To encrypt a 0, uniformly select $b_1, \ldots, b_m \in \{0, 1\}$ and reduce the vector $\sum_{i=1}^{m} b_i \cdot \boldsymbol{v_i}$ modulo the parallelepiped $\mathcal{P}(\mathbf{W})$. The vector obtained is the ciphertext.
To encrypt a 1, simply choose a vector in the parallelepiped $\mathcal{P}(\mathbf{W})$ as ciphertext at random.

Decryption

The decryption is also bit-wise. To decrypt a component $\boldsymbol{c_i}$ from the message $\boldsymbol{c} = (\boldsymbol{c_1}, \ldots, \boldsymbol{c_k})$ with the private key $\boldsymbol{u}$, we first have to compute $\tau := \langle \boldsymbol{c_i}, \boldsymbol{u} \rangle$. If $\tau \in \mathbb{Z}_{\pm n^{-1}}$, then $\boldsymbol{c_i}$ is decrypted as a 0, and otherwise as a 1.

Lemma 15. Encryption of a 0 will always be decrypted as a 0, and encryption of a 1 has a probability of $2 \cdot n^{-1}$ to be decrypted as a 0.

Proof. Let $\boldsymbol{c}$ be an encrypted 0, i.e. $\boldsymbol{c} = \sum_{i=1}^{m} b_i \cdot \boldsymbol{v_i} \mod \mathcal{P}(\mathbf{W})$ and for every $\boldsymbol{v_i}$, $\langle \boldsymbol{v_i}, \boldsymbol{u} \rangle \in \mathbb{Z}_{\pm n^{-c}}$ with $c = 8$. So, $\sum_{i=1}^{m=n^3} b_i \cdot \boldsymbol{v_i} \in \mathbb{Z}_{\pm n^{3-c}}$. Since this vector is taken modulo the parallelepiped, this leads to: $\boldsymbol{c} = \sum_{i=1}^{m} b_i \cdot \boldsymbol{v_i} + \sum_{i=1}^{m} a_i \cdot \boldsymbol{w_i}$ with $a_i < n^4$. So $\boldsymbol{c} \in \mathbb{Z}_{\pm n^{-1}}$. On the other hand, let $\boldsymbol{c}$ be an encrypted 1. This means, that $\boldsymbol{c} \in \mathcal{P}(\mathbf{W})$ has been chosen uniformly at random. Consider $t := \langle \boldsymbol{c}, \boldsymbol{u} \rangle - \lfloor \langle \boldsymbol{c}, \boldsymbol{u} \rangle \rfloor$, which is therefore uniform in $[0, 1]$. So the probability that $t < n^{-1}$ or $t > n - n^{-1}$ is $2 \cdot n^{-1}$. □

Example 6 (Ajtai-Dwork Cryptosystem)**.** This example shows the generation of the private and public key in the Ajtai-Dwork cryptosystem and afterwards an encryption as well as the decryption of a message. Because of the expected error rate of $2/n$, we have chosen n to be 4, even if the public key becomes very large ($(4 + 4^3)$ 4-dimensional vectors). For the constant c we chose 8 as in the original paper [AD97].
Figure 3.1 shows a randomly chosen vector in the 4-dimensional unit ball. This vector serves as the private key. It determines uniquely the distribution $\mathcal{H}_{\boldsymbol{u}}$ as described in Section 3.1.

$$\boldsymbol{u} = (0.057,\ 0.357,\ -0.322,\ -0.107)$$

Figure 3.1: A Private Key $\boldsymbol{u}$ in ADCS

The side-length ρ_n of the (big) n-dimensional cube B_n is $n^n = 4^4 = 256$. This means that all components of vectors in the public key $(\mathbf{W}, \mathbf{V})$ are bounded by $\pm\rho_n/2 = \pm 128$ (ignoring the distortion of at most $n \cdot n^{-8} = 4^{-7} = 1/16384 \approx 0.000061$). Figure 3.2 shows the vectors $(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n)$ and $(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m)$ displayed in matrix form:

$$\mathbf{W} = \begin{pmatrix} — & \boldsymbol{w}_1 & — \\ & \vdots & \\ — & \boldsymbol{w}_4 & — \end{pmatrix} \qquad \mathbf{V} = \begin{pmatrix} — & \boldsymbol{v}_1 & — \\ & \vdots & \\ — & \boldsymbol{v}_{64} & — \end{pmatrix}$$

The width of the parallelepiped $\mathcal{P}(w_1, \ldots, w_4)$ must not be too flat. For a lower bound, the Ajtai-Dwork cryptosystem provide $n^{-2} \cdot \rho_n = 4^2 = 16$. We computed the width vector $\boldsymbol{d} = (d_1, \ldots, d_4)$, where d_i means, the distance of $\boldsymbol{w_i}$ to the hyperplane spanned by $\boldsymbol{w_j}$ for $j \in \{1, \ldots, 4\} \setminus i$:

$$\boldsymbol{d} = (52.301,\ 61.405,\ 34.091,\ 122.295)$$

Figure 3.3: Width Vector of $\mathcal{P}(\boldsymbol{w_1}, \ldots, \boldsymbol{w_4})$

$$\mathbf{W} = \begin{pmatrix} 111.361 & -72.959 & 25.58 & -64.791 \\ 1.864 & 96.715 & 31.943 & 12.49 \\ -104.582 & -2.665 & 2.287 & 22.324 \\ 109.891 & 39.727 & -120.229 & -92.607 \end{pmatrix}$$

$$\mathbf{V} = \begin{pmatrix} 85.242 & 37.897 & 63.975 & -39.374 \\ 21.721 & -116.605 & -27.26 & 96.742 \\ -48.389 & -73.061 & -28.635 & -14.87 \\ 95.232 & -13.163 & -116.578 & 48.234 \\ -79.08 & -125.434 & -69.492 & 84.906 \\ -69.66 & -74.109 & -76.283 & 29.308 \\ 40.933 & -13.68 & 65.838 & -118.605 \\ -103.457 & -70.307 & 69.297 & -104.5 \\ 64.115 & -83.051 & 56.104 & 83.357 \\ 108.67 & -47.566 & 89.764 & -90.232 \\ -80.441 & -34.243 & -116.846 & 119.248 \\ 37.821 & -31.771 & -95.945 & -3.188 \\ -73.029 & 49.057 & 36.358 & 15.645 \\ 87.477 & 63.015 & -82.902 & 47.413 \\ 54.512 & -51.872 & -110.24 & 74.738 \\ 111.865 & -69.664 & -19.465 & 109.24 \\ -106.326 & 66.969 & -47.131 & 65.516 \\ -40.516 & 55.266 & -97.484 & -104.514 \\ 50.817 & 63.022 & 12.797 & 20.916 \\ -59.21 & 74.23 & 27.124 & 87.656 \\ 48.574 & -16.637 & 59.25 & -104.678 \\ -119.701 & -75.584 & 44.751 & 73.271 \\ -11.676 & 104.07 & 60.054 & -45.063 \\ -109.551 & 38.879 & -54.378 & -35.74 \\ 12.465 & -44.209 & -73.588 & -3.764 \\ -118.457 & 118.053 & -102.574 & 13.095 \\ 93.18 & -6.413 & -12.7 & 65.807 \\ 54.457 & 86.477 & -69.693 & -117.811 \\ 0.705 & 15.366 & -38.739 & -111.922 \\ 94.225 & 72.436 & -47.15 & 49.711 \\ 109.809 & 75.467 & -36.755 & -130.75 \\ -92.609 & -106.799 & -11.413 & 77.518 \\ -85.594 & 37.664 & -65.0 & 116.387 \\ 84.133 & 118.504 & 123.992 & -35.672 \\ -27.443 & 47.063 & 60.23 & -47.758 \\ 72.074 & -87.65 & 28.96 & -23.366 \\ 5.251 & 9.904 & 92.791 & 9.229 \\ 51.861 & 86.912 & 79.441 & 68.885 \\ -69.658 & 69.729 & -108.008 & -96.17 \\ 85.361 & 102.322 & 77.271 & -4.76 \\ -17.866 & -65.033 & -45.604 & 32.13 \\ -55.664 & -48.167 & 37.623 & -106.467 \\ 42.958 & -124.148 & 84.047 & -101.439 \\ 36.929 & 19.905 & -124.43 & 104.338 \\ 65.68 & -54.71 & 101.684 & -116.436 \\ 76.156 & 126.223 & -90.563 & -42.197 \\ 35.729 & 59.873 & -19.076 & -13.792 \\ 68.928 & 9.493 & -30.876 & -44.548 \\ -106.055 & -40.397 & -103.971 & -120.854 \\ 48.391 & 61.7 & 78.539 & 32.553 \\ -12.036 & 86.799 & 5.46 & 51.523 \\ -64.135 & -126.793 & -106.943 & 51.424 \\ -99.287 & 53.137 & 127.842 & -100.271 \\ -114.689 & -18.972 & 13.81 & 40.056 \\ 92.947 & 33.64 & -26.955 & 36.584 \\ 84.941 & 111.732 & 54.346 & 122.871 \\ 97.883 & 10.836 & -104.588 & -8.969 \\ 55.356 & -90.582 & 43.981 & -12.4 \\ 33.171 & 39.711 & -77.389 & 8.683 \\ -100.729 & 111.406 & -26.696 & -12.694 \\ -9.498 & 25.471 & -120.0 & 75.799 \\ -77.281 & 74.703 & -37.413 & 68.098 \\ -0.406 & 7.91 & -93.662 & -112.492 \\ 23.686 & 81.938 & 97.471 & -25.782 \end{pmatrix}$$

Figure 3.2: A Public Key $(\mathbf{W}, \mathbf{V})$ in ADCS

The width of $\mathcal{P}(\boldsymbol{w_1}, \ldots, \boldsymbol{w_4})$, i.e. the minimum of $\boldsymbol{d}$, is 34.091 which satisfies the lower bound of 16. Otherwise new vectors $\boldsymbol{w_i}$'s had to be generated.
In the next step we encrypt the message $\boldsymbol{m} = (0, 1, 0, 1, \ldots)$, the alternating sequence of length 10. As described in Section 3.1 the 0's are encrypted as the sum of a subset of the vectors in $\boldsymbol{v}$ modulo the parallelepiped $\mathcal{P}(\boldsymbol{w_1}, \ldots, \boldsymbol{w_4})$, while the 1's are encrypted as randomly chosen vectors in this parallelepiped. The ciphertext $\boldsymbol{c}$ is shown in Figure 3.4 in matrix form. Every row in the matrix represents the ciphertext of a bit in $\boldsymbol{m}$.

$$c = \begin{pmatrix} 45.113 & 27.758 & -35.498 & -44.199 \\ -25.898 & 31.422 & 13.406 & -14.602 \\ 51.912 & 79.297 & -72.033 & -71.605 \\ 96.992 & 66.266 & -17.203 & -68.291 \\ 47.905 & -64.98 & 7.748 & -60.596 \\ 82.344 & -41.258 & -1.563 & -61.008 \\ -8.991 & 48.88 & 6.495 & -29.125 \\ -1.055 & 48.172 & 20.531 & -4.422 \\ 78.646 & 45.443 & -17.913 & -69.818 \\ 62.172 & -14.242 & -23.766 & -64.322 \end{pmatrix}$$

Figure 3.4: A Ciphertext of $\boldsymbol{m}$

Finally, we decrypt $\boldsymbol{c}$ as described in Section 3.1, by taking the scalar product and check for the distance to the next integer:

$$\tilde{m}_i = \begin{cases} 0 & \text{if } \langle \boldsymbol{c_i}, \boldsymbol{u} \rangle \in \mathbb{Z}_{\pm 1/4} \\ 1 & \text{otherwise.} \end{cases}$$

Figure 3.5 shows the vector consisting of the scalar products $\langle \boldsymbol{c_i}, \boldsymbol{u} \rangle$, which leads to the message $\tilde{m} = (1, 0, 0, 0, 1, 0, 0, 0, 0, 0)$. As expected about half of the 1's are decrypted as 0's (3 out of 5), while all 0's are decrypted correctly.

(28.619, 7.003, 62.084, 41.997, −16.459, −3.008, 17.971, 11.001, 33.922, 12.986)

Figure 3.5: Scalar Products of $\boldsymbol{c_i}$'s and $\boldsymbol{u}$

3.2 Eliminating Decryption Errors

In 1997 Goldreich, Goldwasser and Halevi published a modified method to encrypt and decrypt messages in the Ajtai-Dwork cryptosystem which got rid of the disadvantage of decryption errors. In [GGH97a] the authors also proved that the modified method has no impact on the security of the Ajtai-Dwork cryptosystem. In this section, we cover this modifications and the security proofs.
As in the original scheme, the modified scheme maps 0 to a vector $\boldsymbol{x}$ such that $\langle \boldsymbol{x}, \boldsymbol{u} \rangle$ is close to an integer. However, the modified scheme also makes sure that the vectors of encrypted 1's are far from any integer, i.e. they can uniquely be distinguished from encryption of 0's.
The parameters n, m, ρ_n, the sets S_n, B_n and the vectors $\boldsymbol{u}$, $\boldsymbol{w_1}, \ldots, \boldsymbol{w_n}$, $\boldsymbol{v_1}, \ldots, \boldsymbol{v_m}$ are chosen in exactly the same manner as in the original system.

Public Key (modified)

The first modification is in the public key. Additionally to the original public key $(\boldsymbol{w}, \boldsymbol{v})$, an index i_1 is uniformly chosen from all indices i which satisfies the following condition:

$$\boldsymbol{v_i} = \boldsymbol{a_i} + \sum_{j=1}^{n} \boldsymbol{\delta_j} \text{ such that } \langle \boldsymbol{a_i}, \boldsymbol{u} \rangle \in 2\mathbb{Z} + 1, \boldsymbol{\delta_j} \in S_n. \tag{3.1}$$

where $2\mathbb{Z}+1$ denotes all odd integers. Such an index i_1 exists with a probability of $1-2^{-m}$. The public key is then $(\boldsymbol{w}, \boldsymbol{v}, i_1)$.

Encryption (modified)

Encryption of a zero is done exactly like in the original method, by uniformly selecting $b_1, \ldots, b_m \in \{0, 1\}$ and reducing the vector $\sum_{i=1}^{m} b_i \cdot \boldsymbol{v_i}$ modulo the parallelepiped $\mathcal{P}(\boldsymbol{w})$. The vector

$$\boldsymbol{c_i} = \sum_{i=1}^{m} b_i \cdot \boldsymbol{v_i} \mod \mathcal{P}(\boldsymbol{w})$$

is the ciphertext which corresponds to zero.
The difference lies in the encryption of a 1. Instead of selecting a random vector in $\mathcal{P}(\boldsymbol{w})$, $b_1, \ldots, b_m \in \{0, 1\}$ are chosen uniformly and the vector $\frac{1}{2}\boldsymbol{v_{i_1}} + \sum_{i=1}^{m} b_i \cdot \boldsymbol{v_i}$ is reduced modulo the parallelepiped. The vector

$$\boldsymbol{c_i} = \frac{1}{2}\boldsymbol{v_{i_1}} + \sum_{i=1}^{m} b_i \cdot \boldsymbol{v_i} \mod \mathcal{P}(\boldsymbol{w})$$

is the ciphertext which corresponds to a 1.

Decryption (modified)

The decryption of a vector $\boldsymbol{c_i}$ corresponding to an encrypted bit $\sigma \in \{0,1\}$ is done as follows:

$$\sigma = \begin{cases} 0 & \text{if } \langle \boldsymbol{c_i}, \boldsymbol{u} \rangle \in \mathbb{Z}_{\pm 1/4} \\ 1 & \text{otherwise.} \end{cases} \tag{3.2}$$

With the modified encryption it is now guaranteed that encryption of ones satisfies $\langle \boldsymbol{x}, \boldsymbol{u} \rangle \in \mathbb{Z} + \frac{1}{2} \pm \frac{1}{n}$ which is shown in the following

Proposition 16 (error-free decryption)**.** Let $c = 8$ as in the original paper of Ajtai and Dwork [AD97]. For every $\sigma \in \{0,1\}$, every choice of the private key and the public key, and every choice of b_i's by the encryption algorithm, the ciphertext $\boldsymbol{x}$ satisfies

$$\langle \boldsymbol{x}, \boldsymbol{u} \rangle \in \mathbb{Z} + \frac{\sigma}{2} \pm \frac{1}{n^3}$$

Especially, the decryption is error-free for dimension $n \geq 2$.

Proof. Let $\boldsymbol{x}$ be an encrypted 0, i.e.

$$\boldsymbol{x} = \sum_{i=1}^{m} b_i \cdot \boldsymbol{v_i}$$

with $b_i \in \{0,1\}$, $\boldsymbol{v_i} \in \mathcal{H}_{\boldsymbol{u}}$ $\forall i \in \{1, \ldots, m\}$. Recall, that

$$\mathcal{H}_u = \left\{ \boldsymbol{a} + \sum_{i=1}^{n} \delta_i \,\middle|\, \langle \boldsymbol{a}, \boldsymbol{u} \rangle \in \mathbb{Z}, \boldsymbol{\delta_i} \in S_n \right\}$$

So, $\boldsymbol{x} = \sum\limits_{i=1}^{m} b_i \cdot \left(\boldsymbol{a_i} + \sum_{j=1}^{n} \delta_{ij} \right)$.

$$\begin{aligned} \langle \boldsymbol{x}, \boldsymbol{u} \rangle &= \left\langle \sum_{i=1}^{m} b_i \cdot \left(\boldsymbol{a_i} + \sum_{j=1}^{n} \boldsymbol{\delta_{ij}} \right), \boldsymbol{u} \right\rangle \\ &= \left\langle \sum_{i=1}^{m} b_i \cdot \boldsymbol{a_i} + \sum_{i=1}^{m} \sum_{j=1}^{n} \boldsymbol{\delta_{ij}}, \boldsymbol{u} \right\rangle \\ &= \underbrace{\left\langle \sum_{i=1}^{m} b_i \cdot \boldsymbol{a_i}, \boldsymbol{u} \right\rangle}_{\in \mathbb{Z}} + \underbrace{\left\langle \sum_{i=1}^{m} \sum_{j=1}^{n} \boldsymbol{\delta_{ij}}, \boldsymbol{u} \right\rangle}_{(*)} \end{aligned} \tag{3.3}$$

Consider $(*)$: $\boldsymbol{\delta_{ij}} \in S_n = \{\boldsymbol{x} \in \mathbb{R}^n \mid \|\boldsymbol{x}\| \leq n^{-c}\}$
So,

$$\left\| \sum_{i=1}^{m} \sum_{j=1}^{n} \boldsymbol{\delta_{ij}} \right\| \leq m \cdot n \cdot n^{-c} = n^3 \cdot n \cdot n^{-c} = n^{4-c} = n^{-4} \text{ for } c = 8$$

Therefore also $-n^{-4} \leq (*) \leq n^{-4} \Rightarrow \langle \boldsymbol{x}, \boldsymbol{u} \rangle \in \mathbb{Z}_{\pm n^{-4}}$.
For $\boldsymbol{x}$ an encrypted 1, Equation (3.3) looks like this:

$$\langle \boldsymbol{x}, \boldsymbol{u} \rangle = \underbrace{\left\langle \frac{1}{2} \cdot \boldsymbol{a_{i_1}} \sum_{i=1}^{m} b_i \cdot \boldsymbol{a_i}, \boldsymbol{u} \right\rangle}_{\in \mathbb{Z} + \frac{1}{2}} + \underbrace{\left\langle \sum_{j=1}^{n} \boldsymbol{\delta_{i_1 j}} + \sum_{i=1}^{m} \sum_{j=1}^{n} \boldsymbol{\delta_{ij}}, \boldsymbol{u} \right\rangle}_{(*')}$$

$(*') \leq (*) + n \cdot n^{-c} = n^{(4-c)} + n^{(1-c)} \leq n^{-3}$ □

Example 7 (Error-less en-/decryption)**.** In this example we show the error-less encryption and decryption on the private and public key we took in Example 6. i_1 is the index of a vector we randomly chose from all vectors in $\boldsymbol{v}$ satisfying (3.1), in this case $i_1 = 46$, so

$$\boldsymbol{v_{i_1}} = \boldsymbol{v_{46}} = (35.729,\ 59.873,\ -19.076,\ -13.792)$$

where the scalar product $\langle \boldsymbol{v_{46}}, \boldsymbol{u} \rangle = 31$. The public key for the modified system is therefore $(\boldsymbol{w}, \boldsymbol{v}, i_1)$.
Figure 3.6 shows the ciphertext of m in the error-less setting. For demonstration purposes the seed for the random number generator was set the same as in the last example. So it can be seen, that every second row is exactly the same as in Figure 3.4, because these rows correspond to encryptions of zeros, which is done the exact same way in both systems.

$$c = \begin{pmatrix} 62.383 & 7.759 & 10.547 & -42.668 \\ -25.898 & 31.422 & 13.406 & -14.602 \\ 63.869 & 36.25 & -12.141 & -46.984 \\ 96.992 & 66.266 & -17.203 & -68.291 \\ 77.922 & 66.193 & -75.883 & -75.188 \\ 82.344 & -41.258 & -1.563 & -61.008 \\ 7.344 & 42.938 & -69.348 & -50.867 \\ -1.055 & 48.172 & 20.531 & -4.422 \\ 14.391 & 35.684 & -32.422 & -51.484 \\ 62.172 & -14.242 & -23.766 & -64.322 \end{pmatrix}$$

Figure 3.6: Ciphertext of $\boldsymbol{m}$ in the Error-Less Method

The scalar product of every row with the private key $\boldsymbol{u}$ gives the vector:

$$(7.488,\ 7.003,\ 25.498,\ 41.997,\ 60.504,\ -3.008,\ 43.5,\ 11.001,\ 29.502,\ 12.986)$$

which clearly corresponds to the original message $\boldsymbol{m}$ when applying the modified decryption seen in (3.2).

3.3 Practicality

3.3.1 Space and Time Analysis

Recall that n is apart from the dimension of the lattice also the precision of the binary expansion.
The private key is simply a random vector in the n-dimensional unit-ball with precision of n bits. This gives an amount of n^2 bits.
For the public key, we first compute the space an element $\boldsymbol{v} \in \mathcal{H}_u$ needs to be stored in a binary representation. This n-dimensional vector is of the form $\boldsymbol{v} = \boldsymbol{a} + \sum_{i=1}^{n} \boldsymbol{\delta_i}$. So, the components of the vector are bounded by $-\rho_n/2 - n \cdot n^{-c} \leq |v_i| \leq \rho_n/2 + n \cdot n^{-c}$. This means every component of the vector needs $\log_2 \rho_n = \log_2 n^n = n \cdot \log_2 n$ bits plus n bits for the binary expansion, so $n \cdot (\log_2 n + 1)$. In total a n-dimensional vector $\boldsymbol{v} \in \mathcal{H}_u$ needs $n^2 \cdot (1 + \log_2 n)$ bits. The public key $(\boldsymbol{w_1}, \ldots, \boldsymbol{w_n}, \boldsymbol{v_1}, \ldots, \boldsymbol{v_{n^3}})$ consists of $n + n^3$ vectors from $\mathcal{H}_u$. So it consumes $(n + n^3) \cdot (n^2 \cdot (1 + \log_2 n)) = (n^3 + n^5) \cdot (1 + \log_2 n)$ bits.
The encryption of one bit is a vector in the parallelepiped $\mathcal{P}(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n)$, which is bounded by: $\forall i \in \{1, \ldots, n\} : v_i \leq \rho_n$. In total this gives $n^2 \cdot (1 + \log_2 n)$.

Space	
Object	Size
Private Key	$\mathcal{O}\left(n^2\right)$
Public Key	$\mathcal{O}\left(n^5\right)$
Ciphertext	$\mathcal{O}\left(n^2 \log n\right)$

Time	
Operation	Time
Private Key Generation	$\mathcal{O}\left(n\right)$
Public Key Generation	$\mathcal{O}\left(n^5\right)$
Encryption	$\mathcal{O}\left(n^4\right)$
Decryption	$\mathcal{O}\left(n^2\right)$

Figure 3.7: Space and Time Complexities of ADCS

Generating the private key could be done in linear time, if we assume that the random values can be taken from a precomputed set of random values in constant time. For the public key we have to generate n^3 elements from the distribution $\mathcal{H}_u$, which self takes $\mathcal{O}\left(n^2\right)$ operations, so the whole step needs $\mathcal{O}\left(n^3 \cdot n^2\right) = \mathcal{O}\left(n^5\right)$ time. Encryption consists of n^3 vector additions and a modulo operation. This can be done in $\mathcal{O}\left(n^4\right)$ steps. Decryption is simply taking a scalar product of two vectors, which consists of n multiplications and a summation of n values, so this step takes $\mathcal{O}\left(n^2\right)$ time.
All the space and time complexities are summarized in Figure 3.7.

3.3.2 Practical Tests

For measuring the practicality of the Ajtai-Dwork cryptosystem, we performed the following experiments. For dimensions 4 to 48 we generated 10 key pairs per dimension and

encrypted for every pair 10 messages of 100bit length.
The generation of the private key is instantly (durations are all under 10ms). Figure 3.8 shows the time for generating the public key. It can be seen, that for dimension 48, it needed about 2 hours to complete.

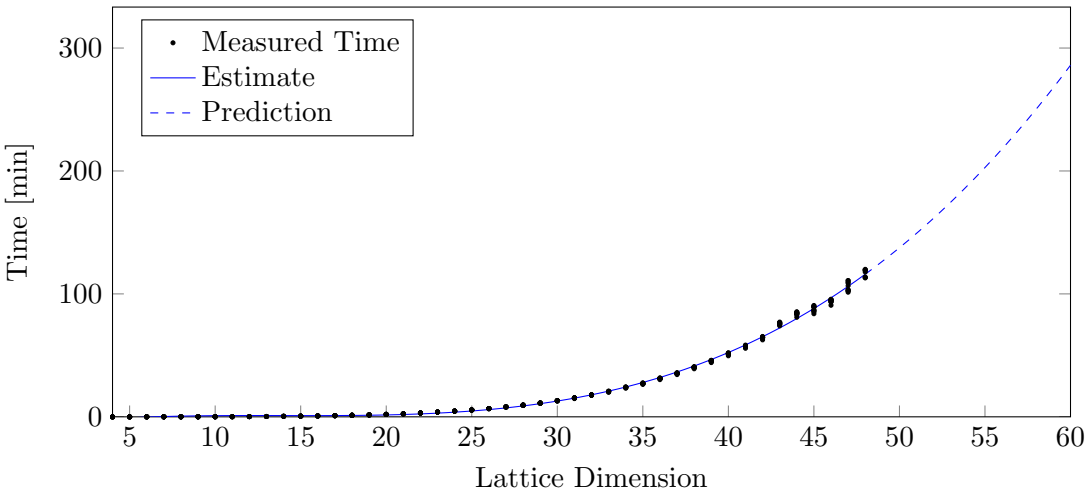

Figure 3.8: Time for Generating a Public Key

Figure 3.9 shows the time for encrypting a 100bit message. The time for encrypting such a message was about 1 minute in dimension 48. Figure 3.10 shows the time for decrypting the ciphertext generated before, one point is the mean of 100 measured decryption times.

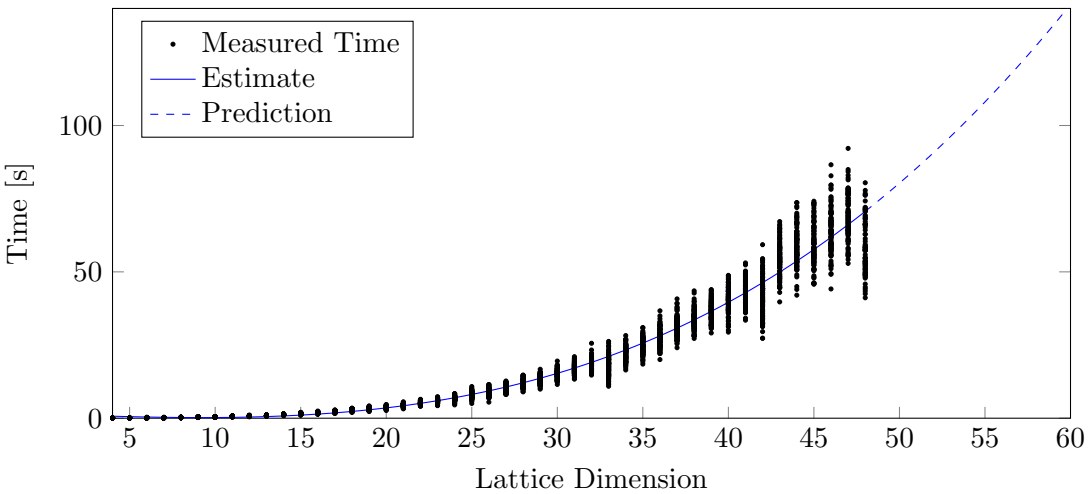

Figure 3.9: Time for Encrypting a 100Bit Message in ADCS

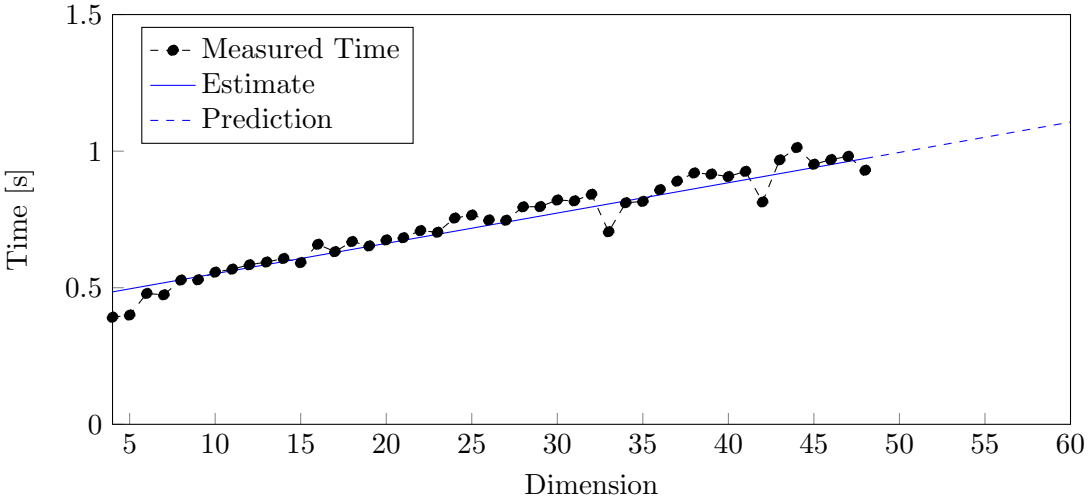

Figure 3.10: Time for Decrypting a 100Bit Message in ADCS

We also measured the decryption failures, which are shown in Figure 3.11. It shows the average number of bits which are decrypted incorrectly. As expected they are about $1/n$, for dimension n.

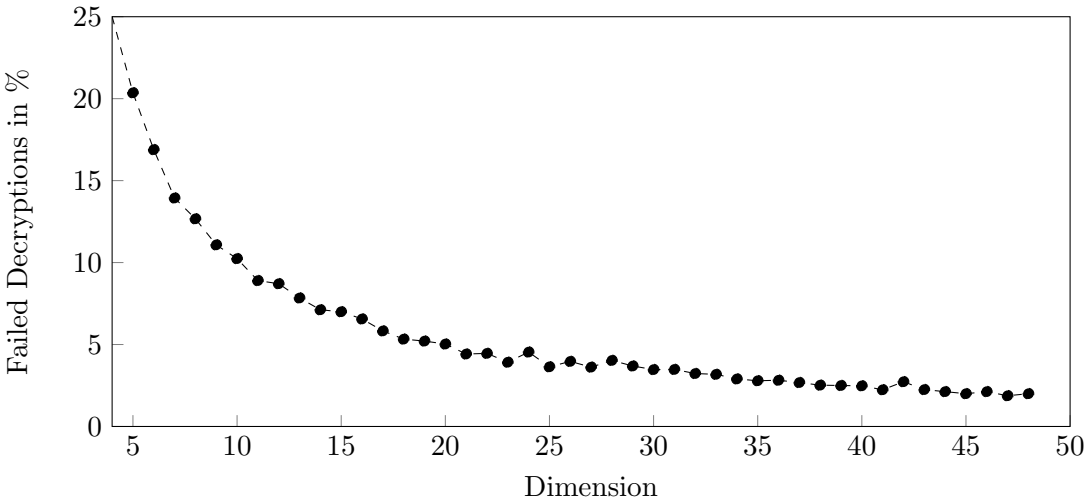

Figure 3.11: Failed Decryptions

Key and Ciphertext Size

In the following, the key and ciphertext sizes can be seen. These are first stored as text, which was then compressed with gzip at compression ratio of 9 (best compression for gzip). The sizes we show are the file sizes of the compressed files.

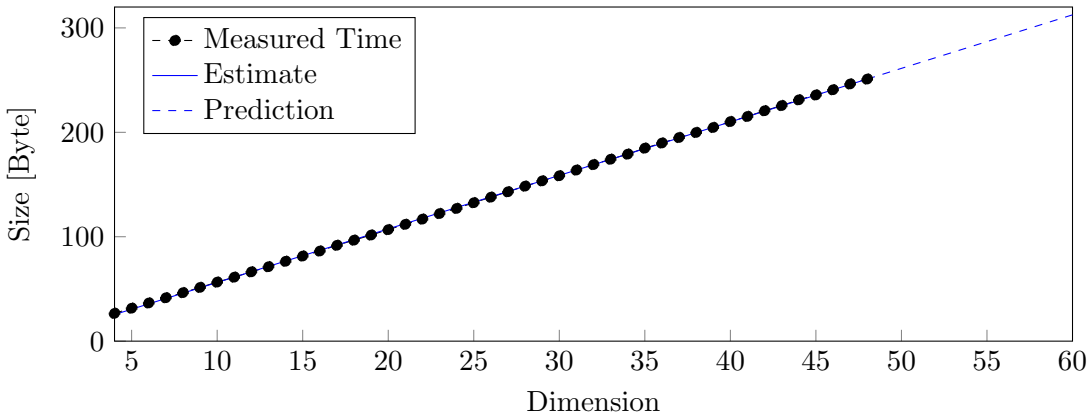

Figure 3.12: Size of a Private Key in ADCS

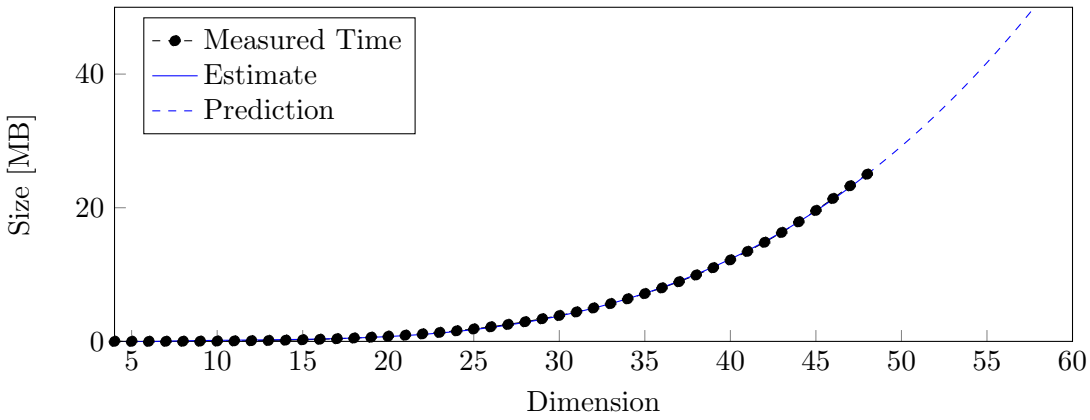

Figure 3.13: Size of a Public Key in ADCS

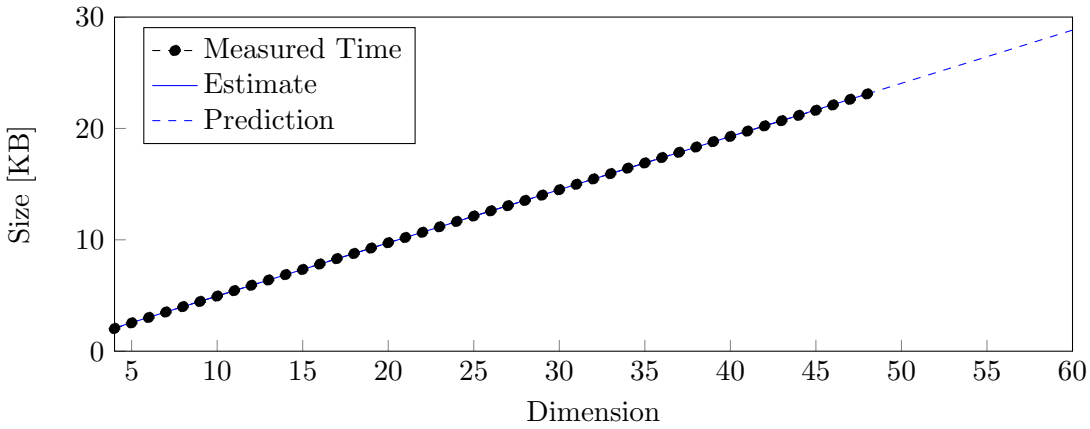

Figure 3.14: Size of a Encrypted 100Bit Message in ADCS

3.4 Conclusion

In this section we gave an overview of one of the cryptosystems of Ajtai and Dwork. The system makes use of the worst-case/average-case equivalence Ajtai found one year before, which in its simplest form states that, if one is able to break the cryptosystem in polynomial time, then one is also able to solve the following famous problems in lattices, namely approximating the shortest vector, finding the unique shortest vector and find a basis for the lattice which is the smallest possible up to a polynomial factor. All of these problems are well studied problems in this field. Ajtai's work is absolutely brilliant, but unfortunately the system is not really practical in dimensions where these three problems are infeasible to solve in a useful amount of time. In our experiments, the key generation in dimension 48 went about two full hours, which could be tolerated in very special areas, where fast key generations are not the major concern, but in this dimension 1 bit is encrypted to about 2000 bits even after compression with today's state of the art compression algorithms. In our opinion this is the major drawback of this system, since a simple file of 1MB size needs 2GB of space when encrypted. Nevertheless, the discovery of the worst-case/average-case equivalence and the invention of the Ajtai-Dwork cryptosystem was a huge milestone in the study of new fields for doing cryptography. Indeed, Ajtai's work encouraged many researcher to investigate in cryptosystems based on lattices.

Chapter 4

Attack by Nguyen and Stern

A year after Ajtai and Dwork published the cryptosystem based on the worst-case/average-case equivalence, Nguyen and Stern came with a converse to the Ajtai-Dwork security result of the cryptosystem ([NS98] and [NS99]), by reducing the question of distinguishing encryptions of ones from encryptions of zeros to approximating SVP and CVP. This implies that breaking the Ajtai-Dwork cryptosystem is unlikely to be NP-hard. Further, in [NS98] Nguyen and Stern also presented a heuristic attack to recover the private key.
In this section we describe the attack on the private key of the Ajtai-Dwork cryptosystem by Nguyen and Stern, presented in [NS98]. The structure is the following: First, we give the idea behind the attack followed by justifications why it works. After that, we continue the example from the last chapter by showing the attack on it.

4.1 Attack on the Private Key

We assume that all the parameters for the cryptosystem are known and that we are in possession of the public key $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$. Recall, that $\langle \boldsymbol{v_i}, \boldsymbol{u} \rangle \in \mathbb{Z}_{\pm n^{-7}}$ for all $i \in \{1, \ldots, m\}$ by construction. We define V_i to be the closest integer to $\langle \boldsymbol{v_i}, \boldsymbol{u} \rangle$. The goal is now to find at least n such V_i's to solve the (over specified) linear equation:

$$\begin{pmatrix} — & \boldsymbol{v_1} & — \\ — & \boldsymbol{v_2} & — \\ & \vdots & \\ — & \boldsymbol{v_m} & — \end{pmatrix} \begin{pmatrix} \boldsymbol{u_1} \\ \boldsymbol{u_2} \\ \vdots \\ \boldsymbol{u_n} \end{pmatrix} = \begin{pmatrix} \boldsymbol{V_1} \\ \boldsymbol{V_2} \\ \vdots \\ \boldsymbol{V_m} \end{pmatrix} \tag{4.1}$$

and get an accurate approximation of the private vectors $\boldsymbol{u_1}, \ldots, \boldsymbol{u_n}$.
For this we first construct a lattice Λ_β only depending on the public key and a constant $0 < \beta \in \mathbb{R}$, which we describe later how to pick.

Definition 34. Let Λ_β be the lattice spanned by the columns of the following $(n+m)\times m$ matrix:

$$M := \begin{pmatrix} \beta\boldsymbol{v_1} & \beta\boldsymbol{v_2} & \dots & \beta\boldsymbol{v_m} \\ 1 & 0 & \dots & 0 \\ 0 & 1 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & 1 \end{pmatrix} \tag{4.2}$$

Nguyen and Stern showed that short vectors in Λ_β give information on the V_i's. More precisely, let $\boldsymbol{x}^T = (\beta(\lambda_1\boldsymbol{v_1} + \dots + \lambda_m\boldsymbol{v_m}), \lambda_1, \dots, \lambda_m) \in \Lambda_\beta$, i.e. $\lambda_i \in \mathbb{Z}$. If this vector is short enough, they conclude that the sum $\sum_{i=1}^m \lambda_i V_i$ must be 0 (we show this in Theorem 17). This means that the vector $(\lambda_1, \dots, \lambda_m)$ is orthogonal to $\boldsymbol{V} = (V_1, \dots, V_m)$. Corollary 19 shows, that there are many such short vectors. It may be assumed that there are at least $(m-1)$ such short and linearly independent vectors, which we denote by $\boldsymbol{\lambda_i}$.
The lattice Λ_V spanned by the m-dimensional vector $\boldsymbol{V}$ is 1-dimensional. Consider its dual lattice in $\mathbb{Z}^m$, denoted by $\Lambda_V^\star$, which is $(m-1)$-dimensional. $\Lambda_V^\star$ can be generated by $(m-1)$ independent vectors $\boldsymbol{\lambda_i}$. So if one knows $(m-1)$ independent vectors $\boldsymbol{\lambda_i}$, one can determine the one-dimensional dual lattice of $\Lambda_V^\star$, i.e. one can find a vector $\boldsymbol{V'}$ generating the $\Lambda_V^{\star\star} \in \mathbb{Z}^m$. There exists $\varepsilon \in \mathbb{Z}$, such that $\boldsymbol{V} = \varepsilon\boldsymbol{V'}$. If all the entries of $\boldsymbol{V'}$ are coprime (which happens with overwhelming probability), then $\varepsilon = \pm 1$.

4.2 Proofs

Theorem 17. Let $\boldsymbol{x} = (\beta(\lambda_1\boldsymbol{v_1} + \dots + \lambda_m\boldsymbol{v_m}), \lambda_1, \dots, \lambda_m)^T \in \Lambda_\beta$. If

$$n^7 \left\| \sum_{i=1}^m \lambda_i \boldsymbol{v_i} \right\| + \sum_{i=1}^m |\lambda_i| < n^7,$$

then $\sum_{i=1}^m \lambda_i V_i = 0$.
In particular, this equality is satisfied if $\beta^2 \geq \frac{n^14}{2n^7-1}$ and $\|\boldsymbol{x}\| < \frac{n^7}{\sqrt{2n^7-1}}$.

Proof. By definition of the $\boldsymbol{v_i}$'s

$$\left| \left\langle \sum_{i=1}^m \lambda_i \boldsymbol{v_i}, \boldsymbol{u} \right\rangle - \sum_{i=1}^m \lambda_i V_i \right| \leq n^{-7} \sum_{i=1}^m |\lambda_i|$$

If $|\langle \sum_{i=1}^m \lambda_i \boldsymbol{v_i}, \boldsymbol{u}\rangle| < 1 - n^{-7}\sum_{i=1}^m |\lambda_i|$, then the integer $\sum_{i=1}^m \lambda_i V_i$ is 0, since it is strictly less than 1 in the absolute value. As $\|u\| \leq 1$, a stronger condition is

$$\left\| \sum_{i=1}^m \lambda_i \boldsymbol{v_i} \right\| < 1 - n^{-7} \sum_{i=1}^m |\lambda_i|$$

by Cauchy-Schwarz inequality, and this proves the first statement.
For the second statement, we square the last equation and get

$$\left\|\sum_{i=1}^{m}\lambda_i \boldsymbol{v}_{\mathbf{i}}\right\|^2 < 1 + n^{-14}\sum_{i=1}^{m}\lambda_i^2 - 2n^{-7}\sum_{i=1}^{m}|\lambda_i|\,.$$

But $|\lambda_i| \leq \lambda_i^2$ since the λ_i's are integers. This gives a new stronger condition:

$$n^7\left\|\sum_{i=1}^{m}\lambda_i \boldsymbol{v}_{\mathbf{i}}\right\|^2 + (2-n^{-7})\sum_{i=1}^{m}\lambda_i^2 < n^7,$$

which is satisfied as soon as $\beta^2 \geq \frac{n^{14}}{2n^7-1}$ and $\|\boldsymbol{x}\| < \frac{n^7}{\sqrt{2n^7-1}}$. □

Theorem 18. For all $\varepsilon > 0$, there exists N such that the following holds for all $n \geq N$. Let $\{i_1, \ldots, i_{m'}\}$ be a subset of $\{1, \ldots, m\}$. If $m' \geq (1+\varepsilon)n^2 \log_2 n$, then there exists $\lambda_1, \ldots, \lambda_{m'}$ (not all zero) in $\{-1, 0, 1\}$ such that

$$\left\|\lambda_1 \boldsymbol{v}_{\mathbf{i_1}} + \cdots + \lambda_{m'} \boldsymbol{v}_{\mathbf{i_{m'}}}\right\| \leq n^{3.5-n/\log_2 n}.$$

Proof. Let $\alpha = n/\log_2 n$ and $\boldsymbol{z}_{\mathbf{i}} := \lfloor n^\alpha \boldsymbol{v}_{\mathbf{i}} \rfloor$. Each vector $\boldsymbol{z}_{\mathbf{i}}$ has integral entries in the set $\{-n^\alpha \rho_n, \ldots, n^\alpha \rho_n\}$. Consider all combinations of $\boldsymbol{z}_{\mathbf{i_1}}, \ldots, \boldsymbol{z}_{\mathbf{i_{m'}}}$, with coefficients in $\{0, 1\}$. There are $2^{m'}$ such combinations. But there are at most $(2m'n^\alpha \rho_n + 1)^n$ distinct values for such combinations. By the pigeon-hole principle, it follows that if $2^{m'} > (2m'n^\alpha \rho_n + 1)^n$, then there exist $\lambda_1, \ldots, \lambda_{m'}$ (not all zero) in $\{-1, 0, 1\}$, such that $\lambda_1 \boldsymbol{z}_{\mathbf{i_1}} + \cdots + \lambda_{m'} \boldsymbol{z}_{\mathbf{i_{m'}}} = 0$. Hence:

$$\sum_{k=1}^{m}\lambda_k \boldsymbol{v}_{\mathbf{i_k}} = \frac{\sum_{k=1}^{m}\lambda_k(n^\alpha \boldsymbol{v}_{\mathbf{i_k}} - \lfloor n^\alpha \boldsymbol{v}_{\mathbf{i_k}} \rfloor)}{n^\alpha},$$

whose norm is less than $n^{-\alpha}\sum_{k=1}^{m}\sqrt{n} = n^{3.5-n/\log_2 n}$. Furthermore,

$$\begin{aligned}\log_2(2m'n^\alpha \rho_n + 1)^n &\leq n\log_2(2m') + \alpha n \log_2 n + n\log_2 \rho_n + n\log_2(1+1)\\ &\leq 2n + 3n\log_2 n + n^2 + n^2\log_2 n\end{aligned}$$

We conclude since $2n + 3n\log_2 n + n^2 + n^2\log_2 n < (1+\varepsilon)n^2\log_2 n$ for sufficiently large n. □

Corollary 19. For all $\varepsilon > 0$, there exists N such that for all $n \geq N$ and all $\beta > 0$, there exist at least $n^3 - (1+\varepsilon)n^2\log_2 n$ linearly independent lattice points in Λ_β, with norm less than $\sqrt{n^3 + \beta^2 n^{7-2n/\log_2 n}}$.

4.3 Example

Example 8. We continue Example 6 to demonstrate the attack on the private key by Nguyen and Stern. First we have to construct Λ_β. Recall that $n = 4$, so Λ_β is a 68×64-matrix. We chose β to be $\left\lceil \sqrt{n^{14}/(2n^7 - 1)} \right\rceil = 91$. Reducing this lattice leads to the matrix of the form

$$Y = \begin{pmatrix} | & | & & | \\ \boldsymbol{y_1} & \boldsymbol{y_2} & \cdots & \boldsymbol{y_m} \\ | & | & & | \end{pmatrix} \tag{4.3}$$

where we are interested in the $\boldsymbol{y_i}$'s with a certain bound of norm. Theorem 17 states that $\sum_{i=1}^{m} \lambda_i V_i = 0$ if $\beta^2 \geq \frac{n^{14}}{2n^7-1}$ and the norm of $\boldsymbol{y}$ satisfies $\|\boldsymbol{y}\| < \frac{n^7}{\sqrt{2n^7-1}}$. β has been chosen to fulfill the requirement, so we only have to take a look at the norms of $\boldsymbol{y}$, which we displayed in Figure 4.1. As one can see, all vectors except of y_{64} satisfy $\|\boldsymbol{y_i}\| < 91$, so they are usable to compute $\boldsymbol{V} = (V_1, \ldots, V_m)$.

$\|\boldsymbol{y_1}\|$:	5.0	$\|\boldsymbol{y_{17}}\|$:	4.57	$\|\boldsymbol{y_{33}}\|$:	5.25	$\|\boldsymbol{y_{49}}\|$:	4.79
$\|\boldsymbol{y_2}\|$:	5.19	$\|\boldsymbol{y_{18}}\|$:	5.2	$\|\boldsymbol{y_{34}}\|$:	5.93	$\|\boldsymbol{y_{50}}\|$:	5.76
$\|\boldsymbol{y_3}\|$:	4.92	$\|\boldsymbol{y_{19}}\|$:	5.78	$\|\boldsymbol{y_{35}}\|$:	5.43	$\|\boldsymbol{y_{51}}\|$:	5.86
$\|\boldsymbol{y_4}\|$:	5.49	$\|\boldsymbol{y_{20}}\|$:	5.11	$\|\boldsymbol{y_{36}}\|$:	5.64	$\|\boldsymbol{y_{52}}\|$:	5.62
$\|\boldsymbol{y_5}\|$:	4.94	$\|\boldsymbol{y_{21}}\|$:	4.58	$\|\boldsymbol{y_{37}}\|$:	4.88	$\|\boldsymbol{y_{53}}\|$:	5.63
$\|\boldsymbol{y_6}\|$:	4.86	$\|\boldsymbol{y_{22}}\|$:	5.49	$\|\boldsymbol{y_{38}}\|$:	5.42	$\|\boldsymbol{y_{54}}\|$:	5.89
$\|\boldsymbol{y_7}\|$:	5.54	$\|\boldsymbol{y_{23}}\|$:	5.09	$\|\boldsymbol{y_{39}}\|$:	5.66	$\|\boldsymbol{y_{55}}\|$:	6.5
$\|\boldsymbol{y_8}\|$:	5.5	$\|\boldsymbol{y_{24}}\|$:	6.23	$\|\boldsymbol{y_{40}}\|$:	5.86	$\|\boldsymbol{y_{56}}\|$:	4.85
$\|\boldsymbol{y_9}\|$:	5.73	$\|\boldsymbol{y_{25}}\|$:	5.27	$\|\boldsymbol{y_{41}}\|$:	6.04	$\|\boldsymbol{y_{57}}\|$:	5.62
$\|\boldsymbol{y_{10}}\|$:	5.88	$\|\boldsymbol{y_{26}}\|$:	5.44	$\|\boldsymbol{y_{42}}\|$:	5.84	$\|\boldsymbol{y_{58}}\|$:	6.21
$\|\boldsymbol{y_{11}}\|$:	5.47	$\|\boldsymbol{y_{27}}\|$:	6.13	$\|\boldsymbol{y_{43}}\|$:	5.52	$\|\boldsymbol{y_{59}}\|$:	5.85
$\|\boldsymbol{y_{12}}\|$:	5.94	$\|\boldsymbol{y_{28}}\|$:	5.36	$\|\boldsymbol{y_{44}}\|$:	5.03	$\|\boldsymbol{y_{60}}\|$:	6.05
$\|\boldsymbol{y_{13}}\|$:	6.2	$\|\boldsymbol{y_{29}}\|$:	6.13	$\|\boldsymbol{y_{45}}\|$:	5.41	$\|\boldsymbol{y_{61}}\|$:	5.04
$\|\boldsymbol{y_{14}}\|$:	5.62	$\|\boldsymbol{y_{30}}\|$:	5.42	$\|\boldsymbol{y_{46}}\|$:	5.06	$\|\boldsymbol{y_{62}}\|$:	6.22
$\|\boldsymbol{y_{15}}\|$:	6.36	$\|\boldsymbol{y_{31}}\|$:	6.0	$\|\boldsymbol{y_{47}}\|$:	5.61	$\|\boldsymbol{y_{63}}\|$:	5.14
$\|\boldsymbol{y_{16}}\|$:	5.71	$\|\boldsymbol{y_{32}}\|$:	5.47	$\|\boldsymbol{y_{48}}\|$:	6.35	$\|\boldsymbol{y_{64}}\|$:	166.02

Figure 4.1: Table of Norms

Let $\boldsymbol{x_i}$ be the last m entries of the vector $\boldsymbol{y_i}$ for all $i \in \{1, \ldots, 63\}$. Since $\boldsymbol{V'}$ is the 1-dimensional orthogonal complement of the $\boldsymbol{x_i}$'s, we can compute it as follows.

$$\boldsymbol{V'} = \boldsymbol{e_1} \det \mathbf{X^1} - \boldsymbol{e_2} \det \mathbf{X^2} + \boldsymbol{e_3} \det \mathbf{X^3} - \ldots \pm \boldsymbol{e_{m-1}} \det \mathbf{X^{m-1}} \tag{4.4}$$

where $\mathbf{X^i}$ is the $(m-1) \times (m-1)$-matrix:

$$\mathbf{X^i} = \begin{pmatrix} — & \boldsymbol{x_1} & — \\ & \vdots & \\ — & \boldsymbol{x_{m-1}} & — \end{pmatrix} \tag{4.5}$$

without the i-th column. $\boldsymbol{V'}$ is now a multiple of $\boldsymbol{V}$, so there exists a $z \in \mathbb{Z}$, such that $\boldsymbol{V'} = z\boldsymbol{V}$. There is a very small possibility that z is not equals to ± 1, but this can be found out, by taking gcd. With overwhelming probability $z = \pm 1$ which leads to $\pm\boldsymbol{u}$, either way it can be used to decrypt the message correctly. In fact, we don't need all the 64 entries of $\boldsymbol{V}$, 4 are enough. $(V_1, \ldots, V_4)$ is

$$(-2,\ 42,\ 18,\ -33)$$

$(V_1, \ldots, V_4)$ has no common divisor, so $z = \pm 1$. Solving the linear equation system

$$\begin{pmatrix} — & \boldsymbol{v_1} & — \\ & \vdots & \\ — & \boldsymbol{v_4} & — \end{pmatrix} \boldsymbol{u'} = \begin{pmatrix} — & \boldsymbol{V_1} & — \\ & \vdots & \\ — & \boldsymbol{V_4} & — \end{pmatrix},$$

reveals a good approximation of $-\boldsymbol{u}$:

$$(-0.057,\ -0.357,\ 0.322,\ 0.107)$$

The vector $\boldsymbol{u} + \boldsymbol{u'}$ is

$$(-0.00002193,\ 0.00003815,\ 0.00001526,\ 0.00004387)\,,$$

so $\boldsymbol{u'}$ is precise enough to use for decryption.

4.4 Practicality

To see how the attack behaves in practice, we run the following experiment. For dimensions 4 to 11, we generated 10 different key pairs per dimension and performed the attack on it. For the reduction part we used LLL with $\delta = 0.99$ performing a full LLL-reduction. It is not easy to give a good estimation how the algorithm proceeds in higher dimensions, but Nguyen is optimistic that attacks even for $n = 32$ looks feasible [NS99] by performing LLL on random samples of Λ_β until enough relations are found rather than on the full $(m+n) \times m$-lattice basis.
Figure 4.2 shows the timing results for our experiment. In all cases the private key was found and the approximation was good enough to decrypt with.

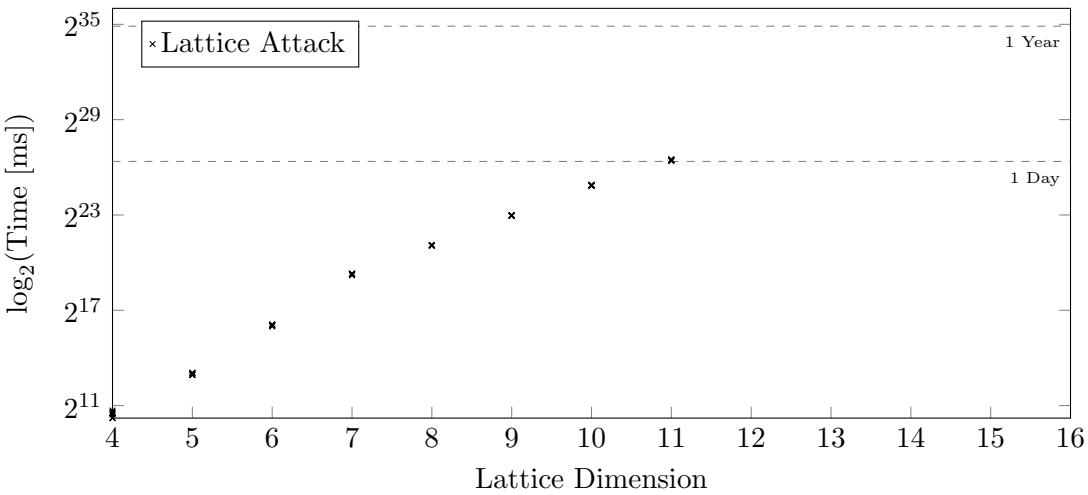

Figure 4.2: Time for a Successful Lattice Attack in ADCS

Back in 1999, Nguyen was able to break the cryptosystem in 3.5 hours for dimension 8. With today's computers we were able to break the system in this dimension in 36 minutes with a unoptimized implementation. So if Nguyen states that it is possible to break the system in dimension 32 in under one year, then this implies, that with today's computers and algorithms it would be possible within 2 months. Since the most time-consuming part is to find enough relations, which can easily be parallelized in the suggested sampling method, this result could even be decreased drastically, depending only on the resources used for the attack. For instance, if we would have four computers with 16 cores each, we should be able to break the system in dimension 32 within one day.

4.5 Conclusion

In this section we covered an attack on the private key in the Ajtai-Dwork cryptosystem introduced by Nguyen and Stern and showed that an attack in dimension 32 looks feasible. This has no effect on the security proof of Ajtai and Dwork, since in these low dimensions, also the lattice problems are feasible to solve in a short amount of time even without parallelization. The long key generation phase, the slow encryption and the huge key sizes in larger dimensions are the reasons, why the system is not practical. In dimensions, where the lattice problems listed in the security proof are difficult to solve, the cryptosystem is far from being practical. We conclude that the Ajtai-Dwork cryptosystem in its original form is not practical even with today's computer power.

Chapter 5

Goldreich-Goldwasser-Halevi

Inspired by the results of Ajtai, Oded Goldreich, Shaft Goldwasser and Shai Halevi [GGH97b] published a cryptosystem based on CVP in 1997. The system is now known as Goldreich-Goldwasser-Halevi cryptosystem or simply GGH. It is very easy to describe and together with the amended cryptosystem by Micciancio the only one discussed in this thesis originally described in terms of lattices.

5.1 Description of the System

The cryptosystem depends on two (public) parameters: $n \in \mathbb{N}$, the dimension and $\sigma \in \mathbb{N}$, a security parameter. In a first step, Bob chooses a random basis $\mathbf{R}$ of short vectors for an n-dimensional lattice. This $\mathbf{R}$ serves as private key. Bob then generates and publishes an other basis $\mathbf{B}$ generating the same lattice, but far not "as reduced as" $\mathbf{R}$ (big orthogonality defect), serving as public key.

Encryption

Alice encodes her message to an element $\boldsymbol{m} \in \mathbb{Z}^n$ and randomly chooses an error-vector $\boldsymbol{e} \in \{\pm\sigma\}^n$ and sends the following ciphertext to Bob

$$\boldsymbol{c} = \mathbf{B} \cdot \boldsymbol{m} + \boldsymbol{e} \tag{5.1}$$

Decryption

Bob computes the original message as follows:

$$\boldsymbol{m} = \mathbf{B}^{-1} \cdot \mathbf{R} \cdot \lfloor \mathbf{R}^{-1} \cdot \boldsymbol{c} \rceil$$

Variation

Alternatively, the message could also be encrypted in the error vector, and the vector $\boldsymbol{m}$ could be randomly chosen. The disadvantage of this method is, that the message space is much smaller since the error vector should not be too large in the L_1-norm to avoid decryption errors (we discuss this in Section 5.1.1).

Remark. This system should not be regarded as a fully determined cryptosystem, but more as a type of system, since many questions are open:

- How to choose $\mathbf{R}$?
- How to generate $\mathbf{B}$ from $\mathbf{R}$?
- How to choose the vector $\boldsymbol{m}$?
- How to choose the vector $\boldsymbol{r}$?

The authors used the following methods to create the challenges: $\mathbf{R} = k \cdot \mathbf{I_n} + \mathbf{Q}$, where $k = \lfloor \sqrt{n} \cdot l \rfloor, l = 4$ and $\mathbf{Q}$ is a random perturbation matrix with entries from $\{-l, \ldots, l\}$. The public key $\mathbf{B}$ was then obtained by applying sufficiently many elementary column operations to $\mathbf{R}$. The message was encrypted with the method just described as variation, particularly they encoded it to a vector $\boldsymbol{r}$ with coefficients $r_i = \pm 3$. The ciphertext is then computed as $\mathbf{B}\boldsymbol{x} + \boldsymbol{r}$, where $\boldsymbol{x}$ is chosen at random from a sufficiently large region in space.

Example 9. This example shows the GGH cryptosystem in action. We chose $n = 5$ and $\sigma = 3$ for demonstration purposes. First step is to randomly choose a "good" basis

$$\mathbf{R} := \begin{pmatrix} 11 & 3 & 0 & -4 & 1 \\ -2 & 9 & 2 & 1 & -1 \\ 3 & -2 & 10 & -1 & -1 \\ 1 & -4 & -2 & 11 & 0 \\ -3 & 1 & 1 & 0 & 11 \end{pmatrix}$$

as private key. This has been done as proposed by the authors: $\mathbf{R} = k \cdot \mathbf{I_n} + \mathbf{Q}$. In order to generate the public key, we built an unimodular matrix $\mathbf{U}$, corresponding of 10 operations of the type "adding an integer multiple of a row to another", where this multiple is chosen randomly from the set $\{1, \ldots, 4\}$ (see Corollary 3 for more detail). To obtain the public

key $\mathbf{B}$, we multiplied $\mathbf{R}$ by $\mathbf{U}$ to get a different basis of the lattice spanned by $\mathbf{R}$:

$$\mathbf{B} = \begin{pmatrix} 81 & 3 & 58 & 26 & 14 \\ 293 & 9 & 222 & 101 & 59 \\ 34 & -2 & 64 & 29 & 6 \\ 52 & -4 & -88 & -39 & 17 \\ 69 & 1 & 33 & 15 & 16 \end{pmatrix}$$

Now, we would like to encrypt the message $\boldsymbol{m} = (-102,\ -79,\ 91,\ 10,\ 29)$, so we randomly choose an error vector $\boldsymbol{e}$, say $(3,\ -3,\ 3,\ -3,\ -3)$ and compute:

$$\boldsymbol{c} := \mathbf{B} \cdot \boldsymbol{m} + \boldsymbol{e} = (-2552,\ -7677,\ 2981,\ -12896,\ -3503)$$

To decrypt $\boldsymbol{c}$ we first compute:

$$\boldsymbol{x} := \lfloor \mathbf{R}^{-1} \cdot \boldsymbol{c} \rceil = (-481,\ -863,\ 89,\ -1426,\ -379)$$

and finally:

$$\boldsymbol{m} := \mathbf{B}^{-1} \cdot \mathbf{R} \cdot \boldsymbol{x} = (-102,\ -79,\ 91,\ 10,\ 29)\,,$$

which gives our chosen message $\boldsymbol{m}$. In other words, we applied Babai's round-off algorithm to find the closest vector, which we then can multiply by $\mathbf{B}^{-1}$ to get the original message. If we try to decrypt just with the inverse of $\mathbf{B}$ we get:

$$\begin{aligned} \boldsymbol{m}' = \mathbf{B}^{-1} \cdot \boldsymbol{c} &= \left(-\frac{125625}{16282},\ -\frac{786815}{48846},\ \frac{99107}{8141},\ -\frac{547405}{24423},\ \frac{111637}{3489} \right) \\ &\approx (-8,\ -16,\ 12,\ -22,\ 32) \end{aligned}$$

which is far away from $\boldsymbol{m}$.

5.1.1 Why This Works

In GGH there is a 1:1-correspondence between the messages and the lattice points by the following function

$$f : \mathbb{Z}^n \to \Lambda(\mathbf{B}), \boldsymbol{m} \mapsto \mathbf{B}\boldsymbol{m}$$

with inverse function

$$f^{-1} : \Lambda(\mathbf{B}) \to \mathbb{Z}^n, \boldsymbol{v} \mapsto \mathbf{B}^{-1}\boldsymbol{v}$$

In the second variant, where the message is decrypted in the error vector, there is an analogous correspondence between the messages and the error vectors. The following refers to the first variant of GGH, but can be transform for the second variant.
The first step in the encryption process is to map the message $\boldsymbol{m}$ to its corresponding lattice point $f(\boldsymbol{m})$, say $\boldsymbol{v}_m$. Then, an error vector $\boldsymbol{e}$ is added, so that the point is not

longer in the lattice; denote the new point by $\boldsymbol{v}'_m$. This vector is the ciphertext. To find $\boldsymbol{v}_m$ from $\boldsymbol{v}'_m$ without knowing $\boldsymbol{e}$ is the CVP, which is easier for a short basis (the private key $\mathbf{R}$) than for a bad basis (the public key $\mathbf{B}$) using one of Babai's γ-CVP-solving algorithms ($\boldsymbol{v}_m = \mathbf{R} \cdot \lfloor \mathbf{R}^{-1} \cdot \boldsymbol{v}'_m \rceil$). The message $\boldsymbol{m}$ can then easily be received by applying the inverse function $f^{-1}(\boldsymbol{v}_m)$.
The crucial point in this system is finding $\boldsymbol{v}_m$ from $\boldsymbol{v}'_m$. All the other steps are easy to compute and not error-prone. We analyze this in the following section:

Error Analysis and Choice of σ

If σ is chosen too large, it can happen that $\boldsymbol{c} = \boldsymbol{v}_m + \boldsymbol{e}$ is outside the parallelepiped centered at $\boldsymbol{v}_m$ (see Figure 2.6) or even closer to a point $\boldsymbol{u} \in \Lambda(\mathbf{R})$ than to $\boldsymbol{v}_m$, and so Babai's nearest lattice point method returns $\boldsymbol{u}$, which corresponds to a different message than the desired $\boldsymbol{m}$. In this section we discuss how to choose σ such that the probability of wrong decryption is very low or even impossible.

Lemma 20. An inversion error occurs if and only if $\lfloor \mathbf{R}^{-1} \cdot \boldsymbol{v}'_m \rceil \neq \mathbf{0}$

Proof. Let $\mathbf{T} := \mathbf{B}^{-1}\mathbf{R}$, which is unimodular since $\Lambda(\mathbf{B}) = \Lambda(\mathbf{R})$ (Proposition 2), so there exists an unimodular matrix $\mathbf{U}$ such that $\mathbf{B} = \mathbf{U}\mathbf{R} \iff \mathbf{U}^{-1} = \mathbf{B}^{-1}\mathbf{R}$, which is also unimodular.
Inversion works like this: $\boldsymbol{v} = \mathbf{T} \cdot \lfloor \mathbf{R}^{-1} \cdot \boldsymbol{v} \rceil$ and $\boldsymbol{e} = \boldsymbol{c} - \mathbf{B}\boldsymbol{v}$, so if $\boldsymbol{v}$ is computed correctly, then so is $\boldsymbol{e}$. We now proof under which condition $\boldsymbol{v}$ is computed correctly:

$$\begin{aligned}
\mathbf{T} \cdot \lfloor \mathbf{R}^{-1} \cdot \boldsymbol{c} \rceil &= \mathbf{T} \cdot \lfloor \mathbf{R}^{-1} \cdot (\mathbf{B}\boldsymbol{v} + \boldsymbol{e}) \rceil \\
&= \mathbf{T} \cdot \lfloor \mathbf{R}^{-1} \cdot \mathbf{B}\boldsymbol{v} + \mathbf{R}^{-1} \cdot \boldsymbol{e}) \rceil \\
&= \mathbf{T} \cdot \lfloor (\mathbf{B}^{-1} \cdot \mathbf{R})^{-1}\boldsymbol{v} + \mathbf{R}^{-1} \cdot \boldsymbol{e}) \rceil \\
&= \mathbf{T} \cdot \lfloor \mathbf{T}^{-1}\boldsymbol{v} + \mathbf{R}^{-1} \cdot \boldsymbol{e}) \rceil
\end{aligned}$$

Since $\mathbf{T}^{-1}$ is unimodular, $\mathbf{T}^{-1}\boldsymbol{v}$ is an integral vector, so it can be taken outside of the rounding.

$$\begin{aligned}
\mathbf{T} \cdot \lfloor \mathbf{R}^{-1} \cdot \boldsymbol{c} \rceil &= \mathbf{T} \cdot \mathbf{T}^{-1}\boldsymbol{v} + \mathbf{T} \cdot \lfloor \mathbf{R}^{-1} \cdot \boldsymbol{e}) \rceil \\
&= \boldsymbol{v} + \mathbf{T} \cdot \lfloor \mathbf{R}^{-1} \cdot \boldsymbol{e} \rceil
\end{aligned}$$

So,

$$\begin{aligned}
\mathbf{T} \cdot \lfloor \mathbf{R}^{-1} \cdot \boldsymbol{c} \rceil = \boldsymbol{v} &\iff \mathbf{T} \cdot \lfloor \mathbf{R}^{-1} \cdot \boldsymbol{e} \rceil = \mathbf{0} \\
&\iff \lfloor \mathbf{R}^{-1} \cdot \boldsymbol{e} \rceil = \mathbf{0}
\end{aligned}$$

since $\mathbf{T}$ is unimodular. □

Theorem 21. Let $\mathbf{R}$ be the private basis used in the inversion and denote the maximum L_1-norm of the rows in $\mathbf{R}^{-1}$ by ρ. Then, as long as $\sigma < 1/(2\rho)$, no inversion errors can occur.

Proof. We want to show, that if $\sigma < 1/(2\rho)$, then $\lfloor \mathbf{R}^{-1} \cdot \boldsymbol{e} \rceil = \mathbf{0}$, which is equivalent to show $\forall j \in \{1, \ldots, n\} : \lfloor (\mathbf{R}^{-1})_j \cdot \boldsymbol{e} \rceil = 0$, where $(\mathbf{R}^{-1})_j$ denotes the j-th row of $\mathbf{R}^{-1}$.
Fix an arbitrary $j \in \{1, \ldots, n\}$ and denote the entries of $(\mathbf{R}^{-1})_j$ by $\hat{r}_i$. Consider $(\mathbf{R}^{-1})_j \cdot \boldsymbol{e}$:

$$\begin{aligned}
(\mathbf{R}^{-1})_j \cdot \boldsymbol{e} &= \sum_{i=1}^{n} \hat{r}_i \cdot \boldsymbol{e}_i \\
&\leq \sum_{i=1}^{n} |\hat{r}_i| \cdot |\boldsymbol{e}_i| \\
&= \sum_{i=1}^{n} |\hat{r}_i| \cdot \sigma \\
&= \sigma \cdot \sum_{i=1}^{n} |\hat{r}_i| \\
&= \sigma \cdot \left| (R^{-1})_j \right| \\
&\leq \sigma \cdot \rho \\
&< \frac{1}{2\rho} \cdot \rho \\
&= \frac{1}{2}
\end{aligned}$$

This means, that $\lfloor (\mathbf{R}^{-1})_j \cdot \boldsymbol{e} \rceil = 0$ and since j was chosen arbitrary, this is true for all $j \in \{1, \ldots, n\}$, which proofs the statement. □

Theorem 22. Let $\mathbf{R}$ be the private basis used in the inversion and denote the maximum L_∞-norm of the rows in $\mathbf{R}^{-1}$ by $\frac{\gamma}{\sqrt{n}}$. Then the probability of inversion errors is bounded by

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

Proof. The proof can be found in [GGH97b, Theorem 6]. □

Remark (Analogy to McEliece Cryptosystem)**.** In some sense GGH and McEliece cryptosystem [McE78] are very related. Both cryptosystems map the message to a corresponding vector and then add a certain amount of noise to it. In McEliece cryptosystem, the message is mapped to its corresponding code word using a Goppa-code capable to correct t errors and then a random error vector of weight t is added. In both systems the public and the private key are the same object but with different representations (in GGH a lattice, in McEliece a linear code). Also in both cryptosystems the private keys is capable to eliminate the noise added in the encryption process.

5.2 Attacks

5.2.1 Basis Reduction Attack

Recall, that $\boldsymbol{c} = \mathbf{B}\boldsymbol{m} + \boldsymbol{e}$. Computing $\mathbf{B}^{-1}\boldsymbol{c} = \boldsymbol{m} + \mathbf{B}^{-1}\boldsymbol{e}$ and rounding the outcome gives almost sure a wrong result, since the maximum L_1-norm of the rows in $\mathbf{B}^{-1}$ is extremely large in comparison to that one from $\mathbf{R}^{-1}$ and so Babai's algorithm gives a wrong cleartext with extremely high probability. A first approach can be applying LLL to the public key to give a reduced basis and then decrypt with this basis. The success of this idea is extremely dependent to the quality of the generation of the public key. Our experiments have shown, that it is very easy to generate a public key, resistant to this kind of attack, simply by multiplying more univariate matrices to the private key to obtain the public key. In Figure 5.2.1 the results are shown, where on the x-axis the number of unimodular matrices multiplications used in the public key generation and on the y-axis the rate of correct decrypted message entries are displayed. The experiments were taken in dimension 56 and the results are the average out of 10 iteration, each with another private key. For

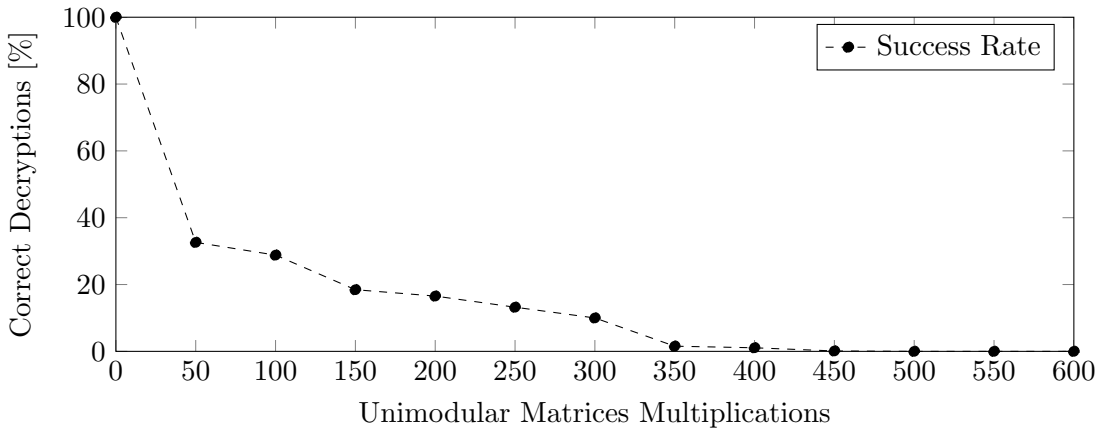

Figure 5.1: Success Rate of Basis Reduction Attack

the same private keys, with public key $\mathbf{B} = HNF(\mathbf{R})$, the attack could always decrypt the last entry of the message in our experiments. Knowing this, it can easily be fixed by setting the last entry to an arbitrary value which will be ignored during decryption.

Example 10. In this example we take up Example 9 and demonstrate the attack. The first step is to reduce the public key $\mathbf{B}$. We decided to choose the LLL algorithm with

$\delta = 0.99$, which leads to

$$\begin{pmatrix} 0 & 3 & -4 & -1 & 11 \\ 2 & 9 & 1 & 1 & -2 \\ 10 & -2 & -1 & 1 & 3 \\ -2 & -4 & 11 & 0 & 1 \\ 1 & 1 & 0 & -11 & -3 \end{pmatrix}$$

which is a very similar lattice matrix basis as the original private key. Continuing with this key to decrypt $\boldsymbol{c}$ gives $(-102, -79, 91, 10, 29)$, which is exactly our chosen message.

Practical Tests

Figure 5.2 shows the measured times and needed block size for attacking an instance of GGH system with $\sigma = 1$. For all measure dimensions, except of 175, the times of 10 independently attacks are shown. Since in dimension 175 a successful attack went 21 days, we ran it only twice. We first tried to reduce the basis only with LLL, which gave a sufficiently reduced basis until dimension 100, after that we continued with BKZ and increased the block size until the reduction was good enough to decrypt the message. The expected runtime for dimension 200 is about one whole year.

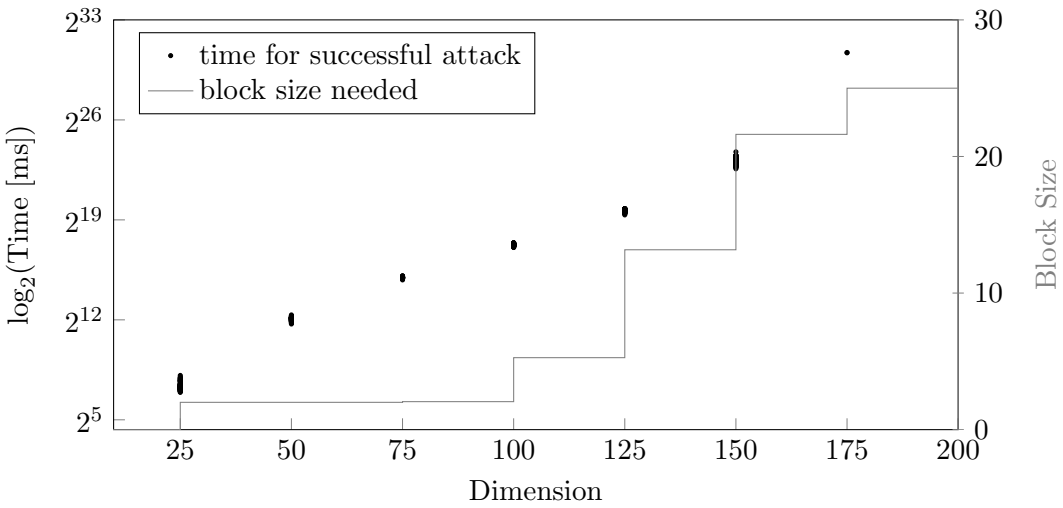

Figure 5.2: Time for a Successful Basis Reduction Attack

5.2.2 Embedding Attack

Recall, that the crucial point in GGH is to solve the CVP. One approach to overcome to solve this NP-hard problem exactly could be the use of the well performing LLL algorithm. As seen in Section 2.4.3, Kannan's embedding technique is a heuristic method to approximate a CVP of dimension n by approximating a $n+1$-dimensional SVP with LLL.

Let $\mathbf{B} = (\boldsymbol{b_1}, \ldots, \boldsymbol{b_n})$ be the public key which is a basis for a lattice Λ, and let $\boldsymbol{c}$ be a point in $\mathbb{Z}^n$ representing the ciphertext . We want to find the closest vector $\boldsymbol{t}$ to $\boldsymbol{c}$ in Λ. Consider the lattice Λ' generated by

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

In Section 2.4.3, we have seen that solving the SVP in the lattice Λ' leads with high probability to the closest vector $\boldsymbol{t}$ of $\boldsymbol{c}$ in Λ. The message $\boldsymbol{m}$ can then be computed solving the equation $\boldsymbol{t} = \mathbf{B}\boldsymbol{m}$.
Since the embedding technique is heuristic, so is the attack, nevertheless it performs very well in dimensions lower than 200 (see Figure 5.3).

Example 11. In this example we show Kannan's embedding technique on the GGH instance of Example 9. The embedded lattice basis is:

$$\mathbf{B}' = \begin{pmatrix} -2552 & 81 & 3 & 58 & 26 & 14 \\ -7677 & 293 & 9 & 222 & 101 & 59 \\ 2981 & 34 & -2 & 64 & 29 & 6 \\ -12896 & 52 & -4 & -88 & -39 & 17 \\ -3503 & 69 & 1 & 33 & 15 & 16 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

By applying the LLL basis reduction algorithm to it, we get

$$\begin{pmatrix} 3 & 2 & 3 & -1 & 2 & 3 \\ -3 & -2 & 0 & 3 & 7 & -3 \\ 3 & -2 & -4 & 5 & 0 & -3 \\ -3 & 3 & -5 & 4 & 4 & 3 \\ -3 & -6 & -3 & -2 & -2 & 5 \\ 1 & -3 & -4 & -4 & 1 & -3 \end{pmatrix}$$

where we can see the error vector $\boldsymbol{e} = (3, -3, 3, -3, -3)$ in the first column together with the sign in its last entry. If there was a -1 in the last entry, the vector would have been $-\boldsymbol{e}$. We can now compute

$$\boldsymbol{m} = \mathbf{B}^{-1}\boldsymbol{t} = \mathbf{B}^{-1}(\boldsymbol{c} - \boldsymbol{e}),$$

which gives us the correct decryption.

Remark. Theoretically, it is not necessary to run the LLL or BKZ algorithm to the end, if a column is found, that matches the error vector, the algorithm can be terminated precociously.
On the other hand one complete reduction with LLL is often not enough for finding the error vector in high dimensions (> 150). We adopted our algorithm in the following way. We first reduce $\mathbf{B}'$ with LLL ($\delta = 0.99$), if a suitable vector is found, then terminate, else try to reduce $\mathbf{B}'$ with BKZ with increasing block size and check after every reduction if the error vector has been found. If block size is equals the lattice rank, then it is guaranteed that the shortest vector is in the reduced lattice (but then, the algorithm is not longer sure to run in polynomial time).

Practical Tests

For measuring the success rate and the time for embedding attacks we run the following experiment. For every measured dimension up to 175 we generated 10 instances of the cryptosystem with $\sigma = 1$ and attacked it. Due to time consumptions we ran only one attack in dimensions 200, 225, 250, 275 and 300. For the attack, we first used LLL and then BKZ with increasing block size from 3 to 25% of the dimension if the solution was not found in the first place. The success rate is displayed in Figure 5.3. It can be clearly seen, that the attacks almost always succeeded until dimension 225. After this dimension we were unable to attack the instances in under a week, so we aborted the experiment after that time. Earlier experiments by Nguyen [NS98], where they only applied the LLL algorithm, showed that after the dimension of 200, an embedding attack is impractical. In our experiments we came to a similar result.

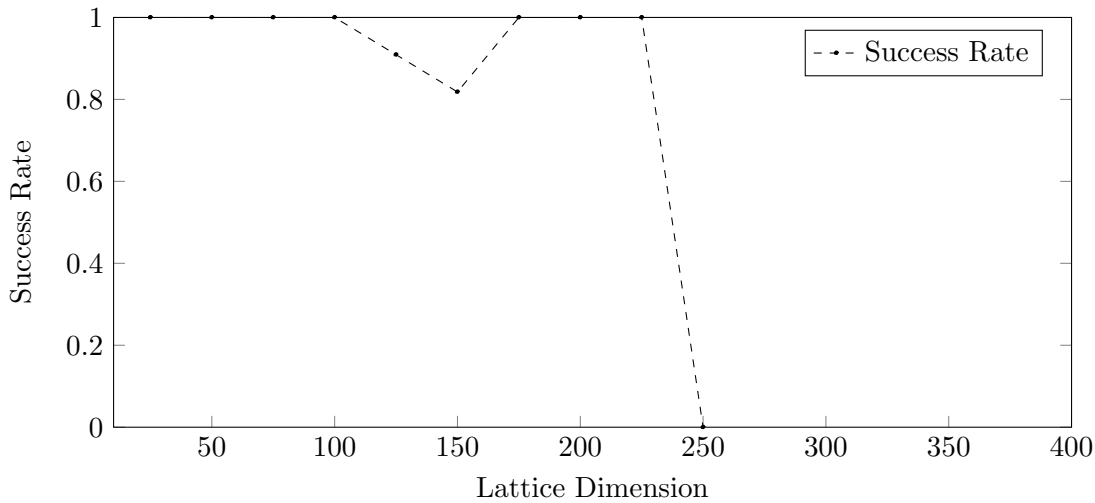

Figure 5.3: Success Rate of Embedding Attack

Figure 5.4 shows the time the attacks consumed. Note that in dimension 200 and 225, only one attack was performed, so the average block size is not as accurate as in the lower dimensions.

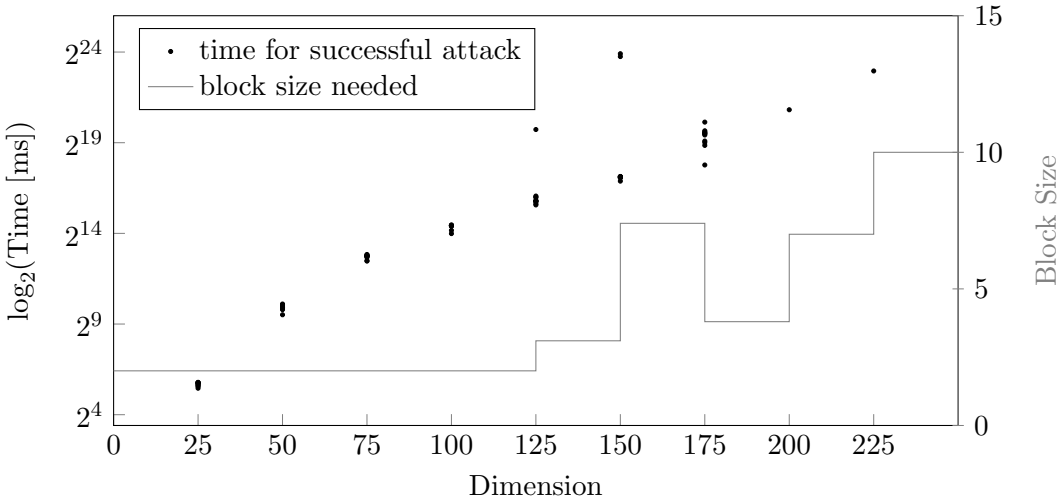

Figure 5.4: Time for a Successful Embedding Attack

5.2.3 Attack by Nguyen

Although the embedding attack presented in the last section is very effective, it turns out, that it works only for dimensions up to about 250. At Crypto99, Nguyen presented a method which is about equally efficient but works fine for dimensions up to 350 [Ngu99]. He was not capable to break GGH for dimension 400, but the system seems not longer practical in such high dimensions, since the public key is over 2MB.

The attack makes use of two security flaws in GGH. The first flaw is that the error vector is bounded by a value depending on the underlying lattice (generated by the randomly chosen basis $\mathbf{R}$). Increasing its norm more than a certain value means also increasing the possibility of decryption errors (see Theorem 21 and Theorem 22). So there is a balance between error-proneness in decryption and norm of the distortion vector $\boldsymbol{e}$, which doesn't allow one to scale the hardness for CVP as high as one likes while remaining practical (i.e. as error-free as possible in decryption). This trade-off makes the CVP instance of GGH easier than a general one.

The second flaw is the way the distortion vector is generated. It is chosen by randomly picking an element from $\{\pm\sigma\}^n$ to be as large as possible while staying error-free in the decryption process. The following equation holds for any vector $\boldsymbol{e}$ chosen in this way:

$$\boldsymbol{c} + \sigma\mathbf{1} \equiv \mathbf{B}\boldsymbol{m} \mod 2\sigma$$

where $\sigma\mathbf{1}$ is the n-dimensional vector consisting only of σ's. Note that $\boldsymbol{c}, \sigma$, and $\mathbf{B}$ are known, so this is a modular system in the unknown $\boldsymbol{m}$. We can simply compute

$$\mathbf{B}^{-1}(\boldsymbol{c} + \sigma\mathbf{1}) \equiv \boldsymbol{m} \mod 2\sigma$$

Obviously, this works only if $\mathbf{B}$ has an inverse in $\mathbb{Z}_{2\sigma}$, but if $2\sigma = p_1 \cdot \dots \cdot p_n$ for p_i pairwise coprime integers, we can compute the kernels modulo the p_i's and use Chinese Remainder Theorem to solve the equation and get $\boldsymbol{m}$ modulo 2σ, from now on denoted by $\boldsymbol{m}_{2\sigma}$. In the published challenges σ was always 3, so it is possible find solutions modulo 2 and 3 and then reconstruct the solution modulo $2\sigma = 6$. In general the entries of $\boldsymbol{m}$ are larger than 2σ, so it doesn't suffices to know $\boldsymbol{m}_{2\sigma}$, but the following equation holds:

$$\boldsymbol{c} - \mathbf{B}\boldsymbol{m}_{2\sigma} = \mathbf{B}(\boldsymbol{m} - \boldsymbol{m}_{2\sigma}) + \boldsymbol{e},$$

where the vector $(\boldsymbol{m} - \boldsymbol{m}_{2\sigma})$ is of the form $2\sigma\boldsymbol{m}'$ for $\boldsymbol{m}' \in \mathbb{Z}^n$. This leads to the following equivalent equations:

$$\frac{\boldsymbol{c} - \mathbf{B}\boldsymbol{m}_{2\sigma}}{2\sigma} = \mathbf{B}\boldsymbol{m}' + \frac{\boldsymbol{e}}{2\sigma} \tag{5.2}$$

This can be considered as an instance of CVP with solution $\boldsymbol{m}'$. So the originally CVP with error vector $\boldsymbol{e}$ of norm $\sigma\sqrt{n}$ is relaxed to a new instance of CVP with error vector of norm $\frac{1}{2}\sqrt{n}$, which is a significant difference. Note that for this instance of CVP, the factor ρ in Theorem 21 can be much larger and still get the the correct solution with Babai's algorithm. The factor ρ has direct relation to the quality of the reduced basis (the larger ρ, the less reduced the basis), i.e. the basis in the new instance of CVP has not to be as reduced as the basis for the original CVP-instance. Moreover, one can verify if the basis is reduced enough by checking if

$$\boldsymbol{c} - \mathbf{B}\tilde{\boldsymbol{m}} \in \{\pm\sigma\}^n,$$

where $\tilde{\boldsymbol{m}}$ denotes the result get by solving the relaxed instance of CVP.

Remark. In order to solve the CVP over $\mathbb{Z}$ instead over $\mathbb{Q}$, we can rewrite (5.2) to

$$\frac{\boldsymbol{c} - \mathbf{B}\boldsymbol{m}_{2\sigma}}{\sigma} = 2\mathbf{B}\boldsymbol{m}' + \frac{\boldsymbol{e}}{\sigma}. \tag{5.3}$$

That way, it is ensured that the left hand side (define it as $\boldsymbol{c}'$) is in $\mathbb{Z}^n$ and the error vector is then in $\{\pm 1\}^n$ with a norm of $\sqrt{n}$.

Example 12. We also want to demonstrate this attack against the instance chosen in Example 9.

Step 1: Compute $\boldsymbol{m}_{2\sigma}$

We first compute $\boldsymbol{c} + \sigma\mathbf{1} \mod 6$, which is (1, 0, 2, 1, 4). The public basis $\mathbf{B}$ modulo 6 is

$$\begin{pmatrix} 3 & 3 & 4 & 2 & 2 \\ 5 & 3 & 0 & 5 & 5 \\ 4 & 4 & 4 & 5 & 0 \\ 4 & 2 & 2 & 3 & 5 \\ 3 & 1 & 3 & 3 & 4 \end{pmatrix}$$

which has determinant 48 and hence is not invertible in $\mathbb{Z}_6$, neither in $\mathbb{Z}_2$ or $\mathbb{Z}_3$. But in $\mathbb{Z}_2$ and in $\mathbb{Z}_3$ we can find solution to

$$\boldsymbol{c} + \sigma\mathbf{1} \equiv \mathbf{B}\boldsymbol{m} \tag{5.4}$$

which are (1, 0, 1, 0, 1) and (0, 1, 1, 0, 0) for $\mathbb{Z}_2$ and $\mathbb{Z}_3$ respectively. The kernels of $\mathbf{B}$ in $\mathbb{Z}_2$ and $\mathbb{Z}_3$ are the sets $\{(1,1,0,0,0)\}$ and $\{(0,1,0,1,2)\}$, respectively. So for both rings we know one solution and the kernel, which yields to all possible solutions in these rings, since the difference of two solutions has to lie in the kernel. So we know that (1, 0, 1, 0, 1), (1, 0, 1, 0, 1) + $(1,1,0,0,0)$ are solutions to (5.4) in $\mathbb{Z}_2$ and (0, 1, 1, 0, 0), (0, 1, 1, 0, 0) + $(0,1,0,1,2)$ are solutions in $\mathbb{Z}_3$. With Chinese Remainder Theorem we can find all possible solutions in $\mathbb{Z}_6$ and the corresponding $\boldsymbol{c}' := \frac{\boldsymbol{c}-\mathbf{B}\boldsymbol{m}_{2\sigma}}{\sigma}$ listed in Table 5.1. c' has to

Table 5.1: Possible Solutions of (5.4) in $\mathbb{Z}_6$

solution modulo 6 ($\boldsymbol{m_6}$)	corresponding $\boldsymbol{c}'$
(0, 1, 5, 2, 1)	$\left(-\frac{1454}{3}, -1509, \frac{1301}{3}, -\frac{6194}{3}, -\frac{1856}{3}\right)$
(0, 2, 4, 4, 2)	$\left(-\frac{973}{2}, -1517, 434, -\frac{4137}{2}, -621\right)$
(0, 3, 3, 0, 3)	$\left(-\frac{1387}{3}, -1424, \frac{1390}{3}, -\frac{6334}{3}, -\frac{1825}{3}\right)$
(0, 4, 2, 2, 4)	$\left(-\frac{2785}{6}, -1432, \frac{1391}{3}, -\frac{12691}{6}, -\frac{1832}{3}\right)$
(0, 5, 1, 4, 5)	$(-466, -1440, 464, -2119, -613)$

be in $\mathbb{Z}^n$ (see (5.3)), so the last row contains the only possible value of $\boldsymbol{m}_6$. In case there was more possible solutions, we could continue with all of them iteratively.

Step 2: Solving the relaxed CVP

We now have the value for $\boldsymbol{m_6}$ and can continue to solve the CVP. In case there were more solutions, just do this step for all. Solving the CVP with $\boldsymbol{m_6} = (0, 5, 1, 4, 5)$ gives the error vector $\boldsymbol{e}' = (1, 0, 1, 0, 0)$ and so $\boldsymbol{e} = (\boldsymbol{e}' \cdot 2\sigma) - \sigma = (3, -3, 3, -3, -3)$ and thus $\boldsymbol{m} = \mathbf{B}^{-1}(\boldsymbol{c} - \boldsymbol{e}) = (-102, -79, 91, 10, 29)$, which is the originally chosen message.

Remark. This attack can easily be avoided by choosing the error-vector differently. However, choosing the error vector uniformly at random from the set $\{-\sigma, \ldots, \sigma\}$ facilitate the CVP the system is based on. So, both methods have their disadvantages. We propose the following trade-off: the error-vector should be chosen randomly but biased from the set $\{-\sigma, \ldots, \sigma\}$, rather than uniformly. For instance for $\sigma = 3$, $\mathbb{P}(\{\pm 3\}) = 0.75$, $\mathbb{P}(\{\pm 2\}) = 0.2$, $\mathbb{P}(\{\pm 1\}) = 0.04$, $\mathbb{P}(\{0\}) = 0.01$.

5.2.4 Attack by Moon Sung Lee and Sang Geun Hahn

In 2000, Moon Sung Lee and Sang Geun Hahn found out, that partial information on the message can be used to simplify the CVP underlying the decryption [LH08]. With this method and the help of Nguyen's attack described in the last section the authors were able to solve the public GGH challenge in dimension 400. In this part, we shortly introduce the idea behind the method and how the authors propose to avoid it to some extent.

Assume that k of the n entries of the message $\boldsymbol{m}$ are known and without loss of generality these are the first k entries of $\boldsymbol{m}$. We then denote the first k entries of $\boldsymbol{m}$ by $\boldsymbol{m^1}$ and the remaining ones by $\boldsymbol{m^2}$, so $\boldsymbol{m} = (\boldsymbol{m^1 m^2})$ where $\boldsymbol{m^1}$ are the known entries and $\boldsymbol{m^2}$ are the unknown ones. In the same manner we split the public key $\mathbf{B}$ into $\mathbf{B^1}$ and $\mathbf{B^2}$. We can now rewrite (5.1) in the encryption process to

$$\boldsymbol{c} = \mathbf{B} \cdot \boldsymbol{m} + \boldsymbol{e} = \begin{pmatrix} \mathbf{B^1} \\ \mathbf{B^2} \end{pmatrix} (\boldsymbol{m^1 m^2}) + \boldsymbol{e} = \mathbf{B^1}\boldsymbol{m^1} + \mathbf{B^2}\boldsymbol{m^2} + \boldsymbol{e},$$

where $\boldsymbol{c}$ and $\mathbf{B^1}\boldsymbol{m^1}$ are known and therefore

$$\boldsymbol{c} - \mathbf{B^1}\boldsymbol{m^1} = \mathbf{B^2}\boldsymbol{m^2} + \boldsymbol{e}.$$

This simplifies the original CVP of dimension n to dimension $n - k$. Furthermore, the authors also state that the lattice gap $(\alpha(\Lambda(\mathbf{B^2})))$ is larger and thus lattice reduction algorithms perform better.

This attack can be useful in the two following scenarios: If entries of the message are predictable for some reason, e.g. because of the way the underlying data is structured, these entries can be cut out and one can attack only the unknown entries. The other possibility is related to the attack of Nguyen discussed in the last section. If $\boldsymbol{m}_{2\sigma}$ is found, the possible values of $\boldsymbol{m}_1$ are divided by the factor 2σ, since we know its modulus. This is also the reason why Lee and Hahn were able to to solve the public GGH challenge in dimension 400, for which Nguyen found only the message modulo 6, but not the message itself.

This attack can be avoided by permuting the message vector, so it is not longer clear, that $\mathbf{B^I}$ is multiplied by $\boldsymbol{m_{\mathrm{i}}}$. Furthermore, if we can avoid Nguyen's attack, this attack is most probably no longer practical.

Example 13. As before, we show this attack on the initial example (Example 9). Let us say, that we know the first entry (-102), for whatever reason. Then we define

$$\mathbf{B^1} := \begin{pmatrix} 81 \\ 293 \\ 34 \\ 52 \\ 69 \end{pmatrix} \text{ and } \mathbf{B^2} := \begin{pmatrix} 3 & 58 & 26 & 14 \\ 9 & 222 & 101 & 59 \\ -2 & 64 & 29 & 6 \\ -4 & -88 & -39 & 17 \\ 1 & 33 & 15 & 16 \end{pmatrix}$$

Now, we can compute $\mathbf{B^1}\boldsymbol{m^1}$ which is $(-8262, -29886, -3468, -5304, -7038)$ and thus $\boldsymbol{c}-\mathbf{B^1}\boldsymbol{m^1}$: $(5710, 22209, 6449, -7592, 3535)$ and use this as ciphertext for the 4-dimensional CVP: $\boldsymbol{c} - \mathbf{B^1}\boldsymbol{m^1} = \mathbf{B^2}\boldsymbol{m^2} + \boldsymbol{e}$, which we can solve for instance with Nguyen's method.

5.3 Practicality

5.3.1 Space and Time Analysis

Since the GGH cryptosystem is more a template, rather than a fully determined cryptosystem it is not simple to give a exact space and time analysis. In this section we give the complexities for the parameters the authors recommend. The **private key** is a good basis for a lattice. The suggestions of the authors is to construct the base matrix as $\mathbf{R} = k \cdot \mathbf{I_n} + \mathbf{Q}$, where $k = \lfloor \sqrt{n} \cdot l \rfloor$ and $\mathbf{Q}$ is a random perturbation matrix with entries from $\{-l, \ldots, l\}$ for $l = 4$, so this matrix can be stored with $\mathcal{O}\left(n^2 \log n\right)$ space. The **public key** is a "bad" basis matrix for the same lattice, obtained by multiplying $\mathbf{R}$ by unimodular matrices. The entries of such a matrix are typically an order of magnitude higher than n, so it needs $\mathcal{O}\left(n^3 \log n\right)$ space. The **ciphertext** is essentially the product of the message vector by the public key, so it is of order $\mathcal{O}\left(n^2 \log n\right)$.
The minimal effort to create the **private key** consists of generating n^2 random integers. We assume here, that this can be done in constant time, e.g. by taking them from a pregenerated pool of random numbers. So the costs of producing the private key is $\mathcal{O}\left(n^2\right)$. However, the authors suggested to reduce this basis and invert it for speed up the decryption phase, so in this case it needs $\mathcal{O}\left(n^3\right)$, since inversion is cubic in time. The computation of the **public key** from the private one, hardly depends on the chosen method. We assume here, that it includes at least a matrix-matrix multiplication of $\mathbf{R}$ by an unimodular matrix, which requires $\mathcal{O}\left(n^3\right)$ time complexity, so we consider this as a lower bound. The **encryption** is a matrix-vector multiplication, so it needs $\mathcal{O}\left(n^2\right)$, while the **decryption** is the application of Babai's round-off algorithm on this product, which is essentially a matrix-vector multiplication, which can be done in $\mathcal{O}\left(n^2\right)$. We assume here, that the inverse has already been precomputed, otherwise the first decryption would

be $\mathcal{O}\left(n^3\right)$. Another implementation where the inverse of the matrix don't have to be computed runs in also in $\mathcal{O}\left(n^2\right)$.

Space	
Object	Size
Private Key	$\mathcal{O}\left(n^2 \log n\right)$
Public Key	$\mathcal{O}\left(n^3 \log n\right)$
Ciphertext	$\mathcal{O}\left(n^2 \log n\right)$

Time	
Operation	Time
Private Key Generation	$\mathcal{O}\left(n^2\right)$
Public Key Generation	$\geq \mathcal{O}\left(n^3\right)$
Encryption	$\mathcal{O}\left(n^2\right)$
Decryption	$\mathcal{O}\left(n^2\right)$

Figure 5.5: Space and Time Complexities of GGH

5.3.2 Practical Tests

The computation of the error-bound σ includes either the inversion of the private key or the computation of the Gram-Schmidt matrix in round-off or nearest-plane, respectively. Both computations are extremely time-consuming computations and have to be done in the key generation phase, since σ has to be delivered with the public key. We wanted to check how long the computations take in practice and if it possible to just skip this step, and assume that σ has a certain value, and what this value might can be.
We also want to examine if the LLL-reduction of the private key is necessary. Therefore we encrypted messages with different error-bounds, and compare the decryption with and without reduced private key.

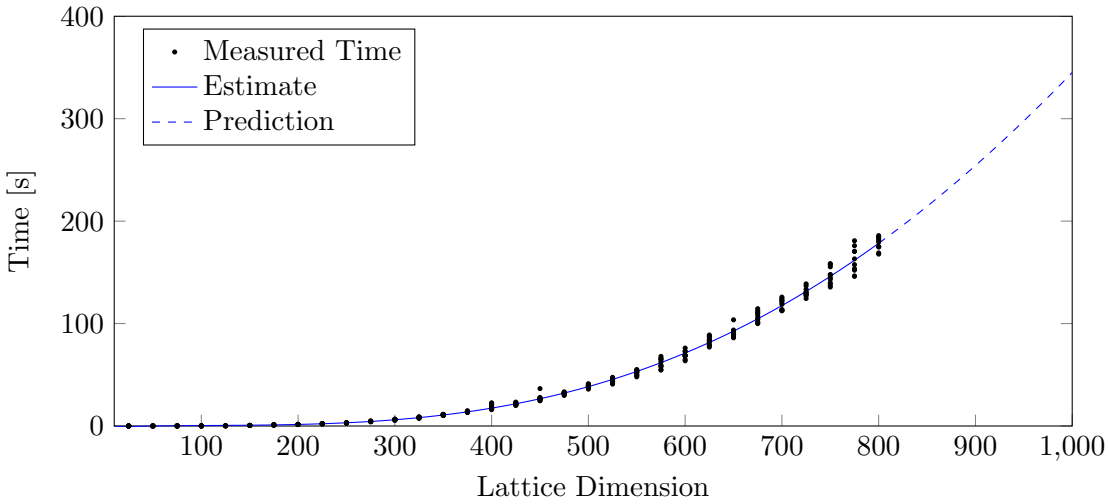

Figure 5.6: Time for Reducing a Private Key $\mathbf{R}$

Reducing the basis $\mathbf{R}$ leads to a more accurate decryption, since Babai's round-off

algorithm provides better results for the reduced basis. However, this reduction comes with a certain cost. We tried to figure out how much this cost is in practice. Therefor, we reduced the private basis generated as suggested by the authors with LLL. The times are given in Figure 5.6. For every dimension $d \in D := \{25 \cdot i \mid 1 \leq i \leq 32\}$ we measured the time for reducing 10 different d-dimensional bases $\mathbf{R}$.
As can be seen in the practical experiment, for dimension 800 the reduction needs about 3 minutes. This can only be bearable if the impact on the private key was well noticeable, what we want to determine in the following.

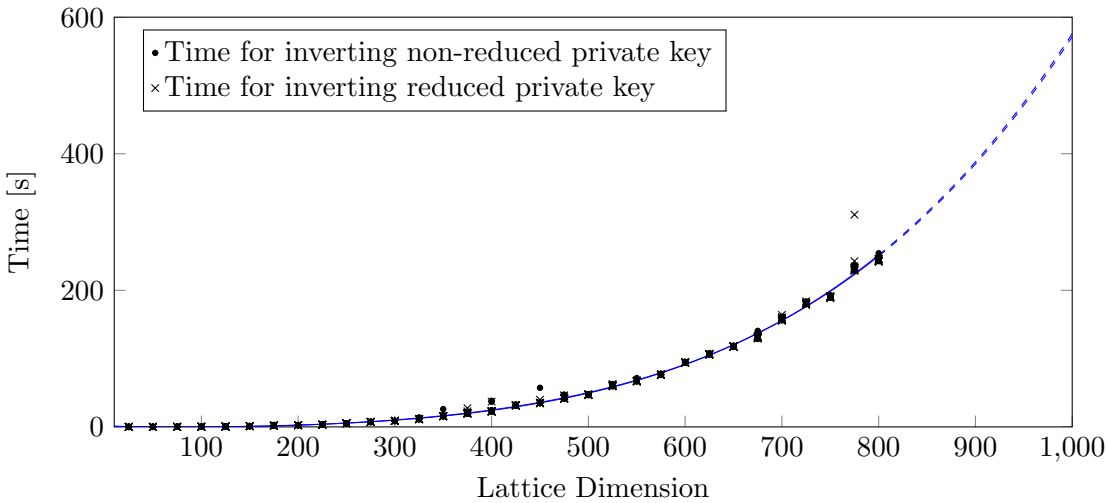

Figure 5.7: Time for Inverting a Private Key $\mathbf{R}$

One step in the key generation phase is to invert the private basis. The inverted matrix is used in two tasks, namely the computation of σ (for the error bound) and to decrypt the message using Babai's round-off algorithm. The latter can also be done without $\mathbf{R}^{-1}$ (see Section 2.4.2). Figure 5.7 shows the time for inverting the private matrix. For every dimension in D we generated 10 bases and inverted them. To see if the reduced bases can be inverted faster, we also inverted the the exact same bases after a reduction with LLL. As one can observe in Figure 5.7, the reduction has not much impact on the timing of the inversion.
So, the actual reason for applying a reduction algorithm to the lattice basis $\mathbf{R}$ would be to provide a higher value of σ, and therefore make it harder for an attacker to solve the CVP. To see how much impact the application of the LLL algorithm has, we computed σ before and after a LLL-reduction of $\mathbf{R}$. Surprisingly, almost no improvement of σ could be determined in our experiment showed in Figure 5.8. Only in the uninteresting dimension 25 and 50 we have seen a difference. This is probably due to the way the private key is generated in the first place (short and near to the axis).

According to our experiments, there is no reason to apply the LLL algorithm at the key generation phase. It unnecessarily defers the key generation, since the actual value σ, that ca be seen as part of the public key, stays almost sure the same.

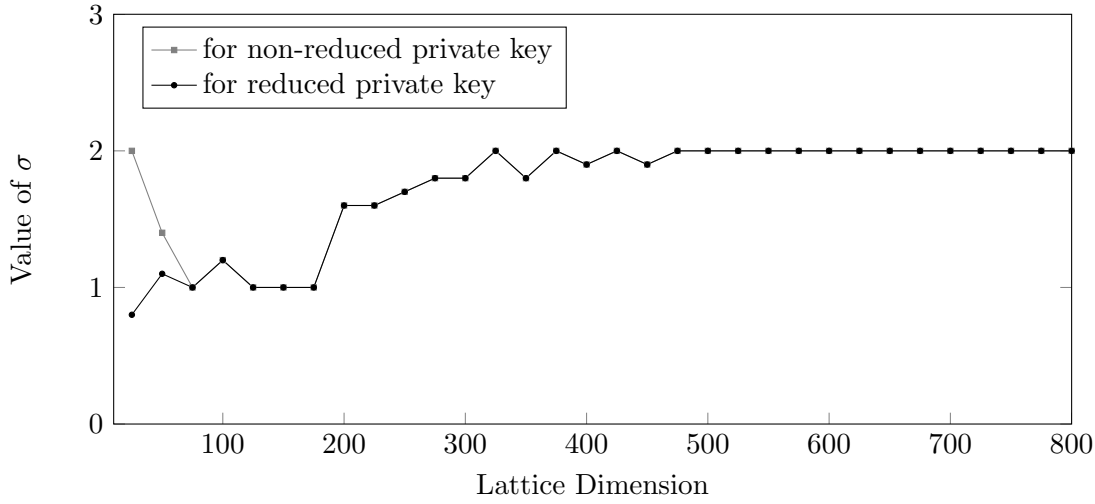

Figure 5.8: Values of σ

In Figure 5.9 the generation of the private key is shown. The experiment is without inverting nor reducing the private key, since we measured this steps separately (see Figures 5.6 and 5.7). All values are far below one second.

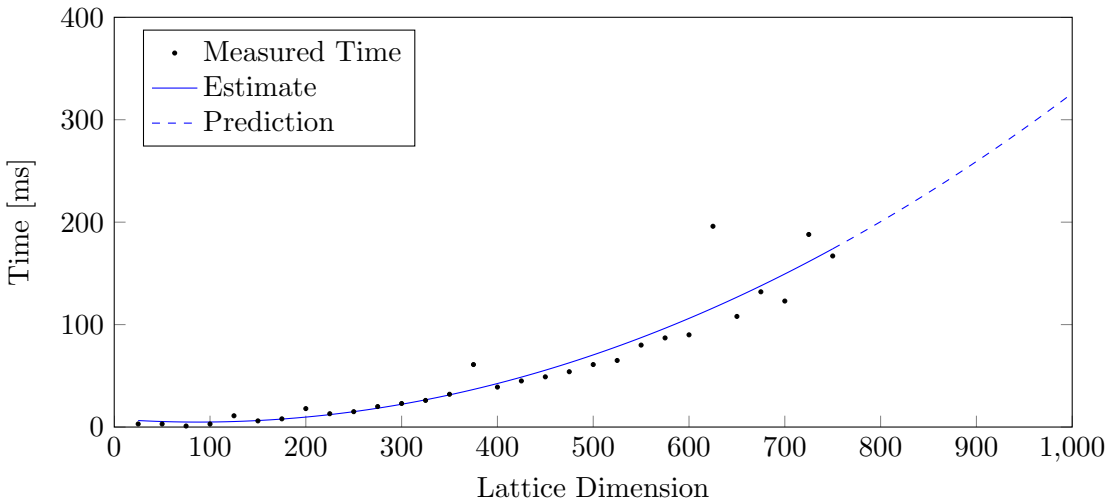

Figure 5.9: Time for Generating a Private Key in MCS

Figure 5.10 shows the time needed to generate the public key in seconds. As mentioned in Section 5.1, there are many ways to generate the public key. The authors suggested $2n$

mixing steps of the type "add multiples of other rows/columns to one column/row", what we also did.

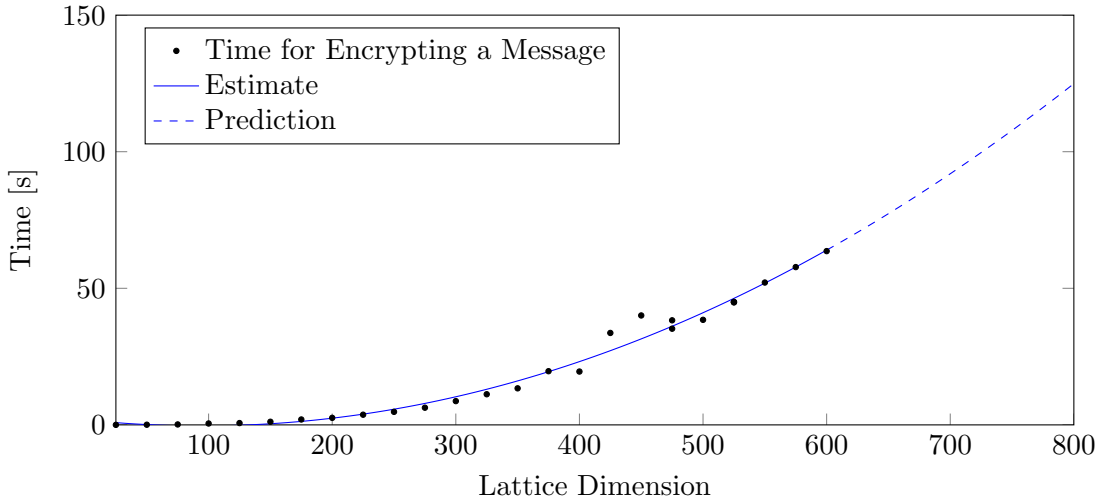

Figure 5.10: Time for Generating a Public Key in GGH

Figure 5.11 shows the time for encrypting a message in the GGH cryptosystem. As can be seen, the higher the dimension, the wider the distribution of the measure points, but all far below 1 second.

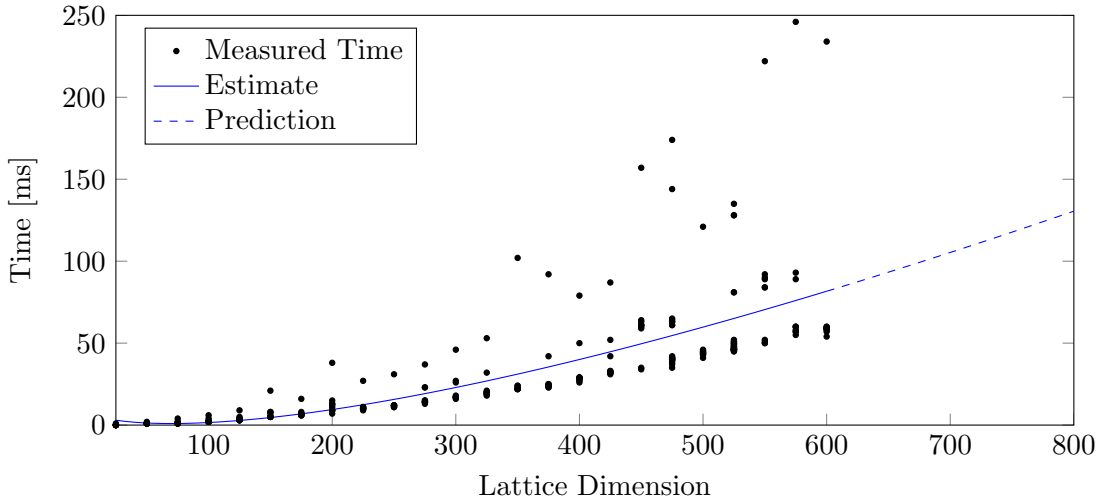

Figure 5.11: Time for Encrypting a Message in GGH

Figure 5.12 shows the decryption of the previous generated ciphertexts with Babai's round-off algorithm, as proposed by the authors. As can be seen in the figure, the decryption is about a factor of 20 slower than the encryption. For decrypting a message in

dimension 600, we needed about 2 seconds, which corresponds to a decryption speed of 10KB/s considering the message set $\{-128, -126, \ldots, 127\}^n$. This is not a bad speed for an asymmetric cryptosystem.

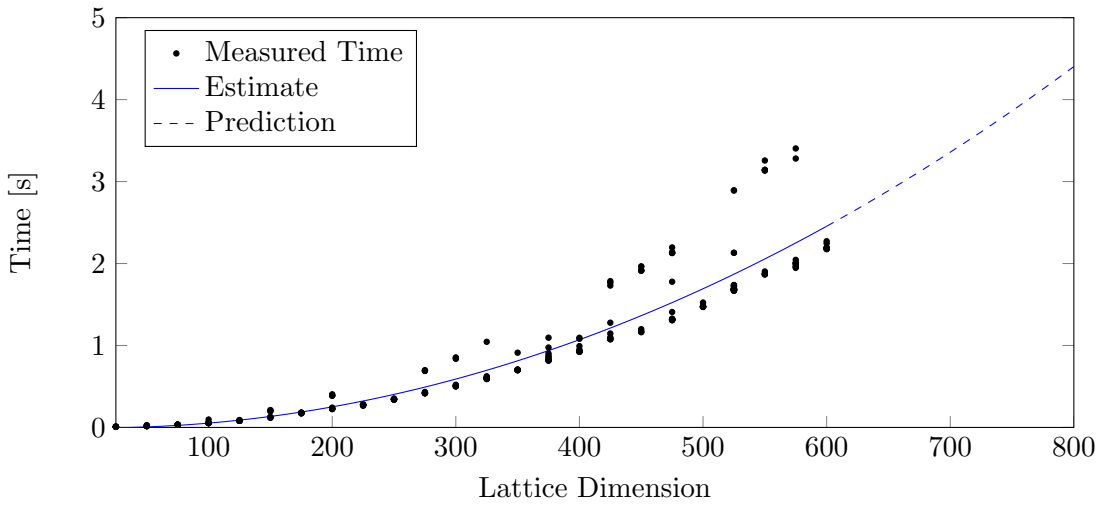

Figure 5.12: Time for Decrypting a Message with Round-Off

We also wanted to see, what happens if we decrypt ciphertexts which are encrypted with too high error vectors and what impact a reduction of the basis on the result has. For this we have chosen $\sigma \in \{1, 2, 4, 8, 12\}$, and encrypted the messages with this error bounds. For every dimension we generated 10 key-pairs and encrypted 100 messages per key-pair. Figure 5.13 shows the results. Every point is the average success ratio of the decryption of the 100 messages. As can be seen, the success-ratio is highly dimension-dependent. Further it can be seen, that even error bounds of 12 can be decrypted in higher dimensions, even if the original σ value was only 1. We presume that in dimension 800, the decryption error is under 1% for messages decrypted with a σ value of 12.
We also performed this experiment with LLL-reduced bases. The result was not what we expected, since the results with and without reduced basis are almost exactly the same. Babai's round-off algorithm should perform better when the basis is reduced. One explanation for the lack of improvement could be, that the way the private basis is chosen generates almost reduced bases. Indeed, the basis vectors for the private key are chosen always near an axis, and so their orthogonality defect is near to 1 and therefore they are quite good reduced.

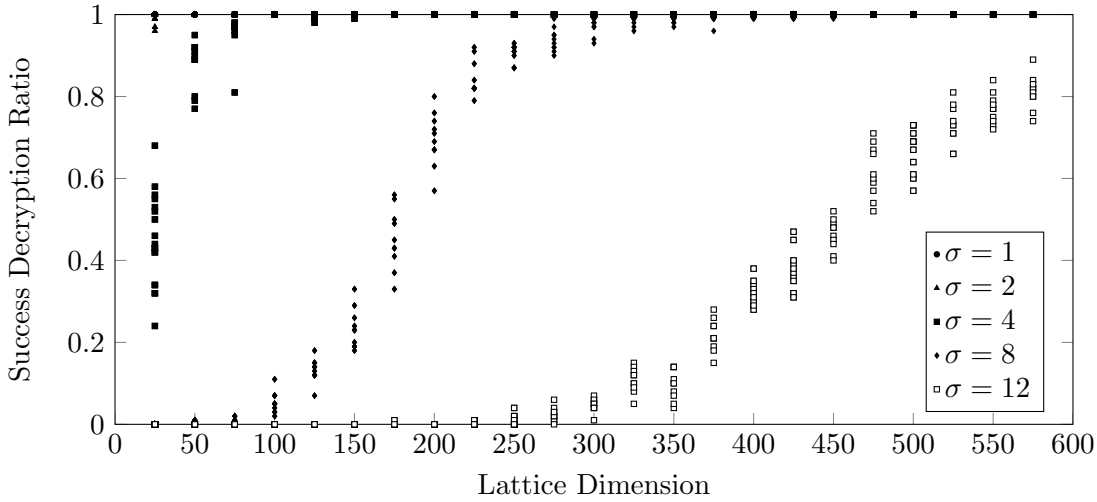

Figure 5.13: Success of Decrypting with Different σ's

These results could have a huge impact on the practicality of the cryptosystem. With this knowledge we can increase the error bounds when we simultaneously add a code capable to detect errors to about 1% of the entries. It is then no longer necessary to choose the error vector from the set $\{\pm\sigma\}^n$, but rather form the set $\{-\sigma, \ldots, \sigma\}$, which makes the system no longer vulnerable to Nguyen's attack.

Key and Ciphertext Size

Figures 5.14 and 5.15 show the sizes of the private and public key, respectively. It can be seen, that the public key is lager by the factor of about 3.

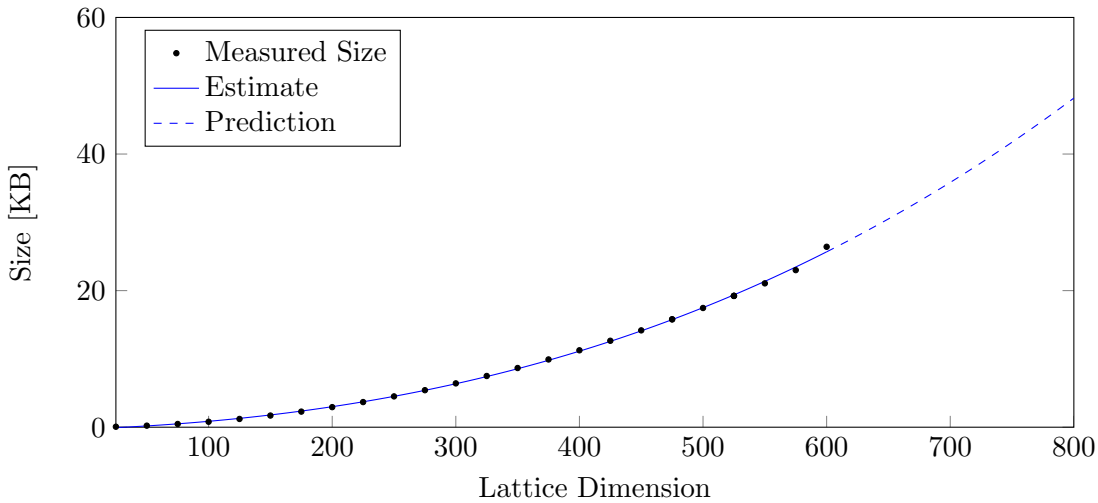

Figure 5.14: Size of a Private Key in GGH

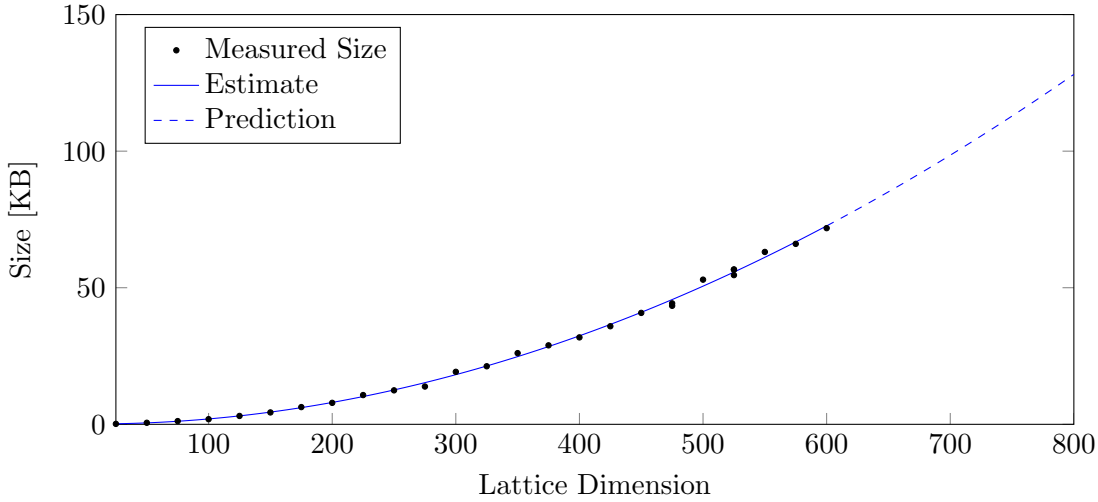

Figure 5.15: Size of a Public Key in GGH

5.4 Conclusion

In this section we discussed a type of cryptosystem, called GGH. It was originally proposed based on lattices. The cryptosystem, as the authors suggested it, has a weakness in the way the error vector is chosen, which can be exploit to attack the system at least until dimension 400. However, the error vector can easily be generated invulnerable to this kind of attack by allowing values between $-\sigma$ and σ, which leads to a weaker CVP in the error-free case. We showed that this issue can be overcome by allowing higher values of σ, which should increase the difficulty of the CVP drastically. The system would not longer be error-free, but we have seen that in higher dimensions (about 400), the error-rate is so small, even for $\sigma = 8$, that we easily can vanish the errors with a suitable error correcting code or at least detect them.

Another drawback of the system in the error-free case is the very time-consuming computation of σ. This value is needed to be sure that the private key is able to correct error vectors with maximum-norm of σ. As shown in Figure 5.7 this step makes the cryptosystem impractical in dimensions where it is secure, so this step has to be avoided. For this we attempted to optimize the cryptosystem in two ways: first we studied the impact on the value of σ when reducing the private key with LLL. Unfortunately, the reduced private key has in almost all tested cases the same value of σ as the non-reduced, but this is also a tribute to the way, the authors proposed to generate the private key, since the generated bases are already reduced to a certain extent, i.e. the orthogonality defect is very low. The second approach was to skip the step of computing σ and encrypt with values of σ up to 12, which seems very promising. In dimensions over 450 even messages, which were

encrypted with errors of L_∞-norm of 8, could be decrypted successfully without errors and the probability of successful decryption of messages encrypted with $\sigma = 12$ is about 90% in dimension 575. We recommend to use higher values of σ and encode the messages before encrypting to have the possibility of checking for correct decryption.
Comparing GGH to the Ajtai-Dwork cryptosystem, many improvements can be seen in the complexity analysis as well as in the practical tests. Public key space is 2 order of magnitude smaller, and so are the time complexities for key generation and encryption. The only point on which GGH asymptotically less optimized is the generation of the private key ($\mathcal{O}\left(n^3\right)$ instead of $\mathcal{O}(n)$), which in practice was never an issue when skipping the computation of σ. Unfortunately, there is no security proof for GGH analogous to Ajtai's worst-case/average-case equivalence.

Chapter 6

Micciancio's Cryptosystem

Two years after the publishing of GGH, Nguyen was able to break four of the five GGH breaking challenges (dimensions 200, 250, 300, 350). The only unbroken challenge had dimension 400 and a public key size of over 2 MB. Nguyen concluded, that unless major improvements were found, lattice based cryptography cannot provide a serious alternative to existing public-key encryption algorithms such as RSA [Ngu99]. In 2001 Daniele Micciancio presented such an improvement of the GGH cryptosystem. He was able to reduce the public key size and ciphertext size by a factor of n, without decreasing the security [Mic01b]. Indeed, it may be even more secure. Micciancio has proven, that attacks against the improved version can be transformed into an attack against the original system, which is at least equally effective. In the following we discuss his improvements, which is often referred as Micciancios Cryptosystem (MCS).

6.1 Description of the System

In [Mic01b] Micciancio introduced a generalized scheme of which GGH is a special case, what we also want to do. We first fix a probability distribution on the set of private bases $\mathbf{R}$. Let $0 < \rho \in \mathbb{R}$ be a correction radius, such that using $\mathbf{R}$ one is able to correct any error of length less than ρ. Consider the family of functions $f_{\beta,\gamma}$ which is determined by the two algorithms β and γ as follows:

β: a (possibly randomized) function with input a matrix $\mathbf{R}$ and output another basis $\mathbf{B}$ for the same lattice. $\mathbf{B}$ will be used as the public key.

γ: a (possibly randomized) function with input the public basis $\mathbf{B}$ and an error vector $\boldsymbol{r}$ and output the coefficients $\boldsymbol{x}$ of a lattice point $\mathbf{B}\boldsymbol{x}$ to be added to the error vector.

$f_{\beta,\gamma}$: the (possibly randomized) function with domain the set of vectors shorter than ρ

defined as follows:

$$f_{\beta,\gamma}(\boldsymbol{r}) = \mathbf{B}\boldsymbol{x} + \boldsymbol{r}$$

where $\mathbf{B} = \beta(\mathbf{R})$ and $\boldsymbol{x} = \gamma(\mathbf{B}, \boldsymbol{r})$

The probability distribution on $\mathbf{R}$ defines now a family of trapdoor functions $f_{\beta,\gamma}$ with public key $(\mathbf{B}, \rho)$ and trapdoor information $\mathbf{R}$ for any fixed β and γ. With this generalized definition of a family of functions, we can define different instances. As mentioned before, the GGH trapdoor function can be regarded as a special case of this, where $\beta(\mathbf{R})$ is a function applying a sequence of elementary random operations to $\mathbf{R}$ and $\gamma(\mathbf{B}, \boldsymbol{r})$ returns an integer vector $\boldsymbol{x}$ chosen at random from a sufficiently large region.
The proposal of Micciancio is the following:

$$\beta : \mathbf{R} \mapsto \mathbf{HNF}(\mathbf{R})$$
$$\gamma : (\mathbf{B}, \boldsymbol{r}) \mapsto \boldsymbol{r} \mod \mathbf{B}$$

This definitions have several benefits, which we want to discuss here. **HNF** denotes a function which outputs the Hermite normal form of an input matrix, as an example think of Algorithm 1 or the algorithm presented by Micciancio and Warinschi in [MW01], which has linear space complexity. The choice of the HNF of $\mathbf{R}$ for the public key has the clear advantage to be smaller than a matrix obtained by random elementary matrix operations, since it is an upper triangular matrix and even in the upper triangular part of the matrix is very sparse. Moreover, the HNF depends only on the lattice and not on a specific basis (see Lemma 6). So it is not possible to reveal less information about the private basis except the lattice itself, then the HNF does. More formally, any information about $\mathbf{R}$ revealed by $\mathbf{B} = \mathbf{HNF}(\mathbf{R})$ can also be revealed from any other basis $\mathbf{B}'$, since $\mathbf{B} = \mathbf{HNF}(\mathbf{B}')$ and $\mathbf{B}$ can be efficiently computed from $\mathbf{B}'$.
The aim of γ is to give a vector in Λ that ideally is uniformly chosen. In [Mic01b] Micciancio clarifies that "unfortunately this is neither a computationally feasible nor a mathematically well defined operation". He shows that reducing the vector $\boldsymbol{r}$ modulo the public basis $\mathbf{B}$ is a very efficient way for encrypting, especially since $\mathbf{B}$ is in HNF, which makes the reduction even easier Given $\boldsymbol{r}$ and $\mathbf{B}$, the integer vector $\boldsymbol{x}$ can be computed with the following formula componentwise, starting by the last component:

$$x_i := \left\lfloor \frac{r_i - \sum_{j>i} b_{i,j} x_j}{b_{i,i}} \right\rfloor \tag{6.1}$$

γ then outputs $\boldsymbol{c} := \mathbf{B}(-\boldsymbol{x}) + \boldsymbol{r} = \boldsymbol{r} \mod \mathbf{B}$

Theorem 23. Let $\mathbf{R}$ be the private basis and denote the minimum L_2-norm of the orthogonalized rows by ρ. Then, as long as $\sigma < 1/2\rho$, no inversion errors can occur in Babai's nearest plane algorithm.

Example 14. In this example we show the key generation, the encryption and the decryption of a Micciancio Cryptosystem in dimension 5. First we generate a private key as described in the GGH cryptosystem. For for demonstration purposes we took the same private key as for the GGH examples:

$$\mathbf{R} := \begin{pmatrix} 11 & 3 & 0 & -4 & 1 \\ -2 & 9 & 2 & 1 & -1 \\ 3 & -2 & 10 & -1 & -1 \\ 1 & -4 & -2 & 11 & 0 \\ -3 & 1 & 1 & 0 & 11 \end{pmatrix}$$

The public key $\mathbf{B}$ is simply the unique Hermite normal form of $\mathbf{R}$:

$$\mathbf{B} = \begin{pmatrix} 1 & 0 & 0 & 1 & 56567 \\ 0 & 1 & 0 & 1 & 26966 \\ 0 & 0 & 1 & 0 & 15774 \\ 0 & 0 & 0 & 2 & 53703 \\ 0 & 0 & 0 & 0 & 73269 \end{pmatrix}$$

Now we would like to encrypt a message. Since ρ is equal to 4.77 ($\rho := 1/2 \min \|r_i^\star\|$, where $r_i^\star$ denotes the i-th orthogonalized row of $\mathbf{R}$) we know that we are able to find the closest lattice point of a vector within the ball of radius 4 on any point $\boldsymbol{v} \in \Lambda$ using Babai's nearest plane algorithm. So we randomly choose a message

$$\boldsymbol{m} = (0,\ -3,\ -2,\ -2,\ -1)$$

The ciphertext of $\boldsymbol{m}$ is the vector $\boldsymbol{c} := \boldsymbol{m} \mod \mathbf{B}$, which is

$$\boldsymbol{c} = (0,\ 0,\ 0,\ 1,\ 39176)$$

One can see, that for all $i \in \{1, \ldots, 5\}$: $0 \leq c_i < b_{i,i}$. For decryption we use the private key $\mathbf{R}$ to find the closest vector $\boldsymbol{v}$ to $\boldsymbol{c}$ using Babai's nearest plane algorithm and compute the message:

$$\boldsymbol{m} = \boldsymbol{c} - \boldsymbol{v} = (0,\ 0,\ 0,\ 1,\ 39176) - (0,\ 3,\ 2,\ 3,\ 39177) = (0,\ -3,\ -2,\ -2,\ -1)$$

6.2 Cryptanalysis

MCS encrypts the message in the error vector. This vector is not chosen from the set $\{\pm\sigma\}$ as in GGH, but rather from the set $\{-\sigma, \ldots, \sigma\}$. This makes the system no longer vulnerable for the attack of Nguyen against GGH (presented in Section 5.2.3), where partial information could be found because of the structure of the error vector. But GGH

has another weakness, namely the short norm of the error vector, which is also rooted in MCS. The norm of the error vector in MCS is even smaller than the one in GGH ($\mathbb{E}[\|\boldsymbol{m}\|] = \sqrt{n(\sigma^2+\sigma)/3}$ instead of $\mathbb{E}[\|\boldsymbol{m}\|] = \sqrt{n}\sigma$). For this reason, a natural choice to attack MCS is the basis reduction attack presented in 5.2.1 and the embedding attack presented in Section 5.2.2. Basis reduction attack is nearly effectless in dimensions higher than 200 as Figure 5.2 shows. Therefore, in the following we focus on the embedding attack.

6.2.1 Embedding Attack

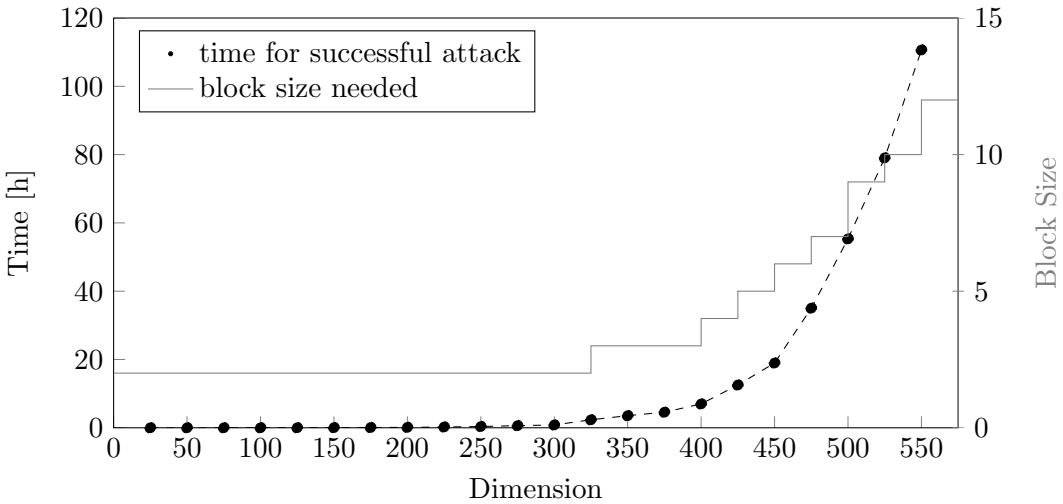

Figure 6.1: Time for a Successful Embedding Attack

To find out what could be a lower bound for the dimension of the MCS to be invulnerable against embedding attacks, we implemented the attack and measured the time for successful termination. The messages are taken from the set $\{-1, 0, 1\}$. Figures 6.1 and 6.2 show the exact same experiment in different scales. Figure 6.1 shows the attack in a linear scale together with the block size in which the BKZ algorithm succeeded. Until dimension 300, the LLL algorithm sufficed to find the message, in higher dimensions, the BKZ algorithm had to be used after reducing with LLL. This can be seen also in the growth of the time from dimension 300 to 325 in Figure 6.2.

Until dimension 350, every measured point is the mean value of five independent attacks in the corresponding dimension. In higher dimensions, we only executed one attack due to time consumption.

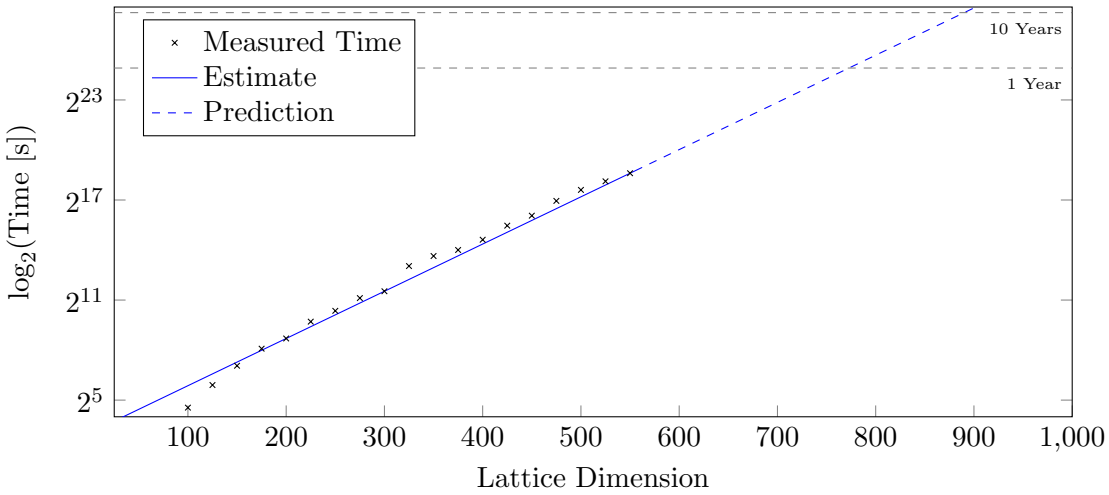

Figure 6.2: Time for a Successful Embedding Attack

According to our experiments it should suffice to use dimension 900. Note that the messages have very small norm, i.e. the error vector is very small in this experiment. Recall, that with this attack we only gain a single message of length n, rather than the private key, which one can use to decrypt more messages. All messages are chosen from the set $\{-1, 0, 1\}^n$. By increasing the message space, the attack time may increase a lot. In [Lud02], Ludwig gets to a similar conclusion, namely he states that a lattice dimension of over 782 has to be used to make the cryptosystem secure. But he also has expressed doubts, if this lower bound really is secure enough, since recent progress in reduction algorithms. He conjectured that Schnorr's lattice reduction algorithm based on random sampling and birthday methods [Sch03] might be able to succeed in dimensions up to 2000.

6.3 Practicality

6.3.1 Space and Time Analysis

Both, the public key and the private key are square matrices, so a priori they use the space of n^2 integers. The **private key** contains only values chosen from a set with constant length, except for the diagonals, which are about $\mathcal{O}(\sqrt{n})$, and thus also less then n. So the matrix consists of n^2 integers, which can be stored with $\log n$ space. The **public key** is a matrix in Hermite normal form, which also can be represented in $\mathcal{O}\left(n^2 \log n\right)$ bits. A **ciphertext** is a vector that contains n entries which are taken modulo the public basis, and thus can be stored with $\mathcal{O}(n \log n)$ bits.
Generating the **private key** is simply choosing a random matrix, which is a very easy

task nowadays and can be done in $\mathcal{O}\left(n^2\right)$ time. The only task to generate the **public key** is to transform the randomly chosen matrix into Hermite normal form. We found many different algorithms solving this task with various complexities. We take the linear space algorithm of Micciancio and Warinschi [MW01] with time complexity of $\mathcal{O}\left(n^5 \log^2 n\right)$ as a reference. The **encryption** is simply the computation of a vector modulo a matrix in HNF, which can be achieved in $\mathcal{O}\left(n^2\right)$ steps using Formula (6.1). For **decryption** there are two natural choices of algorithms, namely the Babai's round-off and Babai's nearest plane algorithm. With $\mathcal{O}\left(n^2\right)$ time complexity, the round-off algorithm is relatively fast. The nearest plane algorithm includes the computation of Gram-Schmidt matrix, which makes it relatively slow compared to round-off. However, this computation has only to be done once, and can be precomputed in $\mathcal{O}\left(n^3\right)$ time, and then the algorithm needs only $\mathcal{O}\left(n^2\right)$ steps.
All these quantities are summarized in Figure 6.3 for a better overview.

Space	
Object	Size
Private Key	$\mathcal{O}\left(n^2 \log n\right)$
Public Key	$\mathcal{O}\left(n^2 \log n\right)$
Ciphertext	$\mathcal{O}\left(n \log n\right)$

Time	
Operation	Time
Private Key Generation	$\mathcal{O}\left(n^2\right)$
Public Key Generation	$\mathcal{O}\left(n^5 \log^2 n\right)$
Encryption	$\mathcal{O}\left(n^2\right)$
Decryption	$\mathcal{O}\left(n^2\right)$

Figure 6.3: Space and Time Complexities of MCS

6.3.2 Practical Tests

In this section we discuss the experiments we made to analyze MCS. Besides the times and the asymptotic behavior in practice we also wanted to study two other questions. Since the computation of the Gram-Schmidt basis takes an extreme amount of time (see Figure 6.4), we want to find out, if this computation is really necessary. The Gram-Schmidt basis is used for computing the value of σ and for Babai's nearest plane algorithm. The latter can be replaced with the faster round-off algorithm, which manage the task without computing the Gram-Schmidt basis. Further, we want to study how large the impact of reducing the private basis matrix is on the value of σ.
To see the impact of reducing the private basis on σ we run the following experiment in different dimensions. We randomly chose a basis as Micciancio proposed it and computed the value of σ. After that we reduced the private basis and computed σ again. We have done this for dimensions up to 700, after that, the computation of Gram-Schmidt went more than 36 hours. The result of this experiment is shown in Figure 6.5, together with the norm of the vectors consisting only of 1's, and only of 2's, respectively. As one can

observe, the reduction of the private key has mostly a positive impact on the value of σ, but even with reduced private key, the Babai's nearest plane algorithm is never capable to ensure the correct decryption of the all-two vector. So the message has to be chosen from the set $\{-1, 0, 1\}$, to ensure correct decryption. Of course we could also choose the message from the $\{-2, \ldots, 2\}$ with restriction that the norm has to be smaller than a certain amount, but this has a bad effect on the security.

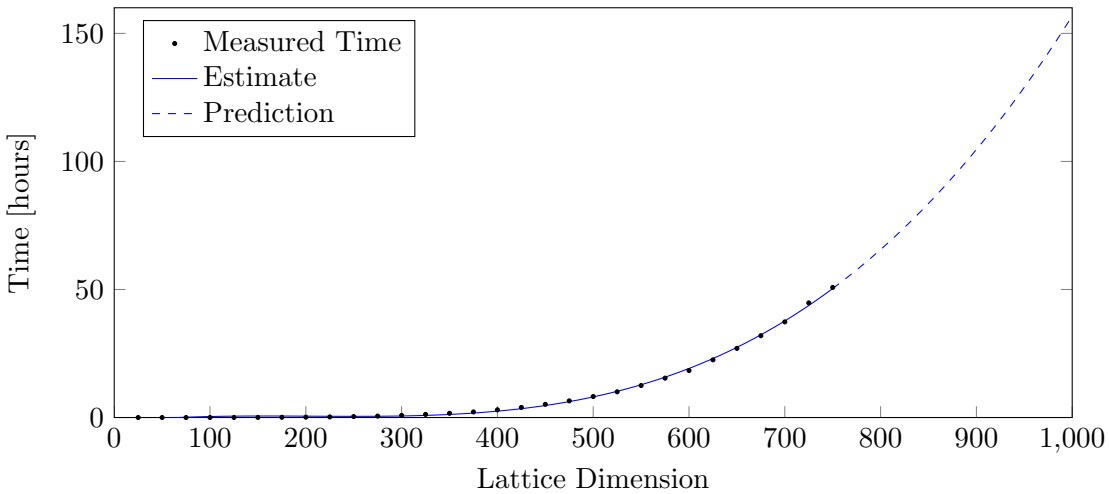

Figure 6.4: Time for Computing Gram-Schmidt Basis

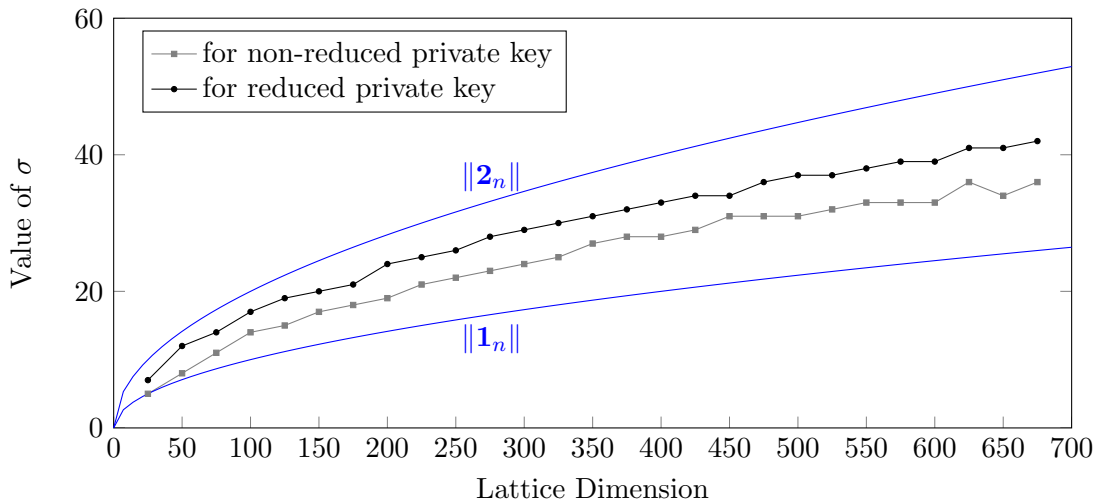

Figure 6.5: Values of σ

According to our experiments, it is not necessary to reduce the private key to enlarge the message space. One could argue, that it is possible to reduce the private key after the

key generation phase to enlarge the probability of error-less decryption with the private key, but the impact is not very big, as can be seen in Figure 6.5.
As for GGH, we also analyzed the decryption with too high error bounds in MCS. We emphasize here, that the comparison to GGH is not quite fair, since we analyzed GGH with round-off and MCS with the more accurate nearest-plane method. So it is more a comparison of the used algorithms than of the cryptosystems. However, the authors of GGH suggested the round-off algorithm, while Micciancio recommended the nearest-plane method for decrypting, so we compared them as designated by the authors.
The results can be seen in Figure 6.6. The difference to GGH with round-off is conspicuous: In dimension 300, the decryption failures for messages decrypted with $\sigma = 12$ is under 1%. In general, for every error bound σ the decryption failures are much lower compared to GGH in the same dimension.

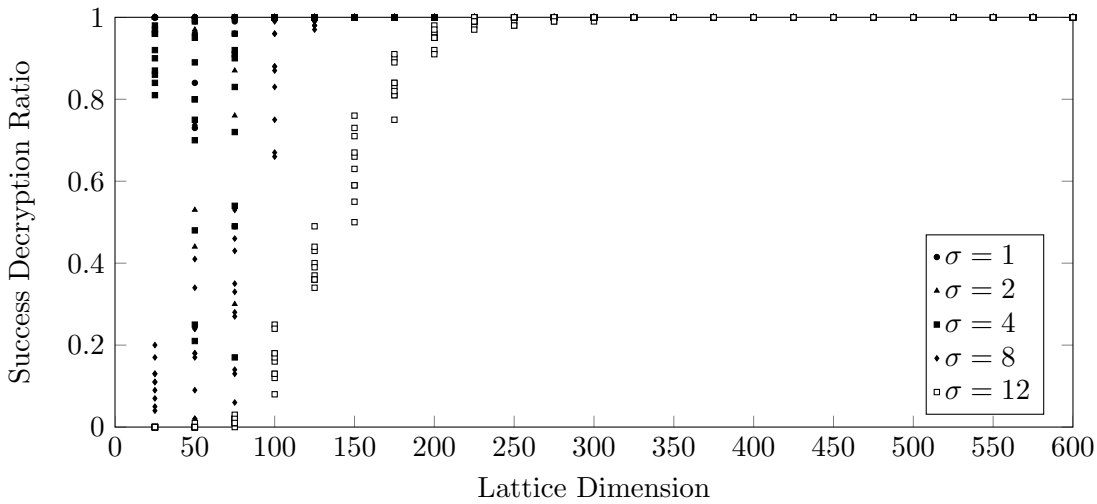

Figure 6.6: Success of Decrypting with Different σ's

Let us now look at the time for key generation, encryption and decryption. For dimensions up to 750, we measured the time to generate one key pair. For every pair we measured the time for encrypting and decrypting 10 messages. Generation of the private key is exactly the same as in GGH, so for these results we refer to Figure 5.9. All measured times are under half a second. In contrast the generation of the public key took half a minute for dimension 750 (Figure 6.7). However, this is an improvement of about a factor of 4 compared to the public key generation in GGH.

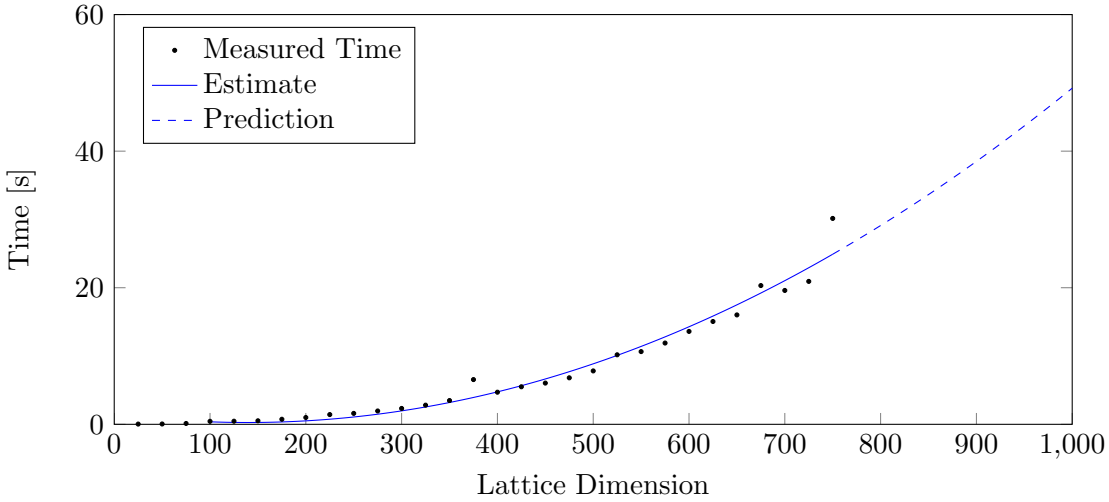

Figure 6.7: Time for Generating a Public Key in MCS

Figure 6.8 shows the time to encrypt a message. All encryptions went faster than 400ms. For decryption, we wanted to analyze the difference in the time of Babai's round-off and nearest plane algorithms. The difference could not have been clearer, in dimension 750, the round-off successfully decrypted the message in 3 seconds, whereas the nearest plane algorithm ran over 20 minutes. Figure 6.9 shows both time series in a log-log scale.

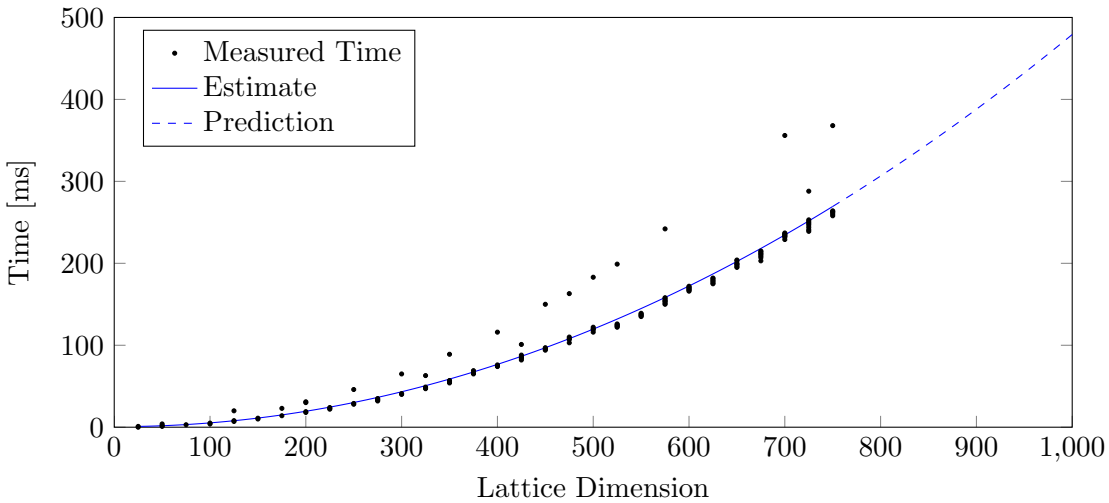

Figure 6.8: Time for Encrypting a Message in MCS

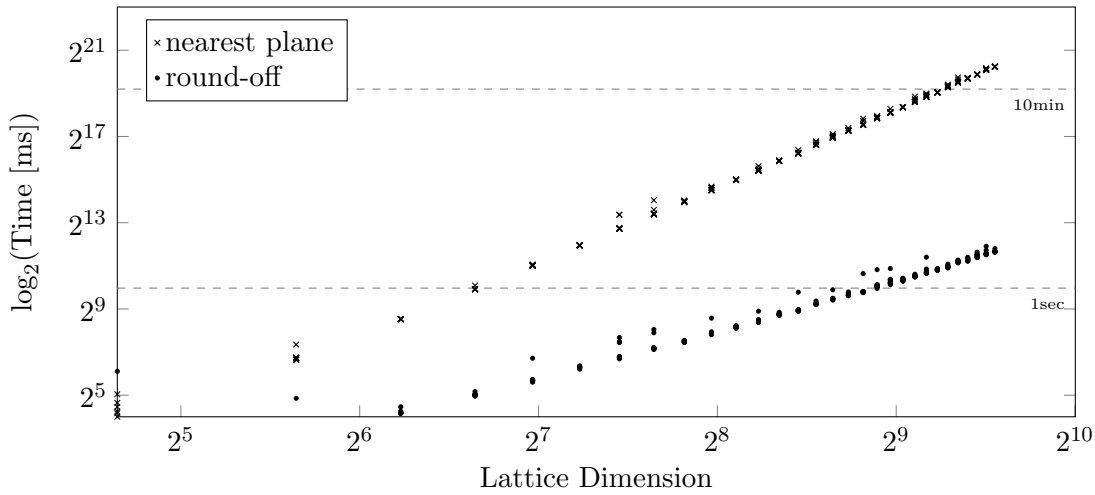

Figure 6.9: Time for Decrypting a Message in MCS

Key and Ciphertext Size

In this section we present the experiments we made for measuring the size of the private key, the public key and the ciphertext. All sizes are measured as follows. The matrices and vectors are stored as plain text into a file, which then was compressed with gzip at compression ratio of 9 (best compression for gzip). The sizes we show are the file sizes of the resulting files.

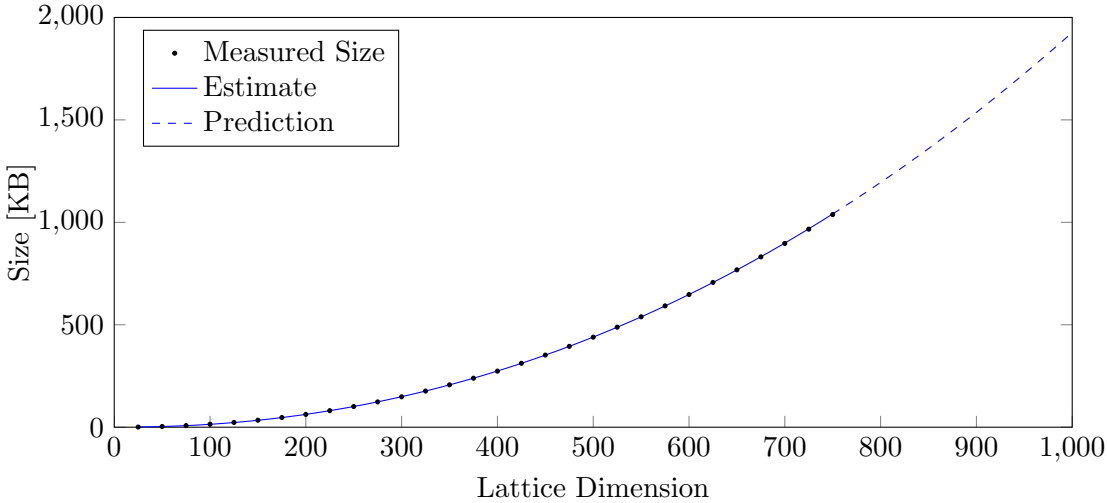

Figure 6.10: Size of a Private Key in MCS

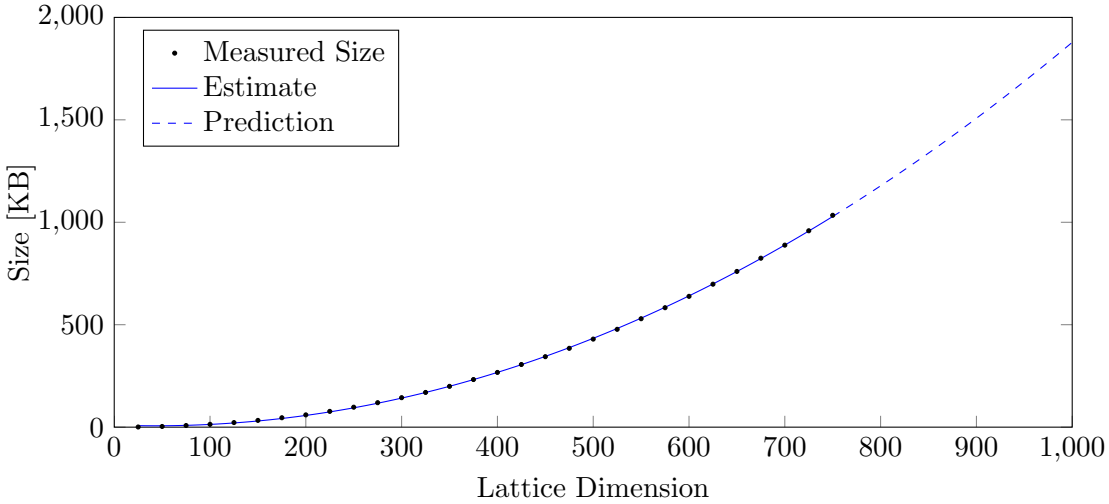

Figure 6.11: Size of a Public Key in MCS

As one can observe in Figures 6.10 and 6.11, the private and the public key are almost of the same size. Since the public key is the HNF of a matrix, we stored it in as a sparse matrix, i.e. only the non-zero entries were stored.
Compared to GGH, both the public key size and the ciphertext size are improved by about one order of magnitude. The ciphertext size is almost linear in MCS as can be seen in Figure 6.12.

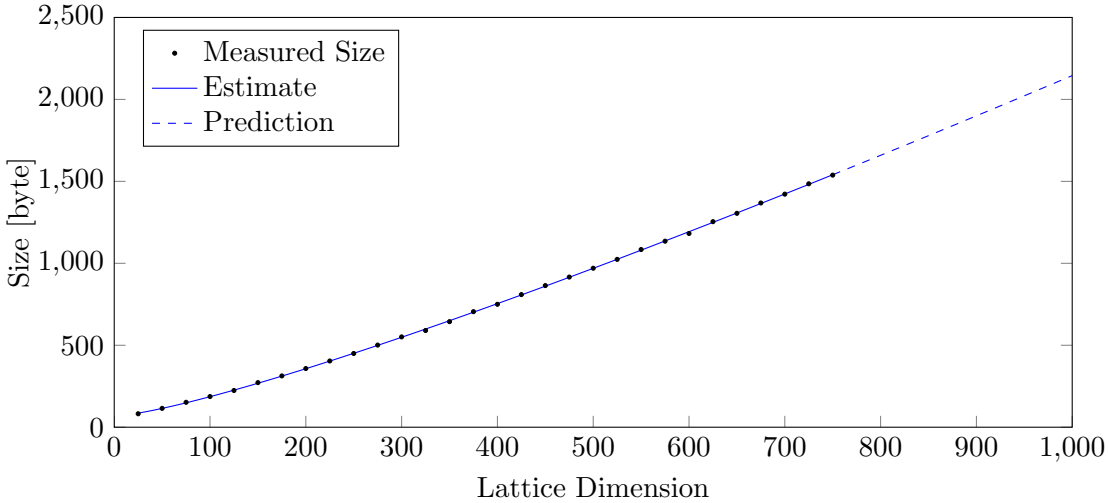

Figure 6.12: Size of a Ciphertext in MCS

6.4 Conclusion

In this section we discussed an improvement of GGH by Micciancio. He was able to decrease the size of the public key and the ciphertext by one order of magnitude, but with the cost of slightly slower encryption time. The security of the system stays the same or even increases with the presented way of generating the public key. The messages are encrypted in the error vector, which makes the system no longer vulnerable to the attack of Nguyen, but simultaneously shrink the message space to values in the set $\{-\sigma, \ldots, \sigma\}$. However, decrypting with Babai's nearest plane method allows slightly higher values for σ than the decryption with round-off. A huge drawback of nearest plane is the need for computing the Gram-Schmidt orthogonalization, which is extremely time-consuming and thus makes the cryptosystem extremely impractical. The computation is used to determine the upper bound for the error vector, i.e. the message space, and is an essential part in the nearest-plane method. However, we showed that this computation is not needed if the faster round-off method is used and if one is able to encode the message with an error-detecting code.

Since the attack of Nguyen is not applicable to MCS, the most promising attack is probably the embedding attack, which is highly dependent on the length of the error vector. We showed that both, the round-off and the nearest plane method, are able to decrypt messages which are encrypted with too high error values to a certain degree (see Figures 5.13 and 6.6).

We conclude that MCS in its originally presented way is not practicable since, the computation of Gram-Schmidt is too time-consuming. However, by using Babai's round-off together with a error-detecting code, the cryptosystem looks quite promising.

Chapter 7

NTRU Cryptosystem

The NTRU cryptosystem was invented by the three authors Hoffstein, Pipher and Silverman and was presented in 1998 [HPS98]. It is based on truncated polynomial rings with convolution multiplication where it has its initial name from (**N**-th degree **tru**ncated polynomial ring), but it can also be expressed using lattices. The aim of this section is to present the cryptosystem using polynomial rings and lattices as well as give some attacks against the private key and the ciphertext followed by a discussion about the practicality of the system.

7.1 Introduction to Truncated Polynomial Rings

In the following we consider $R := \mathbb{Z}[X]/(X^N - 1)$, where $N \in \mathbb{Z}$ is a security parameter, i.e. R is the ring of polynomials with integer coefficients over the variable X modulo the polynomial $X^N - 1$. So the maximal degree of the polynomials is $N - 1$, since we have the equation

$$X^N = 1 \in R$$

This ring has the usual componentwise addition: Let $a(X) = \sum_{i=0}^{N-1} a_i \cdot X^i$ and $b(X) = \sum_{i=0}^{N-1} b_i \cdot X^i \in R$, then

$$a(X) + b(X) = \sum_{i=0}^{N-1} (a_i + b_i) \cdot X^i,$$

and the convolution multiplication in R, defined as follows: $c(X) := a(X) \circledast b(X)$, where

$$c_k := \sum_{i=0}^{k} a_i \cdot b_{k-i} + \sum_{i=k+1}^{N-1} a_i \cdot b_{N+k-i} = \sum_{i+j \equiv k \mod N} a_i \cdot b_j$$

Remark. The convolution product can also be described as a matrix operation. For this, we first define the cyclic rotation C that sends a vector $(x_1, x_2, ..., x_n)^T$ to $(x_n, x_1, ..., x_{n-1})^T$.

For an arbitrary $\boldsymbol{x} \in \mathbb{R}^n$ the circulant matrix of $\boldsymbol{x}$ is defined as

$$[C^*\boldsymbol{x}] := [\boldsymbol{x}, C\boldsymbol{x}, \dots, C^{n-1}\boldsymbol{x}] = \begin{pmatrix} x_1 & x_n & \dots & x_2 \\ x_2 & x_1 & \dots & x_3 \\ \vdots & \vdots & \ddots & \vdots \\ x_n & x_{n-1} & \dots & x_1 \end{pmatrix}$$

We now have that $f \circledast g$ is equivalent to the matrix operation $[C^*\boldsymbol{f}]\boldsymbol{g}$ if we consider $\boldsymbol{f}$ and $\boldsymbol{g}$ as vectors consisting of the coefficients of f and g. This operation is associative, commutative and distributive, so $(R, +, \circledast)$ forms a ring. Further, the following equations hold:

$$[C^*\boldsymbol{f}](\boldsymbol{g_1} + \boldsymbol{g_2}) = [C^*\boldsymbol{f}]\boldsymbol{g_1} + [C^*\boldsymbol{f}]\boldsymbol{g_2} \tag{7.1}$$

$$[C^*\boldsymbol{f}][C^*\boldsymbol{g}] = [C^*([C^*\boldsymbol{f}]\boldsymbol{g})] \tag{7.2}$$

$$[C^*(C\boldsymbol{f})]\boldsymbol{g} = [C^*\boldsymbol{f}](C\boldsymbol{g}) \tag{7.3}$$

Definition 35. When we reduce a polynomial $p(X)$ modulo say q, we mean to reduce the coefficients of $p(X)$ modulo q, so that the result $\bar{p}(X) \in R/(q) = \mathbb{Z}[X]/(X^N - 1, q)$.

7.2 Description of the System

In this section we describe the original cryptosystem as described in [HPS98] with a slightly different notation. Later on, many improvements are presented, like a digital envelope [HLPS99]. We assume, that Alice wants to send a message to Bob. Like in every public key cryptosystem, Bob has to create his private and public key, then Alice is able to encrypt a message with Bob's public key and finally Bob can decrypt the message with his private key. We discuss how these three steps look like in NTRU cryptosystem in detail and then explain why this works.

Notation 5. The NTRU-system has the three integer parameters N, p and q to control the security and practicality of the system. p and q need not to be prime, but coprime ($\gcd(p, q) = 1$). Further, d_f, d_g and d_r are integer bounds for the coefficients of the polynomials f, g and r in R and control the probability of decryption errors.

$$\mathcal{L}(d_1, d_2) := \{p(X) \in R \,|\, p \text{ has } d_1 \text{ 1's and } d_2 \text{ -1's, and the rest of the coefficients are } 0\}$$

We define the spaces $\mathcal{L}_f$, $\mathcal{L}_g$ and $\mathcal{L}_r$ as follows:

$$\mathcal{L}_f := \mathcal{L}(d_f, d_f - 1), \qquad \mathcal{L}_g := \mathcal{L}(d_g, d_g), \qquad \mathcal{L}_r := \mathcal{L}(d_r, d_r)$$

Remark. Note that in contrast to $\mathcal{L}_g$ and $\mathcal{L}_r$, $\mathcal{L}_f$ has one -1 less than $+1$'s. This is because $f \in \mathcal{L}_f$ has to be invertible, but if it had as many -1's as 1's f would satisfy $f(1) = 0$, i.e. it would not be invertible.

Key Creation

To create the private and the public key, Bob randomly chooses polynomials $f \in \mathcal{L}_f$ and $g \in \mathcal{L}_g$. f must be invertible modulo p and q, denote the inverses f_p^{-1} and f_q^{-1}, respectively, so we have

$$f_p^{-1} \circledast f = 1 \mod p \tag{7.4}$$

$$f_q^{-1} \circledast f = 1 \mod q. \tag{7.5}$$

Then, Bob computes the public key

$$h := p \cdot f_q^{-1} \circledast g \mod q \tag{7.6}$$

or in matrix notation

$$\boldsymbol{h} := p \cdot [C^* \boldsymbol{f}_q^{-1}]\boldsymbol{g} \mod q \tag{7.6'}$$

while f and g remain private.

Encryption

To encrypt a message $\boldsymbol{m} \in \mathbb{Z}_q^n$ (denote the corresponding polynomial by m), Alice randomly chooses a polynomial $r \in \mathcal{L}_r$ and computes

$$c := h \circledast r + m \mod q. \tag{7.7}$$

or in matrix notation,

$$\boldsymbol{c} := [C^* \boldsymbol{h}]\boldsymbol{r} + \boldsymbol{m} \mod q. \tag{7.7'}$$

Decryption

To decrypt the message c, Bob first reduces the entries of the vector

$$t \equiv f \circledast c \mod q, \tag{7.8}$$

such that all values t_i are bounded by $-\frac{q}{2} \leq t_i < \frac{q}{2}$. The original message can then be computed by

$$m \equiv f_p^{-1} \circledast c \mod q. \tag{7.9}$$

Or equivalently in matrix notation

$$\boldsymbol{t} \equiv [C^* \boldsymbol{f}]\boldsymbol{c} \mod q, \tag{7.8'}$$

$$\boldsymbol{m} \equiv [C^* \boldsymbol{f}]_p^{-1}\boldsymbol{c} \mod q. \tag{7.9'}$$

Remark. To provide a better overview we summarize the parameters together with their domains and visibilities in the following table:

Parameter	Domain	Visibility
N	$\mathbb{N}$	public
p	$\mathbb{N}$	public
q	$\mathbb{N}$	public
d_f	$\mathbb{N}$	public
d_g	$\mathbb{N}$	public
d_r	$\mathbb{N}$	public
f	$\mathbb{Z}[X]/(X^N-1)$	private
g	$\mathbb{Z}[X]/(X^N-1)$	private
h	$\mathbb{Z}[X]/(X^N-1)$	public

Example 15 (NTRU System)**.** The aim of this example is to show the cryptosystem in practice for $N=5$, small modulus $p=3$ and large modulus $q=16$, $d_f=2$, $d_g=2$ and $d_r=1$. We randomly chose the polynomials

$$f = X^2+X-1 \text{ and } g = -X^4+X^2-X+1.$$

as private key and computed the inverse of f modulo p and q respectively, which are $f_p^{-1} = X^4+2X^3+2X^2+2$ and $f_q^{-1} = 15X^4+12X^3+3X^2+9X+10$ and the corresponding circulant matrices are

$$[C^*\boldsymbol{f}]_p^{-1} = \begin{pmatrix} 2 & 1 & 2 & 2 & 0 \\ 0 & 2 & 1 & 2 & 2 \\ 2 & 0 & 2 & 1 & 2 \\ 2 & 2 & 0 & 2 & 1 \\ 1 & 2 & 2 & 0 & 2 \end{pmatrix} \qquad [C^*\boldsymbol{f}]_q^{-1} = \begin{pmatrix} 10 & 15 & 12 & 3 & 9 \\ 9 & 10 & 15 & 12 & 3 \\ 3 & 9 & 10 & 15 & 12 \\ 12 & 3 & 9 & 10 & 15 \\ 15 & 12 & 3 & 9 & 10 \end{pmatrix}.$$

By (7.6), h becomes $4X^4+9X^3+8X^2+X+10$, and so

$$[C^*\boldsymbol{h}] = \begin{pmatrix} 10 & 4 & 9 & 8 & 1 \\ 1 & 10 & 4 & 9 & 8 \\ 8 & 1 & 10 & 4 & 9 \\ 9 & 8 & 1 & 10 & 4 \\ 4 & 9 & 8 & 1 & 10 \end{pmatrix}$$

Let the message be the vector $\boldsymbol{m} = (-1,\ 0,\ 1,\ 1,\ 0)$. We randomly chose the polynomial $r = -X^4+X$ and computed the ciphertext $c = h \circledast r + m$ which is $-X^4+5X^3-7X^2+2X+2 \mod q$, or equivalently

$$[C^*\boldsymbol{h}]\boldsymbol{r} + \boldsymbol{m} \equiv (2,\ 2,\ -7,\ 5,\ -1) \mod q$$

To decrypt this ciphertext we first compute the product $\boldsymbol{a} \equiv [C^* \boldsymbol{f}]\boldsymbol{c}$, which is

$$(2,\ -1,\ 11,\ -10,\ -1)\,,$$

then align its coefficients to values x between $-\frac{16}{2} \leq x < \frac{16}{2}$, which gives the vector $\boldsymbol{a}' = (2,\ -1,\ -5,\ 6,\ -1)$. Multiplying this vector from the left by $[C^* \boldsymbol{f}]_p^{-1}$ gives the original message $m = (2,\ 0,\ 1,\ 1,\ 0) \bmod p$.

7.2.1 Why This Works

The first step in the decryption is to multiply c from the left by f, so consider $t = f \circledast c$.

$$\begin{aligned} t &= f \circledast c \\ &= f \circledast (h \circledast r + m) \\ &= f \circledast (h \circledast r) + f \circledast m \\ &= f \circledast (p \cdot f_q^{-1} \circledast g) \circledast r + f \circledast m \\ &\equiv p \cdot g \circledast r + f \circledast m \qquad (\text{mod } q). \end{aligned} \tag{7.10}$$

Next step is to reduce t such that all coefficients lie in the interval $[-\frac{q}{2}, \frac{q}{2})$. Depending on the parameters of the system, it is very likely or even certain that this process restores the original coefficients of $p \cdot g \circledast r + f \circledast m$ in $\mathbb{Z}[X]$, although it is possible that the decryption does not return the originally message. In Section 7.2.2, we go deeper into this issue. This decryption error can be used by Eve to get information about the private key f (see [GN07] for more information).
By multiplying t with f_p^{-1} we get:

$$\begin{aligned} f_p^{-1} \circledast t &= f_p^{-1}(p \cdot g \circledast r + f \circledast m) \\ &= 0 + f_p^{-1} \circledast f \circledast m \\ &\equiv m \qquad (\text{mod } p). \end{aligned} \tag{7.11}$$

7.2.2 Decryption Errors

In the decryption process of NTRU it may happen that the decrypted message is not equivalent to the original message. The probability of decryption failure depends on N, d_f, d_g, d_r and also on m. In this section we discuss how these errors occur.
The crucial point for a correct decryption is that in (7.10), t should be equal to

$$p \cdot r \circledast g \circledast m \circledast f \in \mathcal{P}.$$

Equality mod q ($t \equiv p \cdot r \circledast g \circledast m \circledast f \in \mathcal{P}_q$) does not suffice. To demonstrate this we assume that the equation is true in $\mathcal{P}_q$ but not in $\mathcal{P}$, i.e. there exists $\boldsymbol{e} \in \mathbb{Z}^N$, such that

$$t = (p \cdot r \circledast g \circledast m \circledast f) + q \cdot \boldsymbol{e} \in \mathcal{P}.$$

Since p and q are co-prime the term $(q \cdot \boldsymbol{e})$ is not equal to $\mathbf{0}$ in $\mathbb{Z}_p^N$ with high probability, so

$$\begin{aligned} f_p^{-1} \circledast t &= f_p^{-1}(p \cdot g \circledast r + f \circledast m) + f_p^{-1} \circledast (q \cdot \boldsymbol{e}) \\ &= 0 + f_p^{-1} \circledast f \circledast m + f_p^{-1} \circledast (q \cdot \boldsymbol{e}) \\ &= m + f_p^{-1} \circledast (q \cdot \boldsymbol{e}) \\ &\not\equiv m \pmod{p} \end{aligned}$$

in general.

Remark. Alice cannot be sure that the decryption will not fail, because thereto she would need the private key f.

Parameters

In [HPS98] the authors presented a set of parameters in different dimensions, displayed in Table 7.1. Note that p is always chosen to be 3, this makes the inversion of f modulo p simpler. Further, N is always prime and q a power of 2. In [Sil99] they give optimized algorithms for inversion of polynomials modulo primes and prime powers.

	N	p	q	d_f	d_g	d_r
NTRU167	167	3	128	61	20	18
NTRU263	263	3	128	50	24	16
NTRU503	503	3	256	216	72	55

Table 7.1: NTRU Parameter Sets

7.3 Lattice Attacks on NTRU

There are quite a few attacks on NTRU cryptosystem such as brute force, meet-in-the-middle, multiple transmission attack and even a ciphertext attack if a decryption error occurs [HGNP+03]. In this thesis we are are more interested in attacks based on lattices, which are anyway more promising attacks, since decryption errors can be minimized (about every 2^{15} encryption, and one needs a few hundreds of them [GN07]) and for brute force attacks, the sets of possible candidates are too large ($> 2^{213}$ for NTRU-2005). We first show an attack on the private key presented in [CS97] and then a related attack on the ciphertext.

7.3.1 Attack on the Private Key

In this section we show an attack on the private key. It was first presented by Don Coppersmith and Adi Shamir [CS97] in 1997. It is based on lattices and takes advantage

of the good performance of the LLL algorithm. The goal is to find the secret key (f, g) just from the public key. The attack is based on the following $(2N \times 2N)$-lattice determined by $\boldsymbol{h}$ and p, which are both public:

Definition 36 (NTRU-Lattice)**.** The NTRU-Lattice is the lattice spanned by the basis

$$\mathbf{L} := \begin{pmatrix} \mathbf{I_N} & \mathbf{0} \\ \mathbf{H} & q \cdot \mathbf{I_N} \end{pmatrix}, \tag{7.12}$$

where $\mathbf{0}$ denotes the $N \times N$ matrix containing only 0's and $\mathbf{H} = p_q^{-1}[C^*\boldsymbol{h}]$, where p_q^{-1} denotes the multiplicative inverse of p in $\mathbb{Z}_q$.

The volume of $\mathbf{L}$ is q^N and the dimension $2N$. This means that the length of the shortest vector is approximately $\det(L)^{\frac{1}{2N}} = \sqrt{q}$ as described in [BC12]. The vector $(\boldsymbol{f}, \boldsymbol{g})$ and circular shifts are also in the NTRU-lattice generated by $\mathbf{L}$. Further, it is very likely that this vector is a shortest vectors, since its norm is $\sqrt{2 \cdot d_f - 1 + 2 \cdot d_g} < \sqrt{q}$ in the cases recommended by the NTRUEncrypt. So, by reducing $\Lambda(\mathbf{L})$ with the LLL algorithm, the probability is high to find $(\boldsymbol{f}, \boldsymbol{g})$ or circular shifts of it, which can also be used to decrypt the message.

Example 16 (Private Key Attack on NTRU)**.** In this example we demonstrate the attack of Coppersmith and Shamir by attacking the private key from Example 15. The first step is to construct $\mathbf{L}$ from the public key $h = 4X^4 + 9X^3 + 8X^2 + X + 10$, so $\mathbf{H} = p_q^{-1} \cdot [C^*\boldsymbol{h}] = 8 \cdot [C^*\boldsymbol{h}]$:

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 110 & 44 & 99 & 88 & 11 & 16 & 0 & 0 & 0 & 0 \\ 11 & 110 & 44 & 99 & 88 & 0 & 16 & 0 & 0 & 0 \\ 88 & 11 & 110 & 44 & 99 & 0 & 0 & 16 & 0 & 0 \\ 99 & 88 & 11 & 110 & 44 & 0 & 0 & 0 & 16 & 0 \\ 44 & 99 & 88 & 11 & 110 & 0 & 0 & 0 & 0 & 16 \end{pmatrix}$$

Applying a reduction algorithm to this lattice leads to a reduced basis like the following:

$$\mathbf{L}' = \begin{pmatrix} 1 & 0 & 0 & -1 & -1 & 3 & -2 & -2 & 2 & 2 \\ 1 & -1 & 0 & 0 & 1 & -2 & 4 & -1 & -3 & -2 \\ 1 & 1 & 1 & 1 & 1 & 3 & -1 & 3 & 1 & 1 \\ 1 & 1 & -1 & 0 & 0 & -2 & 3 & -2 & 2 & -1 \\ 1 & 0 & -1 & -2 & 0 & -1 & -2 & 3 & -2 & 1 \\ 0 & -1 & 0 & -1 & 1 & 0 & 3 & 2 & 5 & -6 \\ 0 & 1 & 1 & 1 & -1 & -5 & -1 & 2 & -2 & -5 \\ 0 & -1 & -1 & 0 & 1 & 1 & -4 & 0 & -1 & -1 \\ 0 & 1 & 1 & 0 & 0 & 2 & 1 & -5 & -3 & -1 \\ 0 & 0 & -1 & 0 & -1 & 2 & 1 & 1 & 1 & -3 \end{pmatrix}$$

Consider the fifth column vector. It contains exactly as many 1's and -1's we expect for f and g and coincide exactly with

$$(\boldsymbol{f}, \boldsymbol{g}) = (-1,\ 1,\ 1,\ 0,\ 0,\ 1,\ -1,\ 1,\ 0,\ -1)$$

corresponding to the polynomials $f = X^2 + X - 1$ and $g = -X^4 + X^2 - X + 1$. Taking a closer look at $\mathbf{L}'$, we see that also the second column vector has also the expected quantity of 1's and -1's. This vector contains circular shifts of $(-1,\ 1,\ 1\,0\,0)$ and $(1,\ -1,\ 1,\ 0,\ -1)$. Indeed, also these vectors can be used for encryption and decryption.

Why the Attack Works

Recall the definition of the public key $h := p \cdot f_q^{-1} \circledast g \mod q$ or equivalently

$$f \circledast h := p \cdot g \mod q. \tag{7.13}$$

Consider now

$$\Lambda_H := \{(u, v) \in \mathcal{P} \times \mathcal{P} \,|\, u \circledast \mathbf{H} = v \qquad (\text{mod } q)\}$$

where $\mathbf{H} = p^{-1} \cdot h \mod q$. By (7.13) it is clear, that $(f, g) \in \Lambda_H$. Note that if (u_1, v_1) and (u_2, v_2) are in Λ_H, then any $\mathbb{Z}$-linear combination $\lambda_1 \cdot (u_1, v_1) + \lambda_2 \cdot (u_2, v_2)$ is also in Λ_H. So, by identifying $\mathcal{P}$ with $\mathbb{Z}^N$ we see that Λ_H is a subgroup of $\mathbb{Z}^{2N}$ and thus a lattice in $\mathbb{R}^{2N}$.

Note that the NTRU lattice generated by (7.12) is exactly the lattice Λ_H. By reducing Λ_H the chances are good to find at least one vector of the form $(\boldsymbol{f}, \boldsymbol{g})$, because of the fact, that it has small norm compared to the other vectors in the lattice.

Practical Tests

Figure 7.3.1 shows the times for successful attacks together with the needed block size in dimensions 29, 53, 79 and 101, respectively. The attacks in higher dimension didn't terminate within a week, so we aborted them.

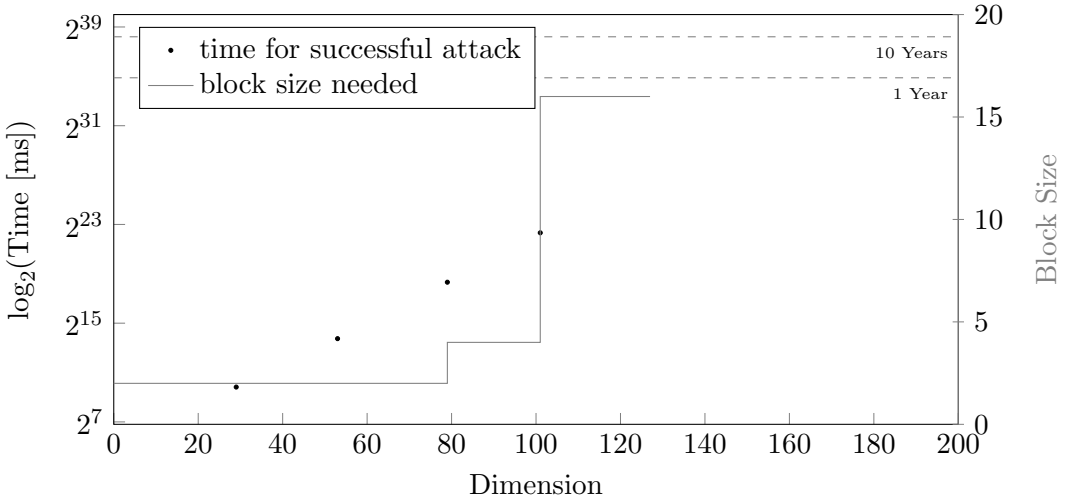

Figure 7.1: Time for Lattice Attack on the Private Key

7.3.2 Attack on the Ciphertext

In this section we discuss another attack based on the idea of the authors of [HPS98]. The basic concept is very similar to the attack on the private key. Analogously, we define a lattice $\Lambda_{h,c}$, given the public key h and the ciphertext c as follows.

$$\begin{pmatrix} \mathbf{I_N} & \mathbf{0} & \mathbf{0} \\ \mathbf{H} & q \cdot \mathbf{I_N} & \boldsymbol{c} \end{pmatrix} \tag{7.14}$$

where $\mathbf{H} = [C^* \boldsymbol{h}]$ (not $p^{-1}[C^* \boldsymbol{h}] \mod q$ like in the last attack). So this is a basis for the lattice containing all the points

$$\{(u, v) \in \mathcal{P} \times \mathcal{P} \mid u \circledast h = v \pmod q\}$$

and additionally the point $\boldsymbol{v}_c := (\mathbf{0}, \boldsymbol{c})$. Recall that $c = r \circledast h + m \mod q$. Consider now the point $\boldsymbol{v}_r := (r, r \circledast h) \in \Lambda_{h,c}$. Since $\boldsymbol{v}_c$ and $\boldsymbol{v}_r$ are both in $\Lambda_{h,c}$, also $\boldsymbol{v}_c - \boldsymbol{v}_r$ is in the lattice.

$$\begin{aligned} \boldsymbol{v}_c - \boldsymbol{v}_r &= (\mathbf{0}, c) - (r, r \circledast h) \\ &= (-r, c - r \circledast h) \\ &= (-r, r \circledast h - r \circledast h + m) \\ &= (-r, m) \end{aligned}$$

Only to be an element of the lattice does not suffice to find this element in the lattice, but when we take a closer look on the norm of this vector, we see that it is at most $\sqrt{2d_r + N}$ (if the message was the all one vector of length N). Compared to the norm of the expected shortest vector, which is $\sqrt{\frac{q^2 N}{2\pi e}}$, the desired vector is relatively small, and therefore has a

good chance to be the shortest vector. This means, that we can reduce the lattice $\Lambda_{h,c}$ until we find a vector with the right amount of 1's and -1's in the first N entries. The rest of the entries will be the message with very high probability.

Example 17. We demonstrate this attack on the NTRU instance chosen in Example 15. The first step is to construct the lattice basis (7.14) for the lattice $\Lambda_{h,c}$ with the public key equals $4X^4 + 9X^3 + 8X^2 + X + 10$ and the ciphertext equals $-X^4 + 5X^3 - 7X^2 + 2X + 2$.

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 10 & 4 & 9 & 8 & 1 & 16 & 0 & 0 & 0 & 0 & 2 \\ 1 & 10 & 4 & 9 & 8 & 0 & 16 & 0 & 0 & 0 & 2 \\ 8 & 1 & 10 & 4 & 9 & 0 & 0 & 16 & 0 & 0 & -7 \\ 9 & 8 & 1 & 10 & 4 & 0 & 0 & 0 & 16 & 0 & 5 \\ 4 & 9 & 8 & 1 & 10 & 0 & 0 & 0 & 0 & 16 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Then we reduce this with the LLL algorithm, resulting in the following matrix.

$$L' = \begin{pmatrix} -1 & 0 & -1 & -2 & 0 & 3 & -1 & -2 & 0 & 0 & 0 \\ -1 & -1 & 0 & 1 & 0 & -1 & 1 & 3 & 2 & -1 & 1 \\ -1 & 0 & -1 & 1 & 0 & 0 & 1 & -1 & 2 & -1 & -1 \\ -1 & 0 & 0 & 1 & 2 & 1 & -1 & 0 & -3 & 0 & 2 \\ -1 & 1 & 0 & -1 & 0 & -2 & -1 & -1 & 0 & 0 & -2 \\ 0 & -1 & 1 & 2 & -2 & -2 & 0 & -2 & 0 & 1 & 3 \\ 0 & 0 & -1 & -1 & 0 & 0 & 2 & 0 & -1 & 0 & 2 \\ 0 & 1 & 0 & -1 & -1 & 0 & 1 & 0 & 1 & -4 & 2 \\ 0 & 1 & 0 & 2 & -1 & 0 & 1 & 1 & -1 & 2 & 4 \\ 0 & 0 & 2 & -1 & 3 & 1 & -1 & 1 & 0 & 0 & 2 \\ 0 & 1 & 2 & 1 & -1 & -1 & 3 & 0 & -1 & -1 & -3 \end{pmatrix}$$

Recall that $d_r = 1$, so we search for a column vector in the reduced basis whose 5 first entries contains a 1 and a -1. This holds for the 2nd and 10-nth column. By checking the next 5 entries, we see that the second vector contains the message $(-1, 0, 1, 1, 0)$.

Practical Tests

Figure 7.3.2 shows the times for successful attacks in dimensions 29, 53 and 79 respectively and the average block size needed for BKZ to reduce the lattice. The attacks in higher dimension didn't terminate within a week, so we aborted them.

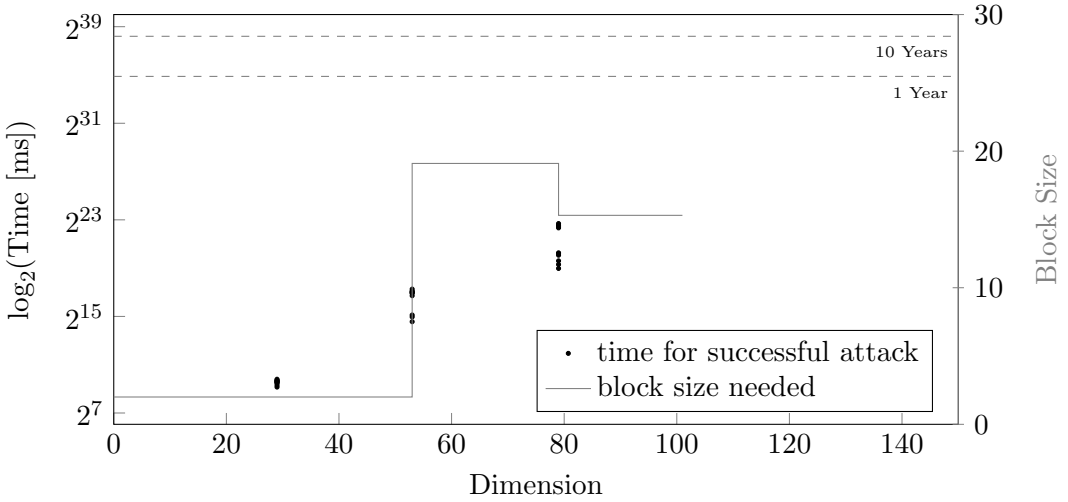

Figure 7.2: Time for Lattice Attack on the Message

7.4 Practicality

7.4.1 Space and Time Analysis

The **private key** consists of the two polynomials f and g and for performance reasons, the inverses of f modulo p and q are also stored. So the private key needs 4 vectors of length n with values less than q, which needs each $\mathcal{O}(n \log q)$ space. Since q is normally about the same magnitude as N, we simplify it to $\mathcal{O}(n \log n)$. The **public key** and the **ciphertext** are also vectors of the same structure, so they need also $\mathcal{O}(n \log n)$ space.
The generation of the **private key** includes choosing the polynomials f and g, where f has to be invertible, and the actual inversions of f modulo p and q respectively. The costly operation here is the inversion, which can be done in $\mathcal{O}(n^2)$ time. The **public key generation** and the **encryption** consist of only one polynomial multiplication, which can be done in $\mathcal{O}(n^2)$ time. **Decryption** needs two multiplications of polynomials, so it costs $\mathcal{O}(n^2)$ as well. Figure 7.3 shows these complexities in a nutshell.

Space	
Object	Size
Private Key	$\mathcal{O}(n \log n)$
Public Key	$\mathcal{O}(n \log n)$
Ciphertext	$\mathcal{O}(n \log n)$

Time	
Operation	Time
Private Key Generation	$\mathcal{O}(n^2)$
Public Key Generation	$\mathcal{O}(n^2)$
Encryption	$\mathcal{O}(n^2)$
Decryption	$\mathcal{O}(n^2)$

Figure 7.3: Space and Time Complexities of NTRU

7.4.2 Practical Tests

In this section we present the results of the experiments we made for analyzing the time consumption of NTRU. Up to 2003, we generated 100 key pairs for every dimension we measured and with every key pair we encrypted and decrypted 10 messages. In total we encrypted 80'000 messages. The dimension was always a prime number and (d_f, d_g, d_r) were chosen as $(\lfloor 0.3 \cdot D \rceil, \lfloor 0.15 \cdot D \rceil, \lfloor 0.05 \cdot D \rceil)$, for D the corresponding dimension. These settings are not recommended values, but for time analysis they should be sufficient. To check this fact, we also performed these experiments for the suggested parameters, marked as '×' in the following figures.

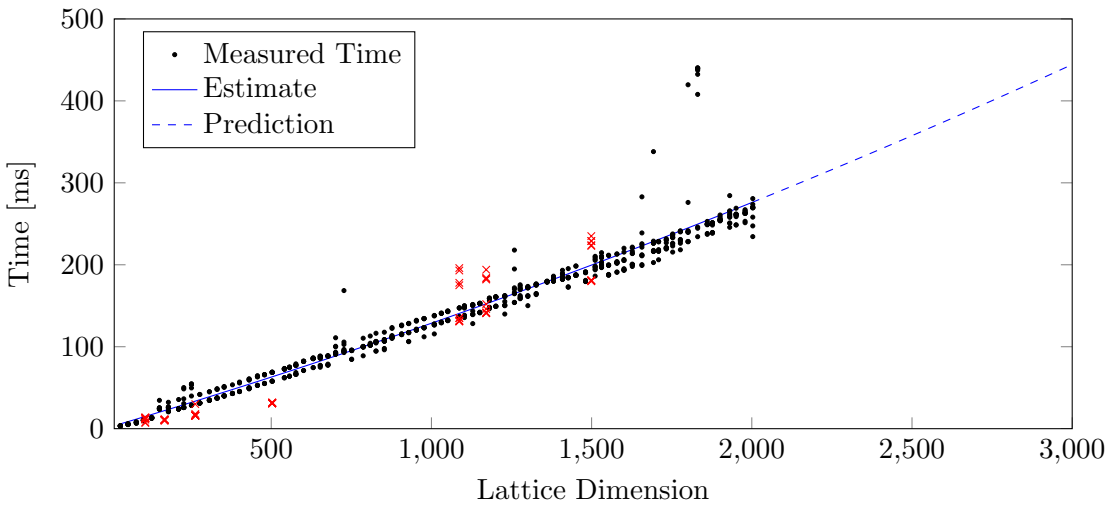

Figure 7.4: Time for Generating a Private Key in NTRU

Figure 7.4 and 7.5 show the time for generating the private and the public key, respectively. As can be seen, both keys can be generated in almost linear time. While the private key generation includes two inversions and thus take more time, the generation of the public key is almost instantly. The key generation time for the private key is similar to the time in GGH and MCS, while for the public key the difference in the time could not be more different.

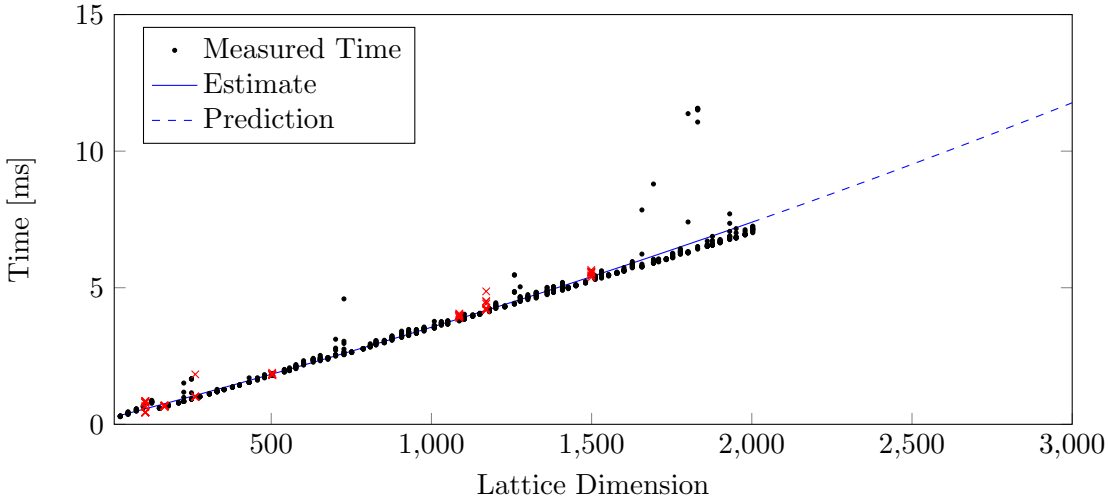

Figure 7.5: Time for Generating a Public Key in NTRU

Figure 7.6 shows the time to encrypt a message. Compared to GGH and MCS, the encryption is about one order of magnitude faster. Also the decryption is much faster and runs almost in linear time, as can be seen in Figure 7.7. Our implementation was able to decrypt 125KB/s in dimensions around 1000.

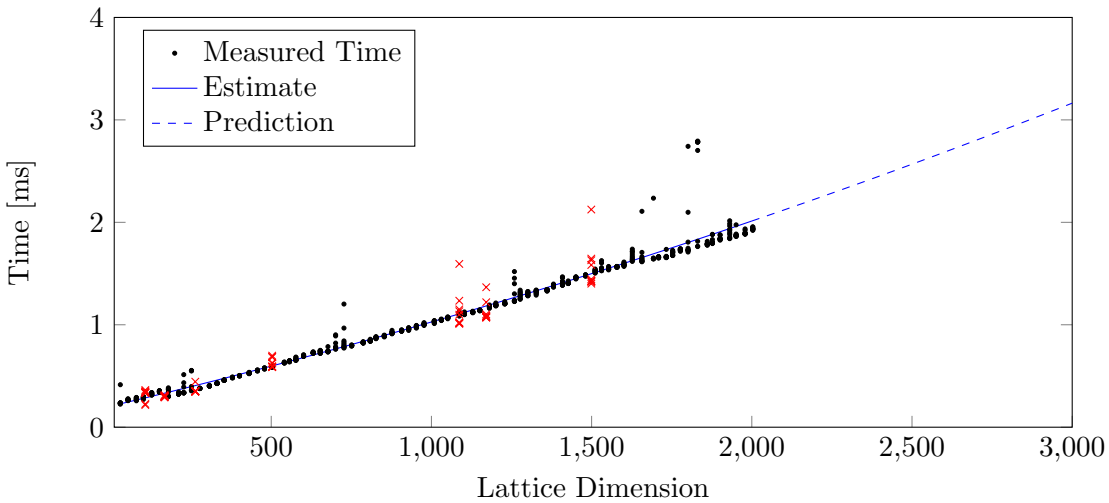

Figure 7.6: Time for Encrypting a Message in NTRU

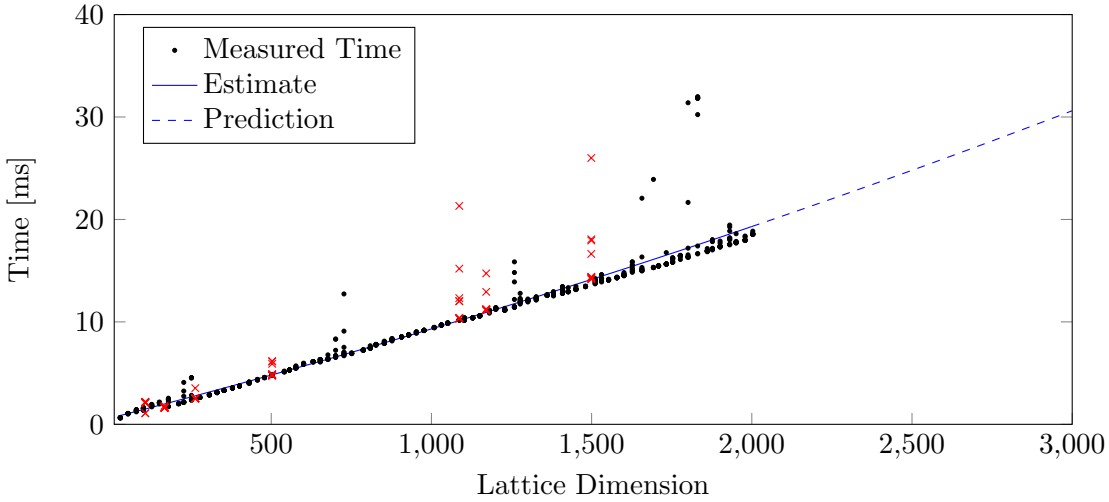

Figure 7.7: Time for Decrypting a Message in NTRU

Key and Ciphertext Size

Figures 7.8 to 7.10 show the sizes for the private key, the public key and the ciphertext, respectively. As can be observed, all the sizes are almost linear in the dimension. In dimension 1000, the sizes are 10KB, 2KB and 3KB for the private key, the public key and the ciphertext, respectively, which makes NTRU perfect for devices with a small amount of memory.

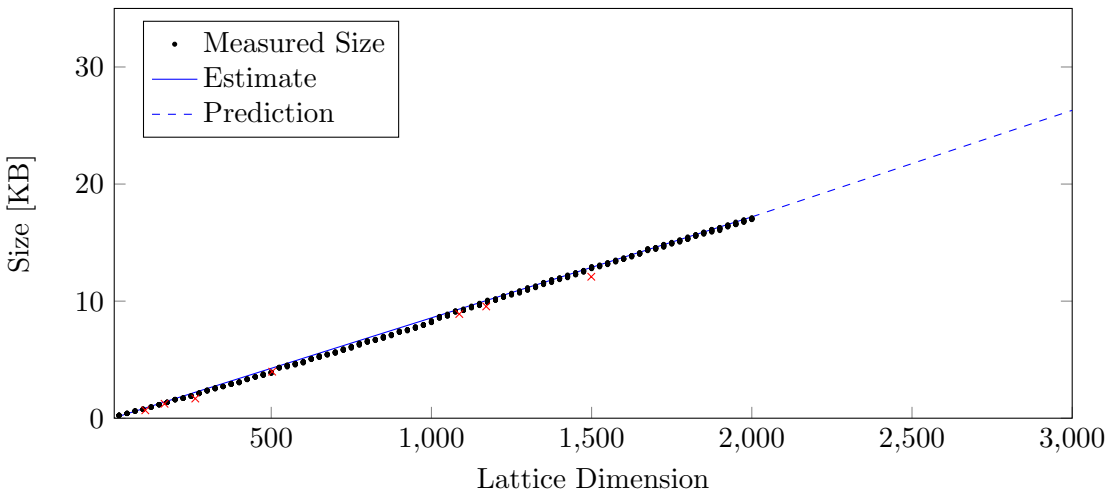

Figure 7.8: Size of a Private Key in NTRU

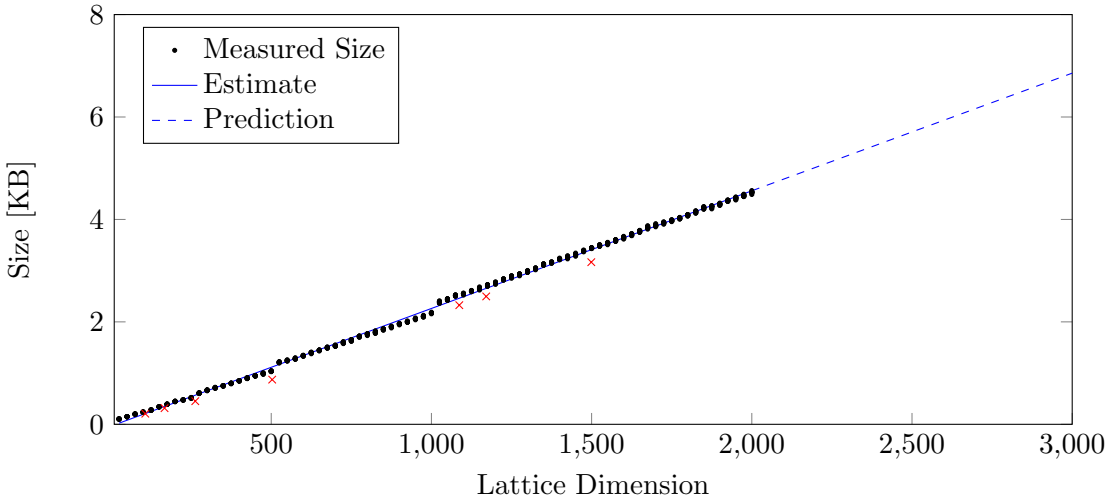

Figure 7.9: Size of a Public Key in NTRU

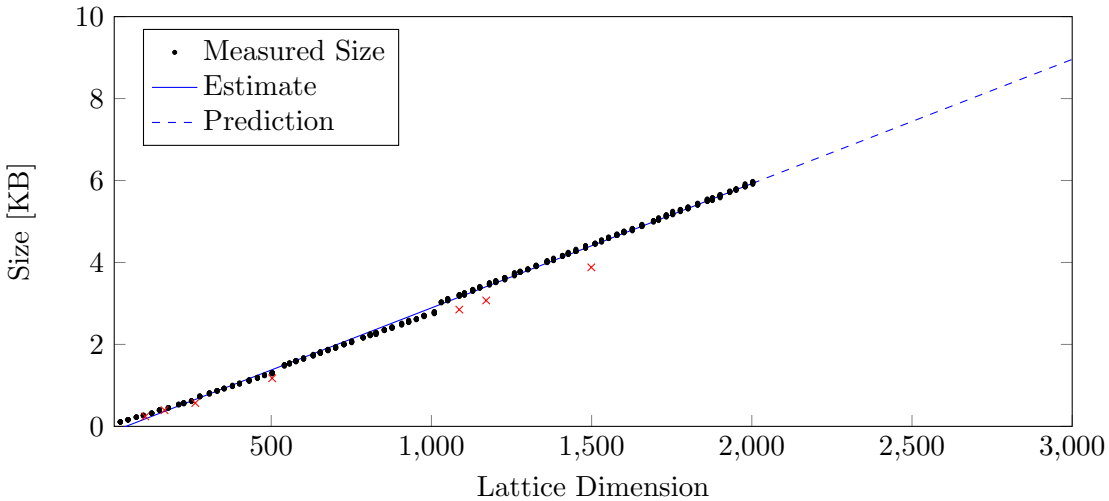

Figure 7.10: Size of a Ciphertext in NTRU

7.5 Conclusion

In this section we discussed the NTRU cryptosystem, which is in use today. The implementation of the encryption scheme is called NTRUEncrypt and is available under `https://github.com/tbuktu/ntru`. It is a good alternative to cryptosystems based on factoring or discrete logarithm as RSA and ECC. The key sizes are in the same order of magnitude and the performance is even much faster than RSA in equivalent cryptographic strength.

We have seen, that it scales the best of all covered cryptosystems, since it is only quadratic in every step (key generation, encryption, decryption) and it is almost linear in space usage ($\mathcal{O}(n \log n)$ for the key and ciphertext sizes). Furthermore, the attacks are only practical in very low dimensions. According to our experiments, dimensions over 300 should be secure against the mentioned lattice attacks.

Chapter 8

Conclusion

In this section we recapitalize the complexities, compare the times as well as the sizes from the results of the experiments as a fair as possible and discuss some future work.
Figure 8.1 shows the computational complexities of the four cryptosystem side by side. As can be observed, all the cryptosystems except of ADCS have the same complexity for generating the private key, namely $\mathcal{O}\left(n^2\right)$. This is not the case for the public key generation, which varies from $\mathcal{O}\left(n^5 \log^2 n\right)$ to $\mathcal{O}\left(n^2\right)$. Encryption and decryption have complexity $\mathcal{O}\left(n^2\right)$ for all cryptosystems, except ADCS which has a complexity of $\mathcal{O}\left(n^4\right)$ for encryption. Almost all the key sizes were improved from one cryptosystem to the next.

	ADCS	GGH	MCS	NTRU
Private Key Generation	$\mathcal{O}(n)$	$\mathcal{O}\left(n^2\right)$	$\mathcal{O}\left(n^2\right)$	$\mathcal{O}\left(n^2\right)$
Public Key Generation	$\mathcal{O}\left(n^5\right)$	$\geq \mathcal{O}\left(n^3\right)$	$\mathcal{O}\left(n^5 \log^2 n\right)$	$\mathcal{O}\left(n^2\right)$
Encryption Speed	$\mathcal{O}\left(n^4\right)$	$\mathcal{O}\left(n^2\right)$	$\mathcal{O}\left(n^2\right)$	$\mathcal{O}\left(n^2\right)$
Decryption Speed	$\mathcal{O}\left(n^2\right)$	$\mathcal{O}\left(n^2\right)$	$\mathcal{O}\left(n^2\right)$	$\mathcal{O}\left(n^2\right)$
Private Key Size	$\mathcal{O}\left(n^2\right)$	$\mathcal{O}\left(n^2 \log n\right)$	$\mathcal{O}\left(n^2 \log n\right)$	$\mathcal{O}(n \log n)$
Public Key Size	$\mathcal{O}\left(n^5\right)$	$\mathcal{O}\left(n^3 \log n\right)$	$\mathcal{O}\left(n^2 \log n\right)$	$\mathcal{O}(n \log n)$
Ciphertext Size	$\mathcal{O}\left(n^2 \log n\right)$	$\mathcal{O}\left(n^2 \log n\right)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$

Figure 8.1: Complexity Summary

It can be seen, that the NTRU cryptosystem scales the best. In our results, the key sizes as well as the ciphertext size behaves almost as they were linear in the dimension. To be fair, we have to emphasize that the ADCS was more a proof of concept for the astonishing worst-case/average-case connection the system is based on. Further it is the only cryptosystem that originally came with an security proof. For NTRU there is also a version, which makes it as secure as worst-case over ideal lattices presented in [SS11], but it has different complexities than the original system.

To compare the four cryptosystems we had to find a common measure. We decided to compare the systems in the dimensions where the best considered attack needs about 10 years to complete. The result can be seen in Figure 8.2.

	ADCS	GGH	MCS	NTRU
Dimension	48	600	800	300
Private Key Generation	10ms	100ms	200ms	50ms
Public Key Generation	2h	1min[1]	20sec[1]	5ms
Encryption Speed[2]	6min	50ms	200ms	0.5ms
Decryption Speed	6sec	1sec	1sec	5ms
Private Key Size	250KB	25KB	45KB	3KB
Public Key Size	24MB	500KB	60KB	1KB
Ciphertext Size[2]	200KB	20KB	2KB	1KB

Figure 8.2: Time and Size Comparison

Remark on the Attacks

All attacks in this theses make use of the good performance of the LLL and BKZ algorithm. Usually the strategy is to embed the desired vector in a lattice and make sure that it is a smallest vector, to find it with one of the mentioned basis reduction algorithms. So the security on the covered cryptosystems rely extremely on the performance of LLL and BKZ. Improvements of these algorithms as [Sch03, HSB+10] put the security of lattice based cryptosystems on risk.

Future Work

The time measures for the attacks on GGH and MCS were taken on instances where the error vector was bounded by $\sigma = 1$, since in most cases, this was the only case where the cryptosystems stayed error-free. To find bases, able to decrypt messages where $\sigma = 2$ or even 3 properly, is a very time-consuming task. However, we have seen, that the probability of decryption errors for $\sigma = 4$ or even 8 are very small in higher dimensions for both systems. It would be interesting to repeat the experiments for such values of σ.
Another improvement could be made by implementing a kind of LLL or BKZ algorithm, which allows the user to specify a check-function, that is called after every internal reduction step and which let the algorithm terminate if it is fulfilled. This way the reduction algorithm doesn't have to complete a whole lattice reduction, while the solution is already found. This could have a huge speedup if the check-function is implemented to respond rapidly. Indeed the check-function could run on a separate thread, so it doesn't slow down the reduction algorithm at all.

Index

Bibliography

[ABSS93] Sanjeev Arora, László Babai, Jacques Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes, and systems of linear equations. In *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*, pages 724–733. IEEE, 1993.

[AD97] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 284–293, New York, NY, USA, 1997. ACM.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 99–108. ACM, 1996.

[Ajt98] Miklós Ajtai. The shortest vector problem in L_2 is NP-hard for randomized reductions (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 10–19, New York, NY, USA, 1998. ACM.

[Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In *Automata, Languages and Programming*, pages 1–9. Springer, 1999.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in P. *Annals of mathematics*, pages 781–793, 2004.

[Bab86] László Babai. On Lovászlattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

[BC12] Jingguo Bi and Qi Cheng. Lower bounds of shortest vector lengths in random NTRU lattices. In *Theory and Applications of Models of Computation*, pages 143–155. Springer, 2012.

[Coh93] Henri Cohen. *A course in computational algebraic number theory*, volume 138. Springer, 1993.

[CS97] Don Coppersmith and Adi Shamir. Lattice attacks on NTRU. In *Proceedings of the 16th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'97, pages 52–61, Berlin, Heidelberg, 1997. Springer-Verlag.

[DB14] Nikesh S Dattani and Nathaniel Bryans. Quantum factorization of 56153 with only 4 qubits. *CoRR*, 2014.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[DK12] Daniel Dadush and Gábor Kun. Lattice sparsification and the approximate closest vector problem. *CoRR*, abs/1212.6781, 2012.

[GGH97a] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Eliminating decryption errors in the Ajtai-Dwork cryptosystem. In *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 105–111. Springer, 1997.

[GGH97b] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In *Advances in Cryptology-CRYPTO'97*, pages 112–131. Springer, 1997.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.

[GMSS99] Oded Goldreich, Daniele Micciancio, Shmuel Safra, and J-P Seifert. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. *Information Processing Letters*, 71(2):55–61, 1999.

[GN07] Nicolas Gama and Phong Q. Nguyen. New chosen-ciphertext attacks on NTRU. In *Public Key Cryptography–PKC 2007*, pages 89–106. Springer, 2007.

[GPL07] GNU general public license, version 3. `http://www.gnu.org/licenses/gpl.html`, June 2007. Last retrieved 2012-05-10.

[HGNP+03] Nick Howgrave-Graham, Phong Q. Nguyen, David Pointcheval, John Proos, Joseph H. Silverman, Ari Singer, and William Whyte. The impact of decryption failures on the security of NTRU encryption. In *Advances in Cryptology-CRYPTO 2003*, pages 226–246. Springer, 2003.

[HLPS99] Jeff Hoffstein, Daniel Lieman, Jill Pipher, and Joseph H. Silverman. NTRU: A public key cryptosystem. *URL: http://grouper. ieee. org/groups/1363/lattPK/submissions/ntru. pdf*, 1999.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *Proceedings of the Third International Symposium on Algorithmic Number Theory*, ANTS-III, pages 267–288, London, UK, UK, 1998. Springer-Verlag.

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Algorithms for the shortest and closest lattice vector problems. In *Coding and Cryptology*, pages 159–190. Springer, 2011.

[HSB+10] Jens Hermans, Michael Schneider, Johannes Buchmann, Frederik Vercauteren, and Bart Preneel. Parallel shortest lattice vector enumeration on graphics cards. In *Progress in Cryptology–AFRICACRYPT 2010*, pages 52–68. Springer, 2010.

[Kan87] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, August 1987.

[LH08] Moon Sung Lee and Sang Geun Hahn. Analysis of the GGH cryptosystem. In *First International Conference on Symbolic Computation and Cryptography*, page 154, 2008.

[LLL82] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[Lud02] Christoph Ludwig. *The Security and Efficiency of Micciancio's Cryptosystem.* Inst. für Theoretische Informatik, 2002.

[Lyu08] Vadim Lyubashevsky. The n^c-unique shortest vector problem is hard. Cryptology ePrint Archive, Report 2008/504, 2008. `http://eprint.iacr.org/`.

[MC06] K.L.P. Mishra and N. CHANDRASEKARAN. *Theory of Computer Science: Automata, Languages and Computation.* PHI Learning, 2006.

[McE78] Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN progress report*, 42(44):114–116, 1978.

[Mic01a] Daniele Micciancio. The hardness of the closest vector problem with preprocessing. *IEEE Transactions on Information Theory*, 47:2001, 2001.

[Mic01b] Daniele Micciancio. Improving lattice based cryptosystems using the Hermite normal form. In *Cryptography and Lattices*, pages 126–145. Springer, 2001.

[Mol02] Richard A Mollin. *RSA and public-key cryptography.* CRC Press, 2002.

[MV10] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 1468–1480, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[MW01] Daniele Micciancio and Bogdan Warinschi. A linear space algorithm for computing the Hermite normal form. In *Proceedings of the 2001 international symposium on Symbolic and algebraic computation*, pages 231–236. ACM, 2001.

[Ngu99] Phong Nguyen. Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from crypto97. In *Advances in CryptologyCRYPTO99*, pages 288–304. Springer, 1999.

[NS98] Phong Q. Nguyen and Jacques Stern. Cryptanalysis of the Ajtai-Dwork cryptosystem. In *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 223–242. Springer, 1998.

[NS99] Phong Q. Nguyen and Jacques Stern. A converse to the Ajtai-Dwork security proof and its cryptographic implications. *Available from ECCC as TR98-010*, 1999.

[NS01] Phong Q. Nguyen and Jacques Stern. The two faces of lattices in cryptology. In *Cryptography and lattices*, pages 146–180. Springer, 2001.

[NS05] Phong Q. Nguyen and Damien Stehlé. Floating-point LLL revisited. In *Advances in cryptology–Eurocrypt 2005*, pages 215–233. Springer, 2005.

[NS09] Phong Q. Nguyen and Damien Stehlé. An LLL algorithm with quadratic complexity. *SIAM Journal on Computing*, 39(3):874–903, 2009.

[NV09] Phong Q. Nguyen and Brigitte Valle. *The LLL Algorithm: Survey and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[S+14] W. A. Stein et al. *Sage Mathematics Software (Version 6.1.1)*. The Sage Development Team, 2014. `http://www.sagemath.org`.

[Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical computer science*, 53(2):201–224, 1987.

[Sch03] Claus-Peter Schnorr. Lattice reduction by random sampling and birthday methods. In *STACS*, volume 2607, pages 145–156. Springer, 2003.

[SE94] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66(1-3):181–199, 1994.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM journal on computing*, 26(5):1484–1509, 1997.

[Sho14] Victor Shoup. Number theory C++ library (ntl) version 6.1, 2014. `http://www.shoup.net/`.

[Sil99] Joseph H. Silverman. Almost inverses and fast NTRU key creation. *NTRU Cryptosystems,(Technical Note# 014)*, 1999.

[SS11] Damien Stehlé and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In *Advances in Cryptology–EUROCRYPT 2011*, pages 27–47. Springer, 2011.

[Ste10] Damien Stehlé. Floating-point LLL: theoretical and practical aspects. In *The LLL Algorithm*, pages 179–213. Springer, 2010.

[vEB81] P. van Emde-Boas. Another NP-complete partition problem and the complexity of computing short vectors in a lattice. *http://staff.science.uva.nl/ peter/vectors/mi8104c.html*, 1981.

[XZL+12] Nanyang Xu, Jing Zhu, Dawei Lu, Xianyi Zhou, Xinhua Peng, and Jiangfeng Du. Quantum factorization of 143 on a dipolar-coupling nuclear magnetic resonance system. *Phys. Rev. Lett.*, 108:130501, Mar 2012.