GGH Cryptosystem and Lattice Reduction Algorithms

GGH CRYPTOSYSTEM AND LATTICE REDUCTION ALGORITHMS

BY

ZHAOFEI TIAN, B.Sc.

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING & SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

Master of Science (2011) McMaster University

(Computing & Software) Hamilton, Ontario, Canada

TITLE: GGH Cryptosystem and Lattice Reduction Algorithms

AUTHOR: Zhaofei Tian

B.Sc., (Applied Mathematics) Peking University, Beijing,

P. R. China

SUPERVISOR: Dr. Sanzheng Qiao

NUMBER OF PAGES: ix, 100

To my wife, my son and my parents

Abstract

The capability of encrypting top secret information remains as a major research problem in the GGH cryptosystem, which depends on various attacking methods. The early approaches to attacking the GGH cryptosystem mainly relied on special properties of the lattice generated by the vectors of the private key. Consequently, those attacks are not appropriate for general cases.

This thesis presents a GGH attacking method for general cases. A lattice basis reduction algorithm is applied to the public key to get a better basis, which is used to decrypt the ciphertext. In the proposed approach, we concentrate on three lattice reduction algorithms: the LLL algorithm, the approximate optimally-reduced algorithm, and the optimally-reduced algorithm. We have implemented a package in MATLAB for the GGH cryptosystem and the three algorithms. We experimented with two groups of experiments and obtained promising results for lattices of low dimensions.

Acknowledgements

I extend my sincere gratitude and appreciation to many people who made this Master's thesis possible.

First and foremost, it is an honor for me to offer the sincerest thank to my supervisor, Dr. Sanzheng Qiao, for kindly providing guidance with his patience and knowledge throughout the development of my study and preparation of this thesis. One simply could not wish for a better or friendlier supervisor.

I would like to show my gratitude to Dr. Alan Wassyng, the Committee Chairman, for his careful review and valuable guidance and suggestions.

It is a pleasure to thank Dr. Michael Soltys. He is not only acting as a member of my Examination Committee, but also the teacher of two courses of mine. It is the spirit of his course "Cryptography" that brings on my research of this topic.

I'm also grateful to my friend Junzheng Yang, for his greatest help that supported me to pass the toughest time when I first arrived at Canada.

Finally, I am indebted to my family and also all whose direct and indirect support helped and encouraged me completing my thesis in time.

Contents

List of Figures

Chapter 1

Introduction

For a long history, we always tend to seek a safe way to exchange messages between each other, and prevent the others from gaining unsolicited access to confidential information. Many mechanisms have been invented for this purpose in different time period. For example, people in Egypt's Old Kingdom carved non-standard hieroglyphs into stones to keep messages secure in B.C. 2500; the Kama Sutra spread as a technique by which lovers can communicate without being discovered in India in A.D. 500.

Nowadays, cryptology has played an important role in both political and military applications through the 20th century. Mathematical cryptography leapt ahead (also secretly) after World War I, when cryptosystems were widely used between countries and armies. One of the most famous cryptosystem that influenced the world is the the Germany's Enigma[35] during the World War II, which was broken by the scientists of Poland, Great Britain and the United States at Britain's Bletchley Park. The World War II is said to be shortened by at least two years[27, 11] for the break of the Enigma Machine, and it led to the development of the first digital computer, whose mathematical model is the *Turing Machine*. The Turing machine model has proven to be of priceless value for

the development of the science of data processing, artificial intelligence, mathematics, and so on[47].

A mechanism that exchanges information secretly is called a *Cryptosystem*[24]. Bob, the *sender, encrypt*s a message (called the *plaintext*) into a secret message (called the *ciphertext*) by an encryption algorithm with a *secret key.* Alice, the *recipient,* receives the ciphertext and *decrypt*s it to discover the plaintext by a decryption algorithm using a key. Before 1975, all cryptosystems are the *Symmetric Cryptography,* which required the sender and the receiver to agree on the same secret key. The Enigma Machine, for example, is a symmetric cryptography.

In 1977, the RSA Public Key cryptosystem[43, 5], named after inventors Rivest, R. L., Shamir, A. and Adleman, L., was introduced to public, which was the first time that the concept of *Public Key Cryptography* circulated in the research community. After the RSA cryptosystem, many Public Key Cryptography were proposed, for example, the ElGamal Cryptosystem[13], the NTRU Cryptosystem[24], the GGH Cryptosystem[16] and so on.

It is in this spirit that the research activity and this thesis is conducted. This thesis uses lattice theory to study the GGH Public Key Cryptosystem and uses lattice reduction algorithms to attack it. The lattice reduction algorithms[14] are designed for producing nearly orthogonal bases for a given lattice, which can be applied to many applications. We showed in experiments that it was also a feasible method to attack the GGH cryptosystem in general cases.

1.1 Thesis Outline

The introductory chapter was devoted to give a general understanding of cryptology. For the public-key cryptographies, we focus on the GGH scheme in this thesis. The tool

we used to cryptanalyze the GGH comes from theory of integer lattices. The research also implies that the GGH cryptosystem is not the only one that can be attacked using lattices technology, RSA, for example, can also be partially decrypted in terms of lattice operations[22, 8, 9].

Chapter 2 will present the necessary definitions of lattices and bases. A definition of the special integer matrices named *Unimodular* matrix and the relations between two arbitrary bases for a lattice will also be shown in this chapter. In the end of chapter 2, we will briefly introduce the two problems related with lattice theory.

Chapter 3 will provide introductions to the GGH cryptosystem, the *Hadamard Ratio* and the *Fundamental Domain*. An algorithm called the *Babai's* algorithm will be introduced along with the GGH cryptosystem, which is widely invoked in many closest vector solving algorithms. In the end of this chapter, we will present our method to attack the GGH cryptosystem. This method will first produce a new reduced basis by lattice reduction algorithms from the Public Key, then decrypt the ciphertext with this new basis.

In chapter 4, we will give a *Basic Enumeration* algorithm which finds a shortest vector in a given lattice. The algorithm uses the *Gram-Schmidt Orthogonalization* and the lattice projection technique to estimate a range that includes a shortest vector and other candidates. Then the shortest vector will be discovered by comparing all vectors within the range.

Chapter 5 is the most important part of this thesis, which will dedicated to three lattice reduction algorithms. The LLL algorithm is a widely used polynomial time algorithm in lattice theory. In this thesis, it will act as a pre-processor to improve the performance of the other two algorithms. The Approximate Optimally-Reduced algorithm and the Optimally-Reduced algorithm are the two major algorithms we focus on.

Their complexities are exponential, but the bases they produced are much better than the LLL-Reduced bases, especially the bases generated by the Optimally-Reduced algorithm, which is even better than the Private Key.

Chapter 6 will give the our experimental results. We will use the algorithms introduced in chapter 5 to produce reduced bases, and compare the Hadamard Ratios between the Private Key and the produced bases in the first pat of this chapter. In the second part of this chapter, we will count the miss ratios of decrypting the ciphertexts with produced bases and the Private Key.

Finally, the chapter 7 will conclude the research done in this thesis, and the possible future works will be proposed as well.

We attached the experimental results in details in the Appendix.

1.2 Thesis Contributions

This thesis presents an approach to attacking the GGH cryptosystem. Differ from other GGH attacking methods[38, 19], which depend on special properties of the certain lattices, the method proposed in this thesis works in general cases. This approach takes advantage of lattice theory, number theory and lattice reduction algorithms[14, 42, 48]. The contributions of this thesis can be categorized as follows:

1. Implemented the Approximate Optimally-Reduced algorithm and the Optimally-Reduced algorithm;

2. Implemented a matrix version LLL algorithm;

3. Identified the dimension of GGH cryptosystems that can be attacked by the lattice reduction algorithms;

4. Designed and implemented an iterative procedure to solving sphere decoding problem in the Optimally-Reduced algorithm;

As a result of this research, we implemented a GGH cryptosysytem package and a lattice reduction algorithm package which includes the matrix-version LLL-Reduced function, the approximate Optimally-Reduced function and the Optimally-Reduced function. They can be imported to MATLAB R2010b and hence be freely invoked by other functions.

Chapter 2

Lattices

In the recent years, constructing lattices to solve problems are more and more involved in many research fields other than number theory. The usage of lattices contributes significant progresses both in theoretical and in practical applications, such as wireless communication, integer programming, cryptology, and so on. A lattice is a set of discrete points in a vector space, which can be represented by integer linear combinations of a set of vectors. There are two important problems[18] related with lattice: the *Shortest Vector Problem* (SVP) and the *Closest Vector Problem* (CVP). We will introduce them later in the second part of this chapter.

This chapter introduces several important concepts related to lattice theory from the point of view of geometry of numbers. Of primary importance in this work is the definitions, notions and problems of a lattice that will be shown. In this chapter we state only the conclusions about lattices that are necessary for the rest of this thesis, and refer the readers to[18, 24] for a comprehensive introduction.

2.1 Lattices and Bases

Column-version representation for matrices is selected in the definitions and operations in this thesis. For example, a matrix $B \in \mathbb{R}^{m \times n}$ will be partitioned into $[\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n]$.

Definition 2.1.1 (Lattice). Let $B = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n] \in \mathbb{R}^{m \times n}$ be a matrix of full-columned rank. The *Lattice L generated* by the columns of $B_{m \times n}$ is the infinite set of linear combinations of vectors $\{ \mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n \}$ with coefficients in $\mathbb{Z}$ (the set of all integers), in other words:

$$L = \{ B\mathbf{z} \mid \mathbf{z} \in \mathbb{Z}^n \}.$$

The set of vectors $\{ \mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n \}$ is called a *basis* for the lattice L.

Given a lattice L and its generator matrix $B = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n] \in \mathbb{R}^{m \times n}$, when there is no confusion, we simply say "*L is generated by B*", which we mean "*L is generated by the columns of the generator matrix B*", represented with $L(B)$ or $L(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n)$.

We can see from the definition that a lattice L is a subset of $\mathbb{R}^m$, which is closed in addition operations and integer scalar multiplication operations. A *basis* for L is any set of n independent vectors that can *generate* L. For a given lattice L, any such two bases are constructed with the same number of vectors. This number is defined as the *dimension* of the lattice L, represented by $n = \dim(L)$.

Figure-(2.1) illustrates a lattice L with its two generator matrices $A = [\mathbf{a}_1, \mathbf{a}_2]$ and $B = [\mathbf{b}_1, \mathbf{b}_2]$. All discrete points in L can be interpreted as integer linear combinations of the columns of matrix A or matrix B.

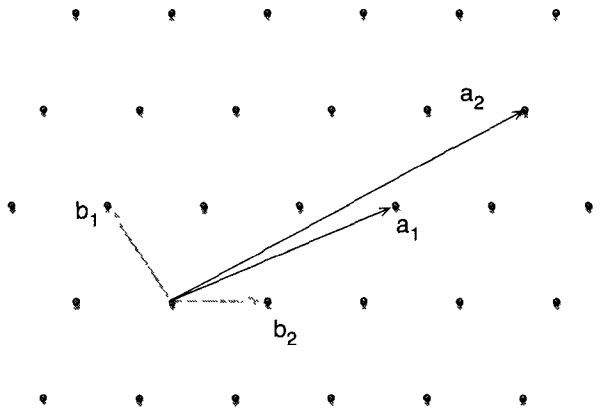

Figure 2.1: A lattice L and its two bases

Definition 2.1.2 (Unimodular). A *nonsingular* integer matrix M is called *unimodular* if and only if the determinant of M satisfies:

$$\det(M) = \pm 1.$$

Proposition 2.1.3. *Any two bases for a lattice are related with an unimodular matrix.*

Proof. Suppose B and B' are two arbitrary generator matrices for a given lattice L. Since B generates L, B also generates all columns of B'. Then there must exist an integer matrix M of coefficients, such that $B' = BM$. On the other hand, we can find another integer matrix M' such that $B = B'M'$, because B' generates the lattice L as well.

Therefore,

$$B = B' \times M' = (B \times M) \times M'.$$

In addition, B and B' are all n dimensional full-column rank matrices. According to the Cramer's Rule, M is uniquely determined by B and B', hence M is nonsingular and invertible. Since $B' = B \times M$, we know $B = B' \times M^{-1}$. Therefore $M' = M^{-1}$.

Since the matrices M and M^{-1} are integer matrices, their determinant are integers as well. Therefore, the only choice for $\det(M)$ and $\det(M^{-1})$ is $\det(M) = \det(M^{-1}) = \pm 1$. □

Example 2.1.4 (Unimodular Matrix M [24]). Let L be a 3-dimensional lattice generated by the generator matrix

$$A = \begin{bmatrix} -97 & -36 & -184 \\ 19 & 30 & -64 \\ 19 & 86 & 78 \end{bmatrix}.$$

Then L is identical with the lattice $\bar{L}$ generated by the generator matrix

$$B = \begin{bmatrix} -4179163 & -3184353 & -5277320 \\ -1882253 & -1434201 & -2376852 \\ 583183 & -2376852 & 736426 \end{bmatrix},$$

since there exists an unimodular matrix

$$M = \begin{bmatrix} 4327 & 3297 & 5464 \\ -15447 & -11770 & -19506 \\ 23454 & 17871 & 29617 \end{bmatrix},$$

such that $B = A \times M$. We can check that the two lattices generated by the columns of A and B are equivalent by calculating $\det(M) = -1$.

This fact gives us a very nice property in lattice theory that for a given lattice L and its any two generator matrices A and B, there always exists an unimodular matrix M such that $A = BM$. Meanwhile, given a generator matrix B for the lattice L, a new generator matrix can be constructed by multiplying B with any unimodular matrix. Therefore, the absolute value of the determinants of all generator matrices is a constant. Consequently,

we define the *determinant* of a lattice L as this absolute value, which is

$$\det(L) = |\det(B)| \tag{2.1.1}$$

where B is a generator matrix for L.

2.2 Lattice Related Problems

Recall the two kinds of problems related with lattices we mentioned at the beginning of this chapter. According to the hardness problems related with cryptography, we are more interested in algorithms for finding a lattice vector closest to a given arbitrary target vector, when we attack the GGH[16] or other lattice based cryptosystems. Solving SVP always accompanies with the algorithms of solving CVP. Therefore we will give precise definitions for those problems[24, Chapter 6].

1. **The Shortest Vector Problem** (SVP)

   Find a shortest nonzero vector in a given lattice L, i.e., search for a vector $\mathbf{v} \in L$ that minimizes the Euclidean norm $\|\mathbf{v}\|_2$.

2. **The Closest Vector Problem** (CVP)

   Given a lattice L and a vector $\mathbf{w} \in \mathbb{R}^m$, normally $\mathbf{w} \notin L$, find a vector $\mathbf{v} \in L$ that is closest to $\mathbf{w}$ among all points of L, i.e., find a vector $\mathbf{v} \in L$ that minimizes the Euclidean norm $\|\mathbf{w} - \mathbf{v}\|_2$.

The complexity of solving CVP has been proved to be $\mathbb{NP}$-hard[1, 18]. Solving SVP is $\mathbb{NP}$-hard under certain situations as well[34]. Therefore, they are commonly regarded

as the problems of same difficulty. In practical terms, solving CVP is considered to be a little bit harder than solving SVP under the same dimension.

Kannan published an exact SVP solving algorithm[25] with super exponential complexity $n^{\mathcal{O}(n)}$ in 1983. The algorithm finds a shortest vector by enumerating all possible points within certain range. So far the best known algorithm for solving SVP in high dimensions comes from Miklós Ajtai, Ravi Kumar and D. Sivakumar[2], which is a randomized algorithm that takes the complexity $P(n)2^{\mathcal{O}(n)}$ to find a shortest vector, where $P(n)$ is polynomial of degree n. Ajtai's algorithm constructs a sequence of sieves that hierarchically filter points of a lattice by their Euclidean Lengths. With a high probability, a shortest vector is likely to be discovered. P. Q. Nguyen and T. Vidick[37] proved that the Ajtai's algorithm was not only theoretically fast but also practically implementable. For most lattice reduction algorithms, whether solving SVP or solving CVP, are all trying to build (or likely try to build) a Minkowski-Reduced bases (whose definition will be introduced in the chapter "Lattice Reduction Algorithms"). Bettina Helfrich[23] gave a relatively fast algorithm that could construct Minkowski-reduced bases theoretically in high dimension. The algorithm runs in polynomial-time operations for a fixed dimension lattice taking advantages of the LLL algorithm[29] and the Kannan's algorithm[25].

There are also two minor lattice related problems : the Approximate Shortest Vector Problem (apprSVP) and the Approximate Closest Vector Problem (apprCVP). They are required to find a nonzero vector whose length is no more than a given factor $\psi(n)$ times longer than an exact shortest (closest) nonzero vector. For apprSVP, C. P. Schnorr[44] showed a polynomial time algorithm to obtain an approximate shortest vector with a factor $(6k^2)^{n/k}$ by a block generation method based on the LLL algorithm, where k

is a fixed integer divider of n. The algorithm uses $\mathcal{O}(n^2(\sqrt{k}^{\,k+o(k)}))$ arithmetic operations. Nicolas Gama[15] improves Schnorr's algorithm by factor of log2. Perhaps the best known algorithm for solving apprSVP is the famous LLL algorithm[29], which takes polynomial time to achieve an approximate shortest vector with a factor up to $\mathcal{O}(2^{\frac{n-1}{2}})$, the vector it discovered is not the best one though. In practice, the run time of LLL algorithm can be further decreased by a small factor using the *Deep Insertion* method[40, Page 150].

Chapter 3

GGH and Lattice Based Cryptography

The recorded history of encoding and decoding a message can be up to four thousands years, in non-standard hieroglyphs carved into monuments from Egypt's Old Kingdom. In the recent days, cryptosystems play a key roles in military applications and national securities, especially during World War I and World War II. For example, the most famous *Enigma Machine* of German[35] is a classical cryptosystem in World War II. The widely used modern cryptosystems are all based on hardness of a variety of mathematical problems. For example, ElGamal[13] is based on discrete logarithm problem, and RSA[5] takes advantage of factorization of large numbers. Since the complexity of solving CVP has been proved to be $\mathbb{NP}$-hard[1, 18] on average cases, the GGH Cryptosystem[16] is designed to be a novel encryption and decryption mechanism based on hardness of solving CVP.

It always motivates us to introduce cryptosystems based on various hard mathematical problems, hence the top secrete information encoded by more than one algorithm remains secure even part of them is broken. Lattice based cryptosystems are born with many advantages[24], for example, the encoding and decoding progresses are much

faster than the cryptosystems based on discrete logarithm and the large number factorization. Besides, the property of matrix operations makes the GGH cryptosystem easier to be implemented in hardware and software of modern computers than RSA and ElGamal.

After the definitions of lattices and bases in the last chapter, this chapter will give the most commonly used measurement for the bases of a lattice, the *Hadamard Ratio*[24]. Then we will introduce the Fundamental Domain and the Babai's algorithm. Those are widely involved in lattice based cryptosystems. After that, the GGH cryptosystem will then be described that as the major problem of this thesis, which is a direct application in terms of the hardness of solving CVP.

3.1 Basic Definitions

GGH is not the only lattice based cryptosystem that underlies the hardness of solving CVP or SVP. The Ajtai-Dwork cryptosystem and the NTRU cyrptosystem introduced by Hoffstein, Pipher and Silverman are all based on the difficulty of solving CVP[24]. Due to similarities of lattice operations for those cryptosystems, we will introduce a few basis definitions which are necessary for constructing them.

3.1.1 Hadamard Ratio

Since a lattice owns more than one basis, we always tend to find a "good" one, especially the orthogonal ones. However, for most lattices there exists no orthogonal basis. Therefore, a standard principle must be clarified to judge a given basis is a good one, or needs to be improved further. Intuitively, a good basis should help us handle the lattice related

problems, and hence could produce high quality results during lattice operations. Of course the term "high quality" is ambiguous.

The *Gram-Schmidt* algorithm in vector space sparks us on how to measure the quality of a given basis. The vectors in a good basis for a lattice should be as short as possible. Hadamard[24] introduced a quantitative formula for the lattice bases measurement.

Definition 3.1.1 (Hadamard Ratio). Given a basis $B = \{ \mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n \}$ and the n dimensional lattice L generated by B, the Hadamard Ratio of the basis B is defined by the quantity:

$$\mathcal{H}(B) = \left(\frac{\det(L)}{\|\mathbf{b}_1\|_2 \cdot \|\mathbf{b}_2\|_2 \cdots \|\mathbf{b}_n\|_2}\right)^{1/n}. \tag{3.1.1}$$

where $\|\cdot\|_2$ represents the Euclidean Norm of $\cdot$.

The reciprocal of the Hadamard Ratio is also known as *orthogonality defect.* The *Hermite-Hadamard Inequality*[4] shows a relation between the determinant of a lattice L, the determinant of its arbitrary basis B, and the lengths of the vectors in B,

$$\det(L) = |\det(B)| \leq \|\mathbf{b}_1\|_2 \cdot \|\mathbf{b}_2\|_2 \cdots \|\mathbf{b}_n\|_2.$$

Therefore, the range of Hadamard Ratio can be derived from the above inequation that:

$$0 < \mathcal{H}(B) \leq 1 \tag{3.1.2}$$

We will use the **Equation**-(3.1.1) as one measurement to judge the qualities of bases produced by lattice reduction algorithms in the thesis. The more orthogonal a basis B is, the closer to 1 its Hadamard Ratio $\mathcal{H}(B)$ is.

3.1.2 Fundamental Domain

By connecting the points at the tip of elements in a basis for a lattice, a parallelepiped area could be created, which is called *Fundamental Domain*[24].

Definition 3.1.2 (Fundamental Domain). Let L be a lattice of dimension n and let B be a basis for L. The *fundamental domain* (or equivalently the *fundamental parallelepiped*) for L corresponding to this basis B is the set

$$\mathscr{F}(B) = \{\, B\mathbf{r} \mid 0 \le r_i < 1, \quad (1 \le i \le n) \,\}.$$

We define the *volume* of $\mathscr{F}(B)$ as the volume of the corresponding parallelepiped in $\mathbb{R}^n$, which is constructed by connecting all the points of vectors in B together. It turns out that the *volume* of the fundamental domains defined above for the lattice L is an extremely important concept. Just like the determinant of the lattice, the volume of fundamental domains for a lattice is an invariant which is independent of the choice of bases for the lattice.

Corollary 3.1.3. *Let $L \subset \mathbb{R}^n$ be a lattice of dimension n. Then every fundamental domain of L has the same volume. Hence* $\mathrm{Vol}(\mathscr{F})$ *is an invariant of the given lattice L, independent of the particular fundamental domain used to compute it, that is:*

$$\mathrm{Vol}(\mathscr{F}) = \det(L).$$

The whole proof of the above Corollary can be found in [24, Chapter 6.4]. Recall the property of determinants of bases for a lattice L, we can arrive at the two invariants of the lattice theory that

$$\det(L) = \mathrm{Vol}(\mathscr{F}) = |\det(B)|,$$

for an arbitrary basis B and its fundamental domain $\mathscr{F}(B)$.

The **Figure**-(3.2) illustrates two fundamental domains. The two bases $B = \{ \mathbf{b}_1, \mathbf{b}_2 \}$ and $A = \{ \mathbf{a}_1, \mathbf{a}_2 \}$ form two independent fundamental domains for the 2-dimensional lattice. They are distinguished by different shadows. Although the two fundamental domains have different shapes, the areas they covered are the same.

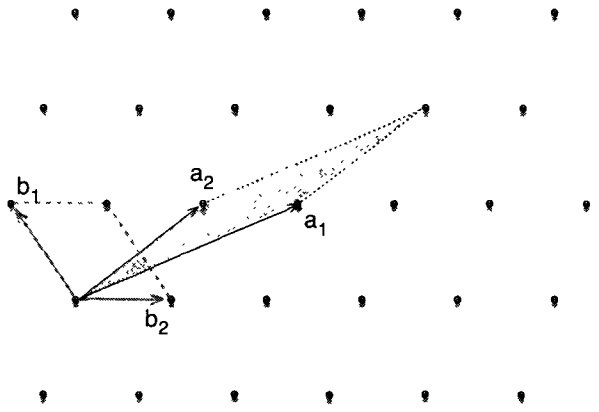

Figure 3.2: A lattice with its two fundamental domain $\mathscr{F}$s

The following proposition builds up a feasible theoretical foundation for lattice based cryptosystems, GGH, for example. The proof of this proposition is given in [24], which uses a counterexample and it is not only very long but also hard to follow. We will introduce a different but much easier and simpler proof by taking advantage of *Cramer's Rule* in a vector space.

Proposition 3.1.4. *Let $L \subset \mathbb{R}^n$ be a lattice of dimension n and $\mathscr{F}$ be a fundamental domain for L with respected to a basis B. Then every vector $\mathbf{w} \in \mathbb{R}^n$ can be written in the form:*

$$\mathbf{w} = \mathbf{t} + \mathbf{v}, \quad \text{for a unique } \mathbf{t} \in \mathscr{F} \text{ and a unique } \mathbf{v} \in L. \tag{3.1.3}$$

Equivalently, the union of the *translated fundamental domains* with respected to a vector $\mathbf{v}$

$$\mathscr{F} + \mathbf{v} = \{ \mathbf{t} + \mathbf{v} \mid \mathbf{t} \in \mathscr{F}, \mathbf{v} \in L \} \tag{3.1.4}$$

exactly covers $\mathbb{R}^n$ as $\mathbf{v}$ ranges over all vectors in the lattice L.

Proof. Since $B_{n\times n}$ is a generator matrix for the lattice L that gives the fundamental domain $\mathscr{F}$ and $\dim(L) = n$, we know that vectors $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ of B are linearly independent in $\mathbb{R}^n$. Hence the columns of B span the vector space $\mathbb{R}^n$.

Moreover, by the classical *Cramer's Rule* in linear algebra, for any vector $\mathbf{w} \in \mathbb{R}^n$, there exists an unique real coefficient vector $\mathbf{x} \in \mathbb{R}^n$, such that $\mathbf{w} = B\mathbf{x}$.

Set:

$$\begin{cases} z_i = \lfloor x_i \rfloor \\ r_i = x_i - z_i \end{cases}$$

for all i $(1 \le i \le n)$, where $\lfloor x \rfloor$ represents the biggest integer z, such that $z \le x$.

Let $\mathbf{v} = B\mathbf{z}$ and $\mathbf{t} = B\mathbf{r}$, we know that for each i, the inequation $0 \le r_i < 1$ holds. Thus the vector $B\mathbf{r}$ is in the fundamental domain parallelepiped by the lattice generator matrix B. Therefore, we have uniquely constructed the two vectors $\mathbf{v} \in L$ and $\mathbf{t} \in \mathscr{F}$ successfully, such that $\mathbf{w} = \mathbf{t} + \mathbf{v}$. □

3.1.3 Babai's Algorithm

Given an n dimensional lattice L and its generator matrix B, the **Proposition**-(3.1.4) tells us that for any vector $\mathbf{w} \in \mathbb{R}^n$ $(\mathbf{w} \notin L)$, an unique decomposition $\mathbf{w} = \mathbf{t} + \mathbf{v}$ can always be found, such that $\mathbf{v} \in L$ and $\mathbf{t}$ locates in the fundamental domain corresponding to the basis B. This proposition gives us an idea to solve CVP, that is, to identify the translated

fundamental domain-(3.1.4) with respected to $\mathbf{v} \in L$, in which the target vector $\mathbf{w}$ locates.

A commonly used algorithm encouraged by the **Proposition**-(3.1.4) is the *Babai's* algorithm[24], which is an approximation algorithm for solving the CVP. This algorithm, also known as the *Nearest Plane Algorithm,* was developed by L. Babai in 1986[3]. It integerizes every fractional coefficients to the nearest integral to identify the translated fundamental domain that the target vector locates. In an n dimensional lattice, the algorithm will produce an approximate closest vector with a factor ratio of $2(\frac{2}{\sqrt{3}})^n$ to the exact closest vector[3].

Lemma 3.1.5 (Babai's Closest Vertex Algorithm). *Let $L \subset \mathbb{R}^n$ be a lattice generated by a basis B, and let $\mathbf{w} \in \mathbb{R}^n$ be an arbitrary vector. If the vectors in the basis B are sufficiently orthogonal to each other, then the following algorithm solves apprCVP :*

Algorithm 1: BABAI'S CLOSEST VERTEX ALGORITHM

input : A basis B for the lattice and a vector $\mathbf{w} \in \mathbb{R}^n$
output: An approximate closest vector $\mathbf{v} \in L$

1 Solve $\mathbf{w} = B\mathbf{r}$ with $\mathbf{r} \in \mathbb{R}^n$;
2 **for** $i \leftarrow 1$ **to** n **do**
3 Set $z_i \leftarrow \lfloor r_i \rceil$;
4 **return** the vector $\mathbf{v} = B\mathbf{z}$;

The notation $\lfloor r_i \rceil$ in line 3 means z_i is set to be an integer *nearest* r_i.

The quality of the output vector $\mathbf{v}$ from **BABAI'S** algorithm-(1) heavily depends on how orthogonal the vectors of the basis B are. In general, if the vectors in the basis are reasonably orthogonal to each other, then the algorithm solves some versions of CVP very well, but if the basis vectors are highly nonorthogonal, then the vector $\mathbf{v}$ returned by the algorithm is generally far away from the exact closest lattice vector of $\mathbf{w}$.

We will illustrate the accuracy of the vector **v** produced by **BABAI'S CLOSEST VERTEX ALGORITHM**-(1) (short as **BABAI'S** algorithm-(1)) in the **Example**-(3.2.1).

3.2 The GGH Public Key Cryptosystem

In 1996, Oded Goldreich, Shafi Goldwasser, and Shai Halevi[16] introduced a new Public-Key cryptographic system based on the hardness of solving CVP in a high dimension lattice called the GGH cryptosystem named after the authors. Comparing with traditional cryptosystems such as RSA, ElGamal and Diffie-Hellman, the GGH cryptosystem is surprisingly simple but born with high performance taking advantage of matrix operations of modern computers. Besides, GGH is hard to break even in average case[24].

The basic idea of the GGH cryptosystem is a straightforward mathematical problem. Given a target point and a lattice with two different bases, namely the *Private Key* and the *Public Key*. The Private Key is a nearly orthogonal basis B_{good}, and the Public Key B_{bad} is a bad basis that is far away from being orthogonal. According to the **BABAI'S** algorithm-(1), for the target point, the good basis B_{good} can find the correct closest lattice point with a high possibility, but B_{bad} cannot solve CVP in the lattice. In such way, the message will be transfered successfully yet keep security.

However, the security analysis of GGH is not well researched as other cryptosystems, such as RSA and ElGamal. It is fair to say that decrypting the GGH using the Babai's algorithm and the Public Key is as hard as solving CVP in a given lattice[24].

3.2.1 GGH Cryptosystem

To initiate a GGH cryptosystem, Alice begins with constructing a *Private Key*, which will be kept secretly throughout the whole information exchanging process. In order to achieving this, she selects a full-column rank integer matrix:

$$V = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n],\ (\mathbf{v}_i \in \mathbb{Z}^n,\ 1 \leq i \leq n).$$

The columns in V are reasonably orthogonal to one another, which can be checked by calculating the *Hadamard Ratio* of the matrix V. Alice then chooses this matrix V as her Private Key.

Suppose L is the lattice generated by the columns of the matrix V. Alice then randomly generates an integer n-by-n *unimodular* matrix U, which satisfies $\det(U) = \pm 1$. To construct the unimodular matrix U, Alice can randomly generate a sequence of elementary matrices and multiply them together. Hence a new matrix W can be created by computing $W = VU$, and W^{-1} is easily to be discovered. Alice keeps computing the integer matrix W repeatedly until the the Hadamard Ratio of W is small enough. Then the integer matrix W can be chosen as the *Public Key* and be published to audience. According to the **Proposition**-(2.1.3), W is another generator matrix of the lattice L.

1. **ENCRYPTION PROCESS**

   A plaintext $\mathbf{m}$ for GGH is an integer vector of the same dimension, which agrees the lattice L. Bob encrypts the plaintext $\mathbf{m}$ with the Public Key W. A small perturbation integer vector $\mathbf{r}$ called an *ephemeral key* will be added to the encrypted vector at the same time. Geometrically, the *Euclidean Length* of $\mathbf{r}$ should be less than a half of the shortest distance of any two adjacent points in L. By doing this,

the plaintext $\mathbf{m}$ is then encrypted to an integer ciphertext vector $\mathbf{e}$ which does not belong to the lattice L;

i.e.,

$$\mathbf{e} = W\mathbf{m} + \mathbf{r} \tag{3.2.1}$$

In order to check the correctness of the plaintext $\mathbf{m}$ that Alice decrypted, Bob can hash the plaintext $\mathbf{m}$ and send the hashed value h to Alice at the same time. Various hash methods are available to choose, the widely used hash function *MD5*, for example.

2. **DECRYPTION PROCESS**

Decrypting the ciphertext $\mathbf{e}$ is straightforward. Alice uses her Private Key V to find the closest vector $\mathbf{v}$ of the ciphertext vector $\mathbf{e}$ by the **BABAI**'s algorithm-(1). To discover the original plaintext $\mathbf{m}$, Alice can simply compute it with the Public Key W by the formula below:

$$\mathbf{m} = W^{-1}\mathbf{v} \tag{3.2.2}$$

Alice can check the correctness of the plaintext $\mathbf{m}$ by comparing it with the hashed value h.

3. **SECURITY PROTECTION**

If a third person Eva listened the ciphertext $\mathbf{e}$, the only basis available to her is the Public Key W, which is a hardly orthogonal basis for the lattice L. Therefore, the closest vector $\bar{\mathbf{v}}$ she decodes is far away from the exact closest point $\mathbf{v}$. Hence, the plaintext $\bar{\mathbf{m}}$ decrypted by Eva is incorrect. The original information $\mathbf{m}$ keeps safe.

The **Figure**-(3.3) illustrates how the GGH cryptosystem protects a plaintext **m**. The Private Key V and Public Key W generate the same two dimensional lattice. A ciphertext point **e** is shown as a solid circle in the graph. The exact closest vector in the lattice L is **v**. However, what Eva can find is $\tilde{\mathbf{v}}$ by the **BABAI**'s algorithm-(1) and the Public Key, which is not the correct vector.

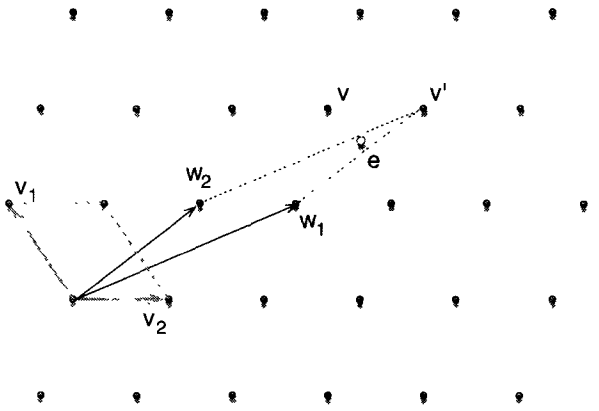

Figure 3.3: Decrypting the closest vector by two bases

In summary, the GGH cryptosystem works with the following steps as illustrated in **Table**-(3.1) :

To guarantee the GGH cryptosystem works, one has to choose an appropriate perturbation vector **r**. It cannot be too small such that the closest point **v** can be discovered using the Public Key without any difficulties, and it cannot be too large that it is impossible to be decrypted even by the Private Key.

Example 3.2.1 (A 3-dimensional GGH example). We demonstrate the GGH cryptosystem with the lattice and the bases in **Example**-(2.1.4) of chapter 2.

Suppose the Private Key V chosen by Alice is in **Example**-(2.1.4), and the Public Key is B. Their *Hadamard Ratio*s are:

$$\mathcal{H}(V) \approx 0.75, \mathcal{H}(W) \approx 0.00002.$$

Hence they both fulfil the orthogonality conditions that the GGH cryptosystem requires.

If Bob wants to send Alice a vector message $\mathbf{m} = [86 \ -35 \ -32]^T$, he can randomly generate a perturbation vector $\mathbf{r} = [-4 \ -3 \ -2]^T$ with relatively small entries comparing with the vectors in Public Key, and he encrypts the plaintext $\mathbf{m}$ with **Equation**-(3.2.1) to produce the ciphertext:

$$\mathbf{e} = [-79081427 \ -35617462 \ 11035473]^T.$$

To decrypt the ciphertext $\mathbf{e}$, Alice first computes the real solution of the linear equation system $V\mathbf{x} = \mathbf{e}$, using the privaite key V. It is:

$$\mathbf{x}_V \approx [81878.97 \ -292300.00 \ 443815.04]^T.$$

According to the **BABAI**'s algorithm-(1), Alice rounds the elements of $\mathbf{x}_V$ into the nearest integers, then she gets an integer coefficient vector corresponding to the privaite key V:

$$\mathbf{z}_V = [81879 \ -292300 \ 443815]^T.$$

Therefore, the closest vector Alice discovers is:

$$\mathbf{v}_V = V\mathbf{z}_V = [-79081423 \ -35617459 \ 11035471]^T.$$

Then, the last step Alice does is to recover the plaintext $\mathbf{m}_V$ by solving the integer linear equation system $W\mathbf{m}_V = \mathbf{v}_V$, which gives her the formula:

$$\mathbf{m}_V = W^{-1}\mathbf{v}_V.$$

Finally, Alice discovers a plaintext:

$$\mathbf{m}_V = [86 \ -35 \ -32]^T.$$

which is identical with the original plaintext **m**. We can see, Alice successfully decrypts the plaintext **m** back by the Private Key V and the **BABAI**'s algorithm-(1).

However, the basis Eva can use to decrypt the cipertext **e** using the **BABAI**'s algorithm is the Public Key W, the corresponding vectors of the above steps are:

The real solution of the linear equation system $W\mathbf{x} = \mathbf{e}$:

$$\mathbf{x}_W \approx [75.76 \ -34.52 \ -24.18]^T.$$

The rounded integer coefficient vector corresponding to the Public Key W:

$$\mathbf{z}_W = [76 \ -35 \ -24]^T.$$

The closest vector Eva discovers is:

$$\mathbf{v}_W = W\mathbf{z}_W = [-79508353 \ -35809745 \ 11095049]^T.$$

The plaintext that Eva decrypts finally is:

$$\mathbf{m}_W = [76 \ -35 \ -24]^T.$$

which is far from the original plaintext **m**.

3.2.2 Attacks on GGH

The first remarkable attack to the GGH cryptosystem comes from Phong Nguyen[38] in 1999. The attack works by carefully choosing n linearly independent lattice points, then constructing a new basis to generate a sublattice of the original lattice by those n points. Then we attack the sublattice to identify the potential properties of the original lattice. Nguyen shows that the dimension of GGH lattice should be larger than 350 to

guarantee the security. The attack followed was introduced by Han, Daewan and Kim, Myung-Hwan and Yeom, Yongjin[19], who constructed a *Paeng-Jung-Ha cryptosystem*, and use lattice reduction algorithms to attack GGH using this scheme. It shows that the signature encrypted with the GGH cryptosystem is insecure in certain special situations. When the special stiuation occurs, the GGH signature is insecure even the dimension of the GGH is as large as 1000.

Since the GGH cryptosystem relies on the orthogonality of the bases for a lattice, i.e., the Private Key V must be much more orthogonal than the Public Key W. Therefore, Alice can always find the correct closest vector of the ciphertext $\mathbf{e}$ simply taking advantages of the good basis V, but Eva cannot. So an intuitive idea would be "*Can Eva discover a more orthogonal basis from the Public Key ?* ". It turns out that the *Lattice Reduction* algorithms can produce nearly orthogonal bases based on the Public Key. Hence running *Lattice Reduction* algorithms on the Public Key will give us good bases that can be regarded as *quasi* private keys to attack the GGH cryptosystem.

The three Lattice Reduction algorithms, the LLL algorithm, the approximate Optimal Reduction algorithm and the Optimal Reduction algorithm, will be introduced in the chapter 5. The experimental data of the GGH attacking from those three algorithms will be shown in the chapter 6.

Alice	**Bob**
Key Creation	
Choose a nearly orthogonal basis V as the Private Key; Choose an unimodular matrix U; Compute a bad basis $W = VU$; Publish W as the Public Key.	
Encryption	
	Choose an integer plaintext vector $\mathbf{m}$; Choose an *ephemeral key* $\mathbf{r}$; Use Alice's Public Key to encrypt: $\mathbf{e} = W\mathbf{m} + \mathbf{r}$. Hash the plaintext $\mathbf{m}$ to get h; Send the ciphertext $\mathbf{e}$ and h to Alice.
Decryption	
Use the Private Key V and the **BABAI**'s algorithm-(1) to decrypt the closest vector $\mathbf{v} \in L$ of $\mathbf{e}$; Compute $W^{-1}\mathbf{v}$ to discover $\mathbf{m}$; Check the correctness by comparing hash value of $\mathbf{m}$ with h.	

Table 3.1: The GGH Cryptosystem

Chapter 4

Enumeration Algorithms for Solving SVP

Although the GGH and other lattice related cryptosystems are mainly based on the hardness of solving CVP, solving SVP is unavoidable during the research activities. In fact, finding a shortest vector in a lattice is a fundamental and basic process in lattice reduction algorithms. Because it is always possible to transform another lattice related problems to an SVP solving problem by applying corresponding strategies. In Chapter 5, we will show how heavily the lattice reduction algorithms rely on the shortest vector discovered. Various qualities of bases can be produced by the same lattice reduction algorithm, if we plug different SVP solving methods into it. Just like the qualities of produced bases are greatly influenced by SVP solving algorithms, the performance of lattice reduction algorithms are dominanted by the performance of those SVP solving algorithms as well.

Not surprising, many approaches have been proposed to solve SVP and apprSVP. For example, the LLL algorithm[29] and Schnoor's algorithm[44] are good choices for

discovering an approximate shortest vector in polynomial time. Likewise, the Kannan's algorithm[25, 26] and Ajtai's algorithm[2] are more expensive, yet give us an exact shortest vector for a given lattice.

This chapter focuses on exact SVP solving algorithms, among which the main algorithm we will describe is the *Basis Enumeration* algorithm. In the first half part of this chapter, we will introduce the Gram-Schmidt Orthogonalization algorithm. Although the Gram-Schmidt Orthogonalization is a widely used algorithm in vector space operations, it is still a valuable tool on estimations of the searching ranges in which all candidates located for the basic enumeration algorithm. This algorithm will be shown in the second half part of this chapter. In the end of this chapter, we will give a brief introduction to other SVP solving algorithms, the Kannan's algorithm and the Ajtai's algorithm, for example.

4.1 Gram-Schmidt Orthogonalization

Usually there are more than one basis for a given lattice L, and the bases are all related with each other by an unimodular matrix according to the **Proposition**-(2.1.3). We are more interested in a special group of bases named *orthogonal bases*, which may not exist in the most of the cases though. It has been shown in chapter 3 that for a given lattice, the Private Key is much better than the public when they are both acting as bases of the lattice where the GGH cryptosystem constructs. In this thesis, the Hadamard Ratio is introduced as the measurement tool to identify the qualities of the keys.

Given a lattice L and its basis $B = \{ \mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n \}$, B is called an *orthogonal basis* if every pair of vectors in basis B satisfies:

$$\langle \mathbf{b}_i, \mathbf{b}_j \rangle = 0, \quad (i \neq j \text{ and } 1 \leq i, j \leq n).$$

where "$\langle \cdot, \cdot \rangle$" is the *inner product* of two vectors. Additionally, B is called an *orthonormal* basis, if $\|\mathbf{b}_i\|_2 = 1$ ($1 \leq i \leq n$), where $\|\mathbf{b}_i\|_2 = \sqrt{\mathbf{b}_{i,1}^2 + \mathbf{b}_{i,2}^2 + \cdots + \mathbf{b}_{i,n}^2}$. In general, a lattice has no orthogonal basis.

Remark 4.1.1 (**Gram-Schmidt Orthogonalization**). Let $B = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n] \in \mathbb{R}^{m \times n}$ be a full-column matrix whose columns span a *vector space* $V \subset \mathbb{R}^m$. Then the following algorithm creates an orthogonal matrix $B^* = [\mathbf{b}_1^*, \mathbf{b}_2^*, \ldots, \mathbf{b}_n^*]$ from the given matrix B:

$$\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^*, \qquad (1 \leq i \leq n). \tag{4.1.1}$$

where $\mu_{ij} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\|\mathbf{b}_j^*\|_2^2}$, for $1 \leq j < i \leq n$.

Algorithm 2: GRAM-SCHMIDT ORTHOGONALIZATION

input : A full-column rank matrix B
output: A Modified *orthogonal* matrix B^*

```
1 Set b*_1 = b_1 ;            // special treatment for the first vector
2 for i ← 2 to n do
3 |   for j ← 1 to i − 1 do
4 |   |   Compute μ_ij = ⟨b_i, b*_j⟩ / ‖b*_j‖_2^2 ;
5 |   Set b*_i = b_i − Σ_{j=1}^{i−1} μ_ij b*_j ;
6 return B* ;
```

Let B^* be the matrix constructed in GRAM-SCHMIDT ORTHOGONALIZATION algorithm-(2) from the given matrix B, it has been proved[46] that the vector space spanned by the columns of B^* is identical with the vector space V that the columns of B spanned. This

remark tells us that the Gram-Schmidt Orthogonalization can be freely invoked in matrix transformation algorithms of vector spaces.

On the contrary, the Gram-Schmidt Orthogonalization (shortened as GSO in the rest of the thesis) cannot be used directly in lattice bases transformations in general. It destroys the integral-coefficient property preserved according to the definition of lattice. However, the main idea of Gram-Schmidt Orthogonalization acts as a key procedure in some special lattice bases operations, lattice Projection, for example.

4.2 Lattice Projection

We have introduced the **GRAM-SCHMIDT ORTHOGONALIZATION** algorithm-(2) in the previous section. However, the geometric meaning of **GSO** has not been mentioned yet. The progress of **GSO** turns out to be an important technique for estimating the length of vectors in SVP solving algorithms named *Projection.* It is necessary to give an introduction to the details of the Projection[23, 36, 12] .

Let $B_{m\times n} = [\,\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\,]$ be a full-column rank matrix, and $B^* = [\,\mathbf{b}_1^*, \mathbf{b}_2^*, \ldots, \mathbf{b}_n^*\,]$ be the orthogonal matrix which is constructed using **GRAM-SCHMIDT ORTHOGONALIZATION** algorithm-(2) from the given matrix B. Denote Span(B), (or equivalently Span($\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$)) , the *Vector Space* spanned by the columns of the matrix B, that is:

$$\text{Span}(B) = \text{Span}(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n) = \{\, B\mathbf{r} \mid \mathbf{r} \in \mathbb{R}^n \,\}.$$

We can see that Span(B) is a subset of $\mathbb{R}^m$. Also, it is easy to prove that ,

$$\text{Span}(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_k) = \text{Span}(\mathbf{b}_1^*, \mathbf{b}_2^*, \ldots, \mathbf{b}_k^*), \quad \text{for any } k \ (1 \le k \le n).$$

Suppose L is an n dimensional lattice generated by the columns of matrix B, Then the lattice L is included in the vector space spanned by the columns of B, that is $L(B) \subset \text{Span}(B)$.

Definition 4.2.1 (Orthogonal Complement). Let $B_{m\times n}$ be a full-column rank matrices, and $W_{m\times p}$ $(p \leq n)$ be a submatrix of B. The *orthogonal complement* of Span(W) (in the vector space Span(B)) represented by Span$^{\perp}(W)$ is:

$$\text{Span}^{\perp}(W) = \{\, \mathbf{v} \in \text{Span}(B) \mid \langle \mathbf{v}, \mathbf{w} \rangle = 0, \quad \text{for all } \mathbf{w} \in \text{Span}(W) \,\}.$$

Assume that $W_{p\times n} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_p]$, we define Span$^{\perp}(W)$ = Span$^{\perp}(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_p)$. It has been proved[24, Chapter 6] that any vector $\mathbf{v} \in$ Span(B) can be uniquely decomposed as :

$$\mathbf{v} = \bar{\mathbf{v}} + \mathbf{w} \tag{4.2.1}$$

for an unique vector $\bar{\mathbf{v}} \in$ Span(W) , and an unique vector $\mathbf{w} \in$ Span$^{\perp}(W)$.

Assume $W^* = [\, \mathbf{w}_1^*, \mathbf{w}_2^*, \ldots, \mathbf{w}_p^* \,]$ is the orthogonal matrix constructed using **Gram-Schmidt Orthogonalization** algorithm-(2) from the matrix W. Then the vector $\bar{\mathbf{v}}$ in **Equation**-(4.2.1) can be computed by,

$$\bar{\mathbf{v}} = \mathbf{v} - \sum_{i=1}^{p} \mu_i \mathbf{w}_i^*, \tag{4.2.2}$$

where $\mu_i = \frac{\langle \mathbf{v}, \mathbf{w}_i^* \rangle}{\|\mathbf{w}_i^*\|_2^2}$, for $1 \leq i \leq p$.

Definition 4.2.2 (Projection). Let $B_{m\times n}$ be a full-column rank matrix, and let $W_{m\times p}$ $(p \leq n)$ be a submatrix of B. For a vector $\mathbf{v} \in$ Span(B), the vector $\bar{\mathbf{v}}$ in **Equation**-(4.2.1) is called the *Projection* of vector $\mathbf{v}$ on the *orthogonal complement* of Span(W).

Given a lattice L and its generator matrix $B = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n]$. Let $\mathbf{v}$ be an arbitrary vector in $\mathbb{R}^n$, we denote $\mathbf{v}(k)$, (for all $1 \leq k \leq n$), the *projection* of the given vector $\mathbf{v}$ on $\text{Span}^{\perp}(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{k-1})$. Accordingly, $L_k(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n)$ represents the projection of lattice $L(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n)$ on the vector space $\text{Span}^{\perp}(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{k-1})$. When there is no confusion, we use a simple symbol L_k for $L_k(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n)$. For the special case $k = 1$, it is not hard to see that $\mathbf{v}(1) = \mathbf{v}$ and $L_1 = L$.

The **Figure**-(4.4) illustrates the projections of a given lattice L and its generator matrix $B = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n]$.

$$
\begin{array}{lccccccc}
L_1 & \mathbf{b}_1(1) & \mathbf{b}_2(1) & \cdots & \mathbf{b}_k(1) & \mathbf{b}_{k+1}(1) & \cdots & \mathbf{b}_n(1) \\
L_2 & & \mathbf{b}_2(2) & \cdots & \mathbf{b}_k(2) & \mathbf{b}_{k+1}(2) & \cdots & \mathbf{b}_n(2) \\
\vdots & & & \ddots & \vdots & \vdots & \cdots & \vdots \\
L_k & & & & \mathbf{b}_k(k) & \mathbf{b}_{k+1}(k) & \cdots & \mathbf{b}_n(k) \\
L_{k+1} & & & & & \mathbf{b}_{k+1}(k+1) & \cdots & \mathbf{b}_n(k+1) \\
\vdots & & & & & & \ddots & \vdots \\
L_n & & & & & & & \mathbf{b}_n(n)
\end{array}
$$

Figure 4.4: Projections of a lattice L with a basis B

It has been proved[23] that the set of projected vectors $\{ \mathbf{b}_k(k), \mathbf{b}_{k+1}(k), \ldots, \mathbf{b}_n(k) \}$ forms a basis for the projected lattice L_k of dimension $n - k + 1$, for all k $(1 \leq k \leq n)$. Recall the process of **GRAM-SCHMIDT ORTHOGONALIZATION** algorithm-(2), we can see that the operation of *Projecting* L on $Span^{\perp}(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{k-1})$ is actually applying the **GSO** algorithm to the columns of B on the columns $\{ \mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{k-1} \}$. Thus the projected basis for the projected lattice L_k are obtained with the following equations:

$$\mathbf{b}_k(i) = \mathbf{b}_k - \sum_{j=1}^{i-1} \mu_{kj} \mathbf{b}_j(j), \qquad 1 \leq i < k \leq n. \tag{4.2.3}$$

where $\mu_{kj} = \frac{\langle \mathbf{b}_k, \mathbf{b}_j(j) \rangle}{\langle \mathbf{b}_j(j), \mathbf{b}_j(j) \rangle}$.

We show in **Figure**-(4.5) the process of projection in a 2-dimensional lattice L and its generator matrix $B = [\mathbf{b}_1, \mathbf{b}_2]$. The vector $\mathbf{b}_2(1)$ is created by projecting the vector $\mathbf{b}_2$ on the orthogonal complement of Span($\mathbf{b}_1$).

However, $\mathbf{b}_2$ is not the only vector that can produce the projected vector $\mathbf{b}_2(1)$. For example, if we project another vector $\mathbf{u}_2$ on the orthogonal complement of Span($\mathbf{b}_1$), the projected vector is also $\mathbf{b}_2(1)$. The fact that a projected vector can be produced by many vectors before the projection process, can help us to formulate the estimation of searching ranges for the basic enumeration algorithm.

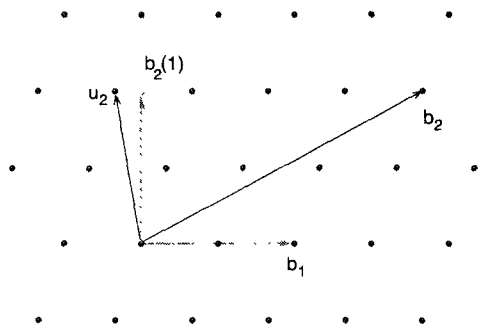

Figure 4.5: Projecting the vector $\mathbf{b}_2$ and $\mathbf{u}_2$ in lattice L

4.3 Basic Enumeration Algorithm

To find a shortest vector in a given lattice L, an intuitive and natural idea is to enumerate all possible candidates, then compare them to discover the one with the shortest length. This exhaustive searching method is the basic idea among all exact SVP solving algorithms, such as Kannan's algorithm[25, 26], Ajtai's algorithm[2]. A common approach

for SVP solving algorithms is to embed an enumeration searching procedure in low dimension operations, and reduce the lengths of the vectors in bases in high dimension operations simultaneously. There are a few accelerating algorithms can be attached to decease the size of exhaustive candidates set. For example, the LLL algorithm can reduce the set to $2^{\mathcal{O}(n^2)}$ candidates[45] to speed up the searching procedure.

Given a lattice L, its generator matrix B and the orthogonal matrix $B^* = [\mathbf{b}_1^*, \mathbf{b}_2^*, \ldots, \mathbf{b}_n^*]$ from the **GSO** algorithm-(2). Assume $\mathbf{v} = B\mathbf{z}$ is a shortest vector in L, where $\mathbf{z}$ is a coefficients vector in $\mathbb{Z}^n$. Then,

$$\begin{aligned}\mathbf{v} &= \sum_{i=1}^{n} z_i \mathbf{b}_i \\ &= \sum_{i=1}^{n} z_i (\mathbf{b}_i^* + \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^*) \quad \text{(by the } \textbf{Equation} - (4.1.1)\text{)} \\ &= \sum_{j=1}^{n} (z_j + \sum_{i=j+1}^{n} \mu_{i,j} z_i) \mathbf{b}_j^* \end{aligned} \tag{4.3.1}$$

Combining the two **Equation**s (4.2.3) and (4.3.1), we can derive a sequence of projections of the shortest vector $\mathbf{v}$ on the orthogonal complement of the hyperspheres $\text{Span}^{\perp}(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{k-1})$:

$$\mathbf{v}(k) = \textstyle\sum_{j=k}^{n} (z_j + \sum_{i=j+1}^{n} \mu_{i,j} z_i) \mathbf{b}_j^*, \quad (1 \le k \le n)$$

and the norms of projections by

$$\|\mathbf{v}(k)\|_2 = \sum_{j=k}^{n} |z_j + \sum_{i=j+1}^{n} \mu_{i,j} z_i| \cdot \|\mathbf{b}_j^*\|_2, \quad (1 \le k \le n). \tag{4.3.2}$$

According to Pythagorean theorem[28, Chapter 11], the **Formula**-(4.3.2) gives us the relation between the vector $\mathbf{v}$ and its n projections in term of their lengths, that

$$\|\mathbf{v}(k)\|_2 \le \|\mathbf{v}\|_2, \quad (1 \le k \le n).$$

We can see the above n inequations also show that all the n projections agree to the same upper bound. This upper bound is the length of the shortest vector. Therefore, by searching all projected vectors within this upper bound, finding a shortest vector is possible.

To start an enumeration search in the lattice, we have to give a guess P for the upper bound as the estimation of the length of a shortest vector. For example, we can let $P = \|\mathbf{b}_1\|_2$, or $P = \sqrt{\gamma_n}\text{Vol}(L)^{1/n}$[33], where γ_n is the *Hermite Constant*, which can be estimated by the inequation $\gamma_n \leq 1 + \frac{n}{4}$. We will choose this γ_n to estimate the length of a shortest vector in our algorithms.

Taking advantage of the guessed upper bound P of the target vector $\mathbf{v}$, we can find the target vector $\mathbf{v}$ by enumerating all possible coefficients $\mathbf{z}$ in the equation $\mathbf{v} = B\mathbf{z}$. Considering the projected components, we can rewrite (in a few steps of computation though) the **Formula**-(4.3.2) to n inequations, which are useful to estimate the distribution ranges of the coefficients $\mathbf{z}$:

$$
\begin{aligned}
(z_n)^2 \cdot \|\mathbf{b}_n^*\|_2^2 &\leq P^2 \\
(z_{n-1} + \mu_{n,n-1} z_n)^2 \cdot \|\mathbf{b}_{n-1}^*\|_2^2 &\leq P^2 - (z_n)^2 \cdot \|\mathbf{b}_n^*\|_2^2 \\
&\cdots \\
(z_k + \sum_{i=k+1}^{n} \mu_{i,j} z_i)^2 \cdot \|\mathbf{b}_k^*\|_2^2 &\leq P^2 - \sum_{j=k+1}^{n} (z_j + \sum_{i=j+1}^{n} \mu_{i,j} z_i)^2 \cdot \|\mathbf{b}_j^*\|_2^2 \\
&\cdots \\
(z_1 + \sum_{i=2}^{n} \mu_{i,j} z_i)^2 \cdot \|\mathbf{b}_1^*\|_2^2 &\leq P^2 - \sum_{j=2}^{n} (z_j + \sum_{i=j+1}^{n} \mu_{i,j} z_i)^2 \cdot \|\mathbf{b}_j^*\|_2^2
\end{aligned}
$$

Let variable k be the cursor point to the index of the coefficient we are searching currently. We start the exhaustive search from estimating the last coefficient z_n ($k = n$)

to the first one z_1 ($k = 1$). When k equals n, the possible range where the coefficient z_n locates can be determined by $|z_n| \leq \frac{P}{\|\mathbf{b}_n^*\|}$. Iteratively calculating coefficients z_i ($k < i \leq n$) according to the above n inequations until $i = k$, we can get the interval I_k for the coefficient z_k, such that $z_k \in I_k$. Hence the ranges of coefficients z_k, z_{k+1}, ... , z_n are discovered. When the index k reaches 1, the enumerating search of coefficients $\mathbf{z}$ will give us a shortest vector $\mathbf{v}$.

The algorithm **ENUMERATE**-(3) mimics the n inequations to find a shortest vector.

Algorithm 3: ENUMERATE

Input : A full-column rank matrix B, an estimation P
Output: An integer vector $\mathbf{z}_{min}$ such that $B \cdot \mathbf{z}_{min}$ forms a shortest vector

Set $B^* \leftarrow \mathsf{GSO}(B)$;
Set $\mathbf{z}_{min} = (0,0,\ldots,0)$, $\mathbf{z} = (0,0,\ldots,0)$, $\mathbf{l} = (0,0,\ldots,0)$;
$i \leftarrow n$;
while $i \leq n$ **do**
  Set $l_i = (z_i + \sum_{j>i} z_j \mu_{j,i})^2 \|\mathbf{b}_i^*\|_2^2$;
  if $(\sum_{j=i}^{n} l_j \leq P)$ **then**
    if $i > 1$ **then**
      $i = i - 1$;
      $z_i = \mathsf{Floor}(-\sum_{j=i+1}^{n} z_j \mu_{j,i} - \sqrt{\frac{P - \sum_{j>i} l_j}{\|\mathbf{b}_i^*\|_2^2}})$;
    else
      $P = \sum_{j=1}^{n} l_j$;
      $\mathbf{z}_{min} = \mathbf{z}$;
      $z_i = z_i + 1$;
  else
    $z_i = z_i + 1$;
return $\mathbf{z}_{min}$

The running time of the algorithm **ENUMERATE**-(3) can be significantly decreased if

we have a good guess P for the upper bound of the length of a shortest vector, which substitutes as an input parameter in the algorithm. The implementation of **ENUMERATE**-(3) uses a depth first search strategy[10, 20]. The vectors $\mathbf{z}$ in line 2 represents the integer coefficients to be enumerated, and the vector $\mathbf{l}$ represents the length of the enumerated point of the coefficients $\mathbf{z}$. As soon as the length of the vector that $\mathbf{z}$ represents is shorter than P, P is decreased to current length in order to reduce the searching ranges calculated in line 11. The line 12 updates $\mathbf{z}_{min}$ under which the shorter length is enumerated.

When the algorithm **ENUMERATE**-(3) terminates, the integer vector $\mathbf{z}_{min}$ is returned, such that $B \cdot \mathbf{z}_{min}$ forms a shortest vector.

4.4 Other Algorithms for Solving SVP

Kannan's deterministic method[25, 26] is another enumeration algorithm, which enumerates the set of candidate points in a super-exponential complexity $\mathcal{O}(n^{9n})$. After taking advantage of a quasi-HKZ-reduced basis, a stronger reduced basis than the LLL-Reduced basis[45], the Kannan's enumeration set can be significantly narrowed, such that the complexity of finding a shortest vector is reduced to $\mathcal{O}(n^{n+o(n)})$. If we embed the LLL reduction algorithm into Kannan's algorithm to transform the given basis into a quasi-HKZ-reduced basis, the complexity can be further reduced to $P(n)2^{\mathcal{O}(n\log n)}$ with a polynomial $P(n)$ of fixed n. Helfrich[23] improved the complexity of Kannan's algorithm to an upper bound of $n^{0.5n+o(n)}$ in 1985. Guillaume Hanrot and Damien Stehlé[20] re-analyzed Kannan's algorithm and improved the above upper bound to $n^{\frac{n}{2e}+o(n)} \approx n^{0.184n+o(n)}$ in worst case.

The main idea of the Kannan's algorithm is to construct a quasi-HKZ-reduced basis. After producing this basis, the first vector $\mathbf{b}_1$ of the constructed basis is a shortest

vector of the given lattice at the end of the algorithm. In order to build the quasi-HKZ-reduced basis, Kannan projected the given basis B to the first vector to achieve an $n-1$ dimensional projected lattice and its corresponding projected basis. By projecting the projected basis (which is one dimension less than B) recursively until the dimension is 1, the shortest projected vector can be easily found. Then Kannan lifts[12] the projected basis recursively back to dimension n. The shortest vector found is lifted back to the original lattice at the same time.

Ajtai's algorithm[2] is a randomized algorithm with exponential complexity $2^{\mathcal{O}(n)}$ times a fixed polynomial. The algorithm gives us a totally different strategy in finding a shortest vector under a overwhelming probability. The Ajtai's algorithm introduces a "sieving" method instead of reducing bases and enumerating potential points. P. Q. Nguyen and T. Vidick[37] shows that the Ajtai's algorithm is not only theoretically current but also implementable in practice. However, one of its drawback is that the algorithm needs exponential space during the sieving operations.

Chapter 5

Lattice Reduction Algorithms

We have seen in the chapter "Attacks on GGH" that one way of attacking the GGH cryptosystem, is to produce a basis better than the Public Key. Hence the closest vector discovered by this better basis is likely closer to the exact closest vector, comparing with the one found using the Public Key. Therefore, the decrypted plaintext message will have more same elements with the exact plaintext. Many algorithms have been proposed on the purpose of transforming a bad lattice basis to reasonably good ones. Those algorithms are called *lattice reduction* algorithms.

The *Hadamard Ratio* has been introduced to measure the quality of a given lattice basis in the chapter "GGH and Lattice Based Cryptography". When we try to generate a new basis using lattice reduction algorithms, the quality of the new produced basis is mainly determined by two issues. Firstly, it depends on what lattice reduction algorithm it used. Secondly, this quality also highly relies on the approach of finding a shortest vector within the lattice reduction algorithm. Therefore, when we plug different SVP solving algorithms to a lattice reduction algorithm, we can achieve various reduced bases of different qualities. For example, LLL algorithm[29] and Schnoor's algorithm[44] are

good choices for creating an approximate orthogonal basis in polynomial time. Kannan's algorithm[25] and Ajtai's algorithm[2] are more expensive, yet can be used to construct highly qualitative and nearly orthogonal bases.

In the beginning of this chapter, We will give a definition of the necessary condition named "Size Reduced" for all lattice reduction algorithms. After that, three lattice reduction algorithms will be introduced as the main algorithms of this thesis. The LLL algorithm is a polynomial time algorithm which is commonly used as an accelerator of other lattice reduction algorithms. Then we will introduce an approximate Optimally-Reduced algorithm. Its complexity is exponential, yet creates a much better basis for the given lattice. The last algorithm is an exact Optimally-Reduced lattice algorithm, which is the most expensive algorithm among the three methods, but generates the best basis for the lattice.

5.1 Size Reduced Bases

Given a lattice L and its generator matrix A, we know the column vectors in A are linearly independent. If applying the **GRAM-SCHMIDT ORTHOGONALIZATION** algorithm-(2) on A, we can form a *QR-Decomposition*[17, 14] of the matrix A. Then a new generator matrix B can be constructed further by:

$$\begin{aligned} B &= AZ \\ &= Q^*U = QDU \\ &= QR \end{aligned} \tag{5.1.1}$$

where the unimodular matrix Z acts as a matrix operator that transforms the matrix A to the matrix B. The two matrices $Q^* = [\mathbf{q}_1^*, \mathbf{q}_2^*, \ldots, \mathbf{q}_n^*]$ and $U = [\mu_{i,j}]_{n\times n}$ are respectively the orthogonal matrix and the upper triangular matrix produced right after the **GRAM-SCHMIDT ORTHOGONALIZATION** algorithm-(2). We compute the *Orthonormal* matrix $Q = [\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n]$ with column vectors $\mathbf{q}_i$ from the matrix Q^*, such that $\mathbf{q}_i = \mathbf{q}_i^* / \|\mathbf{q}_i^*\|_2$, which gives us $\|\mathbf{q}_i\|_2 = 1$. Let D be a diagonal matrix that $D = \text{diag}(\|\mathbf{q}_1^*\|_2, \|\mathbf{q}_2^*\|_2, \ldots, \|\mathbf{q}_n^*\|_2)$. After this, we set $R = DU = [r_{i,j}]_{n\times n}$ to an upper triangular matrix, such that $r_{i,j} = \mu_{i,j} \times \|\mathbf{q}_i^*\|_2$. Hence, a QR-Decomposition is finished.

Definition 5.1.1 (Size-Reduced). A lattice generator matrix $B = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n]$ is called *Size-Reduced*, if the output matrices U or R of the **GRAM-SCHMIDT ORTHOGONALIZATION** algorithm-(2) of B in (5.1.1) satisfy:

$$|\mu_{i,j}| \leq \tfrac{1}{2} \text{ or } |r_{i,j}| \leq \tfrac{1}{2}|r_{i,i}|, \quad (1 \leq i < j \leq n).$$

Size-Reduced is a necessary condition for lattice reduction algorithms. The geometric meaning of it is quite clear. It shows that the Euclidean Lengths of vectors in the matrix B are the shortest ones when runs the **GSO** algorithm-(2). Therefore the bases cannot be further reduced by *Gaussian Elimination Method*[24, Chapter 6] under current situations.

To find a best reduced basis, one may use the *Minkowski Minima*[32, 14] as a rule for determining the quality of bases produced by lattice reduction algorithms. Measuring their lengths by the Euclidean Norm, we say that λ_k ($1 \leq k \leq n$) is the kth *successive minimum* with respect to the given lattice L, if λ_k is the lower bound of the following sphere of the radius λ_k:

$$S_k = \{\, \mathbf{v} \mid \mathbf{v} \in L \text{ and } \|\mathbf{v}\|_2 \leq \lambda_k \,\},$$

which contains k linearly independent lattice points.

It is not hard to see that the Minkowski Minimas are actually the set of minimal Euclidean Lengths with respect to any n linearly independent vectors of the lattice L. Naturally, one may think we can combine a set of n vectors corresponding to the n Minkowski Minimas as the best basis for the lattice L.

Definition 5.1.2 (Minkowski-Reduced). Given a lattice L and its generator matrix $B = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n]$, B is called *Minkowski-Reduced*, if

$$\|\mathbf{b}_i\|_2 = \lambda_i, \qquad (1 \leq i \leq n).$$

For a lattice L of dimension n, it is not true that any n independent vectors can form a basis for the lattice. Thus the Minkowski-Reduced basis does not always exist. In the nineteenth century, Korkine and Zolotarev[40, Chapter 2] first discovered that a lattice may not have a Minkowski-Reduced bases.

Example 5.1.3 (A counterexample of Minkowski Reduced basis[14]). Suppose L is a 5-dimensional lattice generated by columns of the following matrix B:

$$B = \begin{bmatrix} 2 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

We construct a vector

$$\begin{aligned} \mathbf{v} &= [0\ 0\ 0\ 0\ 2]^T \\ &= -\mathbf{b}_1 - \mathbf{b}_2 - \mathbf{b}_3 - \mathbf{b}_4 + 2\mathbf{b}_5 \end{aligned}$$

such that $\mathbf{v} \in L$ and $\|\mathbf{v}\|_2 = 2$.

Thus, the five Minkowski Minimas are all equal 2. Their Euclidean Norms all equal 2. However, the columns of matrix $B' = [\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4, \mathbf{v}]$ cannot generate the given lattice L, because the vector $\mathbf{b}_5 \in L$ cannot be represented by any linear combinations of the columns of B'.

Since the Minkowski-Reduced bases may not exist. We will introduce and define a new *Optimal-Reduced* basis and an algorithm for constructing it in the Chapter "Optimal Reduced Algorithm".

5.2 LLL Reduction Algorithm

The LLL algorithm[29] is one of the most important milestones on research of lattice reduction methods, although it was first proposed in order to factorize polynomials. Theoretically, it runs in polynomial time $\mathcal{O}(n^4 \log P)$ (where n is the dimension of a given lattice L and P is the maximal Euclidean Length of vectors in the given basis). It produces a reduced basis, whose shortest vector can be regarded as a solution of apprSVP with a factor of $2^{(n-1)/2}$ to the length of a exact shortest vector. Though the above factor is exponential in theory, it performs surprisingly well in practice. Because of the high performance of the LLL algorithm in practice, it has become a fundamental tool in lattice reduction algorithms. Many SVP and CVP solving algorithms embed it as a precomputing procedure, for example, Kannan's SVP algorithm[25, 26], Schnorr's apprSVP algorithm[44], Ajtai's SVP algorithm[2], etc.

Definition 5.2.1 (LLL-Reduced). Given a lattice L, a parameter $\omega \in (0.25, 1.0)$ and its generator matrix B, B is called *LLL-Reduced* if the upper triangular matrix R from the

QR-Decomposition-(5.1.1) of B satisfies the following two conditions:

$$|r_{i,j}| \leq \frac{1}{2}|r_{i,i}|, \quad (1 \leq i < j \leq n) \qquad \text{(Size – Reduced Condition)} \qquad (5.2.1a)$$

$$r_{i,i}^2 + r_{i-1,i}^2 \geq \omega r_{i-1,i-1}^2, \qquad \text{(Lovász Condition)} \qquad (5.2.1b)$$

The Lovász Condition-(5.2.1b) above implies that the vectors in an LLL-Reduced generator matrix B are roughly sorted under a weak condition. Think about the two vectors in every 2×2 sub-matrix along the diagonal of the matrix R, the second vector cannot be too much shorter than the previous one. Recall the concepts of projection in the Chapter "Lattice Projection", we know that the condition (5.2.1b) also demonstrates the relation of projections between $\mathbf{b}_i$ and $\mathbf{b}_{i-1}$ on the orthogonal complement of hypersphere Span($\mathbf{b}_1$, $\mathbf{b}_2$, ... , $\mathbf{b}_{i-2}$).

Theoretically, the most commonly used value of the parameter ω is $\frac{3}{4}$. [14, 29] show that columns in LLL-Reduced generator matrices are approximations of Minkowski Minimas with a factor of 2^n, i.e., for an n dimensional LLL-Reduced generator matrix $B^* = [\mathbf{b}_1^*, \mathbf{b}_2^*, \ldots, \mathbf{b}_n^*]$ and Minkowski Minimas of the lattice L generated by the columns of matrix B^*, we have

$$2^{i-n}\lambda_i^2 \leq \|\mathbf{b}_i^*\|_2^2 \leq 2^{n-i}\lambda_i^2, \quad (1 \leq i \leq n).$$

Especially, $\mathbf{b}_1^*$ is an approximate solution of SVP for the lattice L, such that

$$\|\mathbf{b}_1^*\|_2^2 \leq 2^{n-1}\|\mathbf{v}_{shortest}\|_2^2.$$

where $\mathbf{v}_{shortest}$ is a shortest vector of the lattice L.

We can see in the definition of the LLL-Reduced basis that ω cannot be 1. When ω equals 1, the algorithm of constructing an LLL-Reduced basis does not always guarantee convergence. This extreme case that ω equals 1 is called *Optimal LLL-Reduced.* If

it converges, the Optimal LLL-Reduced bases satisfy a very nice property that $\|\mathbf{b}_i^*\|^2 \leq \frac{4}{3}\|\mathbf{b}_{i+1}^*\|^2$, $(1 \leq i \leq n-1)$. Within a few steps, we can arrive at an approximation of a shortest vector with the Optimal-LLL reduced factor that $\|\mathbf{b}_1\| \leq (\frac{4}{3})^{n-1}\|\lambda_1\|$. Unfortunately, there is no algorithm known so far which can provably produce an optimal-LLL reduced basis efficiently (polynomial time in the lattice's dimension n). As a result, we commonly set ω to a number that is close to 1 instead of $\frac{3}{4}$ in practice. We will introduce a slightly modified version of LLL algorithm[31, 39] for the convenience of matrix operations in this thesis.

Before introducing the LLL algorithm, we first give two procedures that will be invoked in the algorithm. They are procedure **DECREASE()** and procedure **SWAPRESTORE()**. The two procedures will be presented in this chapter are the matrix-based version of Franklin T. Luk and Sanzheng Qiao[30].

Given an integer matrix A of full-column rank whose columns generate a lattice L, the **GRAM-SCHMIDT ORTHOGONALIZATION** algorithm-(2) forms another lattice generator matrix $B = AZ = QR$. The procedure **DECREASE()** checks the Size-Reduced condition (5.2.1a) required for lattice reduced bases with two input parameters (i, j) and then reduce the corresponding basis vectors that conflict with the condition (5.2.1a); i.e., if we got $|r_{i,j}| > \frac{1}{2}|r_{i,i}|$ in some steps of process, the procedure **DECREASE()** updates R and Z to make sure $|r_{i,j}| \leq \frac{1}{2}|r_{i,i}|$, and hence the modified vectors satisfy the Size-Reduced condition (5.2.1a). The notation I_n in **DECREASE()** represents the identity matrix of dimension n, and $\mathbf{e}_i$ denotes the ith unit vector.

Line 3 of the procedure **DECREASE()** constructs an $n \times n$ matrix $Z_{i,j}$, we can easily check $\det(Z_{i,j}) = 1$. Thus $Z_{i,j}$ is an unimodular matrix that guarantees it does not change

Procedure Decrease(i, j)

Input : R, Z of the **GRAM-SCHMIDT ORTHOGONALIZATION** algorithm-(2) for the lattice basis B

Output: Modified R, Z that fulfilled the Size-Reduced condition-(5.2.1a)

1 **if** $|r_{i,j}| > \frac{1}{2}|r_{i,i}|$ **then**
2 Set $\gamma \leftarrow$ an integer nearest to $(r_{i,j}/r_{i,i})$;
3 FormMatrix $Z_{ij} = I_n - \gamma \mathbf{e}_i \mathbf{e}_j^T$;
4 Set $R \leftarrow RZ_{ij}$;
5 Set $Z \leftarrow ZZ_{ij}$;
6 **return** modified R, Z

the lattice L. Line 4 and 5 modify the matrices R and Z to satisfy the Size-Reduced condition (5.2.1a).

To construct an LLL-Reduced basis, another condition (5.2.1b) should be satisfied at the same time. The procedure **SWAPRESTORE()** deals with every two adjacent vectors of a generator matrix to check the Lovász Condition (5.2.1b) . It swaps the two columns of the matrix which are invalid to the condition (5.2.1b) such that the Lovász Condition is fulfilled, then it restores the modified matrix R back to the upper triangular form.

Procedure SwapRestore(i)

Input : R, Z and Q

Output: Modified R, Z and Q that fulfilled the Lovász Condition (5.2.1b)

1 **if** $r_{i,i}^2 + r_{i-1,i}^2 < \omega r_{i-1,i-1}^2$ **then**
2 Compute a plane reflection 2×2 matrix J_i ;
3 FormMatrix $Q_i = \text{diag}[I_{i-2}, J_i, I_{n-i}]$;
4 FormMatrix $\Pi_i = \text{diag}[I_{i-2}, P, I_{n-i}]$;
5 Set $R \leftarrow Q_i R \Pi_i$;
6 Set $Z \leftarrow Z\Pi_i$;
7 Set $Q \leftarrow QQ_i$;
8 **return** modified R, Z and Q

In line 2 of the procedure **SWAPRESTORE()**, we construct a *plane reflection* matrix J_i of the form

$$J_i = \begin{bmatrix} c & s \\ s & -c \end{bmatrix}.$$

The purpose of J_i is to let $J_i R_{i-1,i} P$ be a 2×2 upper triangular matrix, where $R_{i-1,i}$ is the 2×2 sub-matrix of R along its diagonal with the index $(i-1,\ i)$, and the matrix P satisfies,

$$P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Hence, P is a permutation matrix, which swaps the two columns $(\mathbf{b}_{i-1},\ \mathbf{b}_i)$ of the generator matrix B. Line 4 builds the n dimensional permutation matrix Π_i which swaps $\mathbf{b}_{i-1}$ and $\mathbf{b}_i$. Therefore, at line 5 and 6, the permutation matrix works. The matrix Q_i constructed in line 3 will restore the swapped matrices R back to an upper triangular matrix. Then the Lovász Condition (5.2.1b) for the two elements (r_{i-1}, r_i) is satisfied in the modified matrix R. For more details on how to construct J_i and the rest of the procedure, see [31, Page 447-448] or [30].

Theorem 5.2.2 (**MATRIX LLL ALGORITHM**). *Given a* lattice $L \subset \mathbb{R}^n$, *a parameter* ω, ($0.25 < \omega < 1$), *and one of its generator matrix* $B = [\mathbf{b}_1,\ \mathbf{b}_2,\ \ldots\ ,\ \mathbf{b}_n] \in \mathbb{R}^{n \times n}$. *Let* R *and* Q *be the upper triangular matrix and the orthogonal matrix constructed in QR-Decomposition-*(5.1.1). *The following algorithm computes an integer unimodular matrix* Z, *and overwrites the matrices* Q *and* R, *such that* BZ *forms an LLL-Reduced generator matrix.*

Algorithm 4: MATRIX LLL REDUCTION ALGORITHM

Input : R, Q

Output: Modified R, Q and an Unimodular Matirx Z

```
1  Set Z ← I and Q ← I ;
2  k ← 2 ;
3  while k ≤ n do
4      if |r_{k-1,k}| > (1/2)|r_{k-1,k-1}| then
5          DECREASE(k − 1, k) ;
6      if r²_{k,k} + r²_{k-1,k} < ω r²_{k-1,k-1} then
7          SWAPRESTORE(k) ;
8          k ← Max (k − 1, 2) ;
9      else
10         for i ← (k − 2) downto 1 do
11             if |r_{i,k}| > (1/2)|r_{i,i}| then
12                 DECREASE(i, k) ;
13         k ← k + 1 ;
14 return the Unimodular Matrix Z
```

Proof. For any ω such that $0.25 < \omega < 1$, the **MATRIX LLL REDUCTION** algorithm-(4) guarantees terminates. Every "SWAP" action decreases the Euclidean Length of the corresponding basis vectors with a factor at least $\sqrt{\omega}$, which makes the algorithm goes toward to termination eventually. The procedure **SWAPRESTORE**(k) in **while** loop will be called in polynomial time if the generator matrix B is an integer matrix, see [29] for the full proof. □

The **LLL REDUCTION** algorithm-(4) first sets a variable k as a cursor. The algorithm terminates when k arrives at $n+1$. It checks the two conditions (5.2.1a) and (5.2.1b) required for the LLL-Reduced bases, if the indexed vector conflicts with Size-Reduced condition in line 4, the procedure **DECREASE()** will be invoked in line 5. Line 6 is one of the key processes. The algorithm identifies whether the Lovász Condition (5.2.1b) is fulfilled at each 2-by-2 submatrix (r_{k-1}, r_k) along the matrix R's diagonal at line 6. If the condition (5.2.1b) is not satisfied, it calls **SWAPRESTORE**(k) to swap the two vectors and restore R to an upper triangular form. Then the **while** loop restarts from the previous k, or 2 if k equals 2. The unimodular matrix Z will be produced at the end of whole algorithm, such that BZ forms an LLL-Reduced basis.

5.3 Approximate Optimal Reduced Algorithm

We have seen in the previous section that the LLL Reduction algorithm is an excellent algorithm. It gives us a upper bound with the factor $2^{(n-1)/2}$ for an approximation of a shortest vector in polynomial time. The LLL algorithm can also produce a quasi orthogonal basis that can be used to accelerate the performance of other SVP solving algorithms. However, for many high precision required cases, the basis produced by the

LLL algorithm is not good enough. At the same time, the Minkowski reduced basis is not so easily to be constructed, besides the Minkowski reduced basis may not exist, see the counter **Example-**(5.1.3).

Recently, F. Luk and S. Qiao[14] defined a new lattice reduced basis named *Optimally-Reduced* bases that combines the advantages of Minkowski reduced bases, and most importantly guarantees existence. An algorithm that constructs approximate Optimal-Reduced bases is also given at the same time.

5.3.1 Bases Definitions

Definition 5.3.1 (**SubLattice**[7]). Let L be a lattice in $\mathbb{R}^n$. We say M is a *SubLattice* of L, if M is a lattice and is also a subset of L.

Suppose W is a generator matrix for the sublattice M, then even the number of columns in W is less than n, the sublattice M generated by the columns of W is still a lattice in $\mathbb{R}^n$. Clearly, the sublattices of L are also subgroups of L and L is a sublattice of itself. If the dimension of a sublattice M equals to the dimension of L, we say that M is a *full-rank* sublattice of L.

Definition 5.3.2 (**Optimally-Reduced**[14]). Given a lattice L, a generator matrix $B = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n]$ of L is called *Optimally-Reduced*, if for each $\mathbf{b}_i$ ($i = 1, 2, \ldots, n$), its Euclidean Length $\|\mathbf{b}_i\|_2 = \min(\|\hat{\mathbf{b}}_i\|_2, \|\hat{\mathbf{b}}_{i+1}\|_2, \ldots, \|\hat{\mathbf{b}}_n\|_2)$ over all sets $\{ \hat{\mathbf{b}}_i, \hat{\mathbf{b}}_{i+1}, \ldots, \hat{\mathbf{b}}_n \}$ of lattice points such that the matrix $[\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{i-1}, \hat{\mathbf{b}}_i, \hat{\mathbf{b}}_{i+1}, \ldots, \hat{\mathbf{b}}_n]$ generates the given lattice L.

The definition of Optimal-Reduced bases is similar with the definition of Minkowski-Reduced bases. The main difference between them is that the Optimally-Reduced bases

choose only those shortest lattice vectors which can generate the original lattice if they are combined together as a basis. Therefore, the Optimal-Reduced basis is always exists, but Minkowski-Reduced bases do not always exist. It is not hard to see that,

$$\|\mathbf{b}_i\|_2 \geq \|\lambda_i\|_2, \quad (1 \leq i \leq n) \tag{5.3.1}$$

Hence, if there is a Minkowski-Reduced basis, it must be an Optimally-Reduced one as well. In term of sublattice, each $\mathbf{b}_i$ is a shortest vector in the *sublattice* generated by the vectors $\{\mathbf{b}_i, \mathbf{b}_{i+1}, \ldots, \mathbf{b}_n\}$ grouped as a basis.

5.3.2 Approximate Optimally Reduced Algorithm

In Kannan's SVP solving algorithm[25, 26], Kannan introduces a method named **SELECTBASIS()** for bases transformations. This method inserts a target vector $\mathbf{v}$ into the current basis B to create a set of $n+1$ vectors which includes $\mathbf{v}$ and n other vectors of B. Then the method **SELECTBASIS()** transforms this set to a basis of n vectors that generates the original lattice. The bases transformation works as finding the fractional coefficients for an equation system of n unknowns. Unfortunately, the method is unpractical when being implemented in modern computers[41]. Because modern computers all operate floating-point numbers, which introduces errors during floating-point arithmetics. We will substitute the method **SELECTBASIS()** of Kannan's algorithm with a novel *Plane Unimodular Transformation*[14] algorithm, which will solve the same problem in *Optimal Reduction* algorithms in this thesis. The approach of this transformation uses the *Extended Euclidean Algorithm.*

Suppose we are given an integer vector $\mathbf{z} = [p \;\; q]^T$ with $\gcd(p, q) = d$. Then according to the *Extended Euclidean Algorithm,* two integrals (a, b) can be found such that $ap + bq = d$. Hence, we can construct an integer nonsingular matrix M (assuming $d \neq 0$) with those integers, that is:

$$M = \begin{bmatrix} p/d & -b \\ q/d & a \end{bmatrix}.$$

Since M is an integer nonsingular matrix, M is invertible. The inverse of matrix M is:

$$M^{-1} = \begin{bmatrix} a & b \\ -q/d & p/d \end{bmatrix}.$$

We can see that M^{-1} is an integer matrix as well , and M^{-1} satisfies:

$$M^{-1}\mathbf{z} = [d \;\; 0]^T.$$

Since both M and M^{-1} are integer matrices, M is an unimodular matrix. The construction process of the unimodular matrix M still holds if $q = 0$ or $p = 0$, in which case we have $\gcd(p, 0) = p$ or $\gcd(0, q) = q$, respectively. A special case that must be mentioned is $p = q = 0$, then $\gcd(p, q) = 0$. If this happens, we set M to a 2-dimensional identity matrix in order to maintain the properties that M is an unimodular matrix and $M^{-1}\mathbf{z} = [d \;\; 0]^T$ ($d = 0$ in this case). The algorithm of constructing M gives us the following fact.

Fact 5.3.3 (Algorithm UNIM2 (p, q)). *Let $\mathbf{z} = [p \;\; q]^T$ be a nonzero integer vector and* $\gcd(p, q) = d$. *An unimodular matrix M as above can be constructed, such that $M^{-1}\mathbf{z} =$* $[d \;\; 0]^T$.

The algorithm **UNIM2** (p, q) can be applied to expand a vector $\mathbf{b}_1$ into a basis A just like what the Kannan's method **SELECTBASIS()** does. Suppose A is an integer matrix that

its columns generate a 2-dimensional lattice L. Let the integer vector $\mathbf{z} = [p \;\; q]^T$ be the coefficients of the lattice point $\mathbf{b}_1$, such that $A\mathbf{z} = \mathbf{b}_1$. However, not every vectors in the lattice L can be expanded to a basis. The vector $\mathbf{b}_1$ is expandable to a basis, if and only if the greatest common dividers of the entries of the coefficients vector $\mathbf{z}$ is ± 1[25], which means $\gcd(p, q) = \pm 1$.

Therefore,

$$\begin{aligned} & M^{-1} \times \mathbf{z} = [\pm 1 \;\; 0]^T \quad (A \times \mathbf{z} = \mathbf{b}_1) \\ \Longrightarrow\ & \mathbf{z} = M \times [\pm 1 \;\; 0]^T \quad (A \times \mathbf{z} = \mathbf{b}_1) \\ \Longrightarrow\ & A \times \underbrace{M}_{\text{Unimodular}} \times [\pm 1 \;\; 0]^T = \mathbf{b}_1 \end{aligned} \tag{5.3.2}$$

For an n dimensional lattice L and a coefficient vector $\mathbf{z} = [z_1, \; z_2, \; \ldots \; , \; z_n]$ that includes more than two entries, we can use the case of *Expanded Euclidean Algorithm* of more than two numbers[21].

Which is,

$$\gcd(z_1, z_2, \ldots, z_n) = \gcd(z_1, \gcd(z_2, \cdots, \gcd(z_{n-1}, z_n) \cdots)). \tag{5.3.3}$$

In the case of n ($n \geq 2$) entries, we apply the algorithm **UNIM2()** repeartedly from the last coefficient z_n to the first one z_1. During this transformation, the given generator matrix A is transformed into a new generator matrix B, whose first column is the vector $\mathbf{b}_1$ represented by the coefficient vector $\mathbf{z}$.

Similarly, if there are n such equations $A \times \mathbf{z}_i = \mathbf{b}_i$ ($1 \leq i \leq n$), and each $\mathbf{z}_i$ satisfies $\gcd(z_{i,1}, \cdots, z_{i,j}, \cdots, z_{i,n}) = \pm 1$, for $1 \leq j \leq n$. By applying the above transform of n ($n \geq 2$) entries with algorithm **UNIM2()** vector by vector, we can find a sequence of unimodular matrices M_i. Each M_i transforms the current generator matrix into a new generator

matrix, whose ith column is the vector $\mathbf{b}_i$. The matrix A then can be transformed to another generator matrix $B = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n]$. Let M be the product of M_i, that is,

$$M = \prod_{i=1}^{n} M_i.$$

Then M is also an unimodular matrix. According to the **Proposition**-(2.1.3), B must be a generator matrix of the lattice generated by the columns of the matrix A.

Given a lattice L and its generator matrix $B = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n]$. Let Q and R be the matrices of the QR-Decomposition-(5.1.1) of the given matrix B. Denote $L(B_k)$ the sublattice generated by the column vectors of the submatrix $B_k = [\mathbf{b}_k, \mathbf{b}_{k+1}, \ldots, \mathbf{b}_n]$. Let R be the matrix $[\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_n]$. Denote R_k, a submatrix of R, the matrix $[\mathbf{r}_k, \mathbf{r}_{k+1}, \ldots, \mathbf{r}_n]$. Let $\mathbf{v} = R_k\mathbf{z}$ be a vector in the sublattice $L(B_k)$. Using the algorithm **UNIM2()** repeatedly, the following procedure **TRANSFORM**$(k, \mathbf{z})$ transforms the generator matrix B_k into a new generator matrix $\bar{B}_k$, which includes the vector $\mathbf{v}$ as the first column, such that $L(B_k) = L(\bar{B}_k)$.

The **for** loop from line 1 to line 10 applies the Expanded Euclidean algorithm-(5.3.3) iteratively. The matrix Z_j constructed at line 5 protects the first $k-1$ vectors of the given generator matrix B, such that they will never be touched in the procedure. Line 4, 6 and 7 synchronize the coefficient vector $\mathbf{z}$ with the matrices of QR-Decomposition of the current matrix B. Therefore, the vector $\mathbf{b}_k$ in B becomes $R_k\mathbf{z}$ when $\mathbf{z}$ is unimmed to the identical vector at the end of the procedure.

Line 8 requires us to find a *plane reflection matrix* Q_j which restores the destroyed upper triangular structure of R at line 6. However, the functionality of matrix Q_j is not as simple as only recovering the upper triangular structure of the matrix R, it has to maintain the properties of Q as an *orthogonormal* matrix as well.

Procedure Transform(k, **z**)

Input : R, Z, Q and **z**

Output: Modified R, Z and Q

1 **for** $j \leftarrow n-k+1$ ***downto*** 2 **do**
2 $M_j = \textbf{UNIM2}(z_{j-1}, z_j)$;
3 $U_j \leftarrow \text{diag}(I_{j-2}, M_j, I_{n-j-k+1})$;
4 $\mathbf{z} \leftarrow U_j^{-1}\mathbf{z}$;
5 $Z_j \leftarrow \text{diag}(I_{k-1}, U_j)$;
6 $R \leftarrow RZ_j$;
7 $Z \leftarrow ZZ_j$;
8 Call **PLANEREFLECTION**() to find a plane reflection Q_j that restores the structure of R ;
9 $R \leftarrow Q_j R$;
10 $Q \leftarrow QQ_j$;
11 **return** modified R, Z and Q

Fact 5.3.4 (Procedure **PLANEREFLECTION**($R_{2\times 2}$)[31]). *The symmetric matrix $Q_j \in R^{n\times n}$ denotes a Plane Reflection matrix in the $(j-1, j)$ plane, where $2 \le j \le n$. It has the following form:*

$$Q_j = \begin{bmatrix} I_{j-2} & & & \\ & c & s & \\ & s & -c & \\ & & & I_{n-j} \end{bmatrix} \tag{5.3.4}$$

where $c^2 + s^2 = 1$.

The purpose of the procedure **PLANEREFLECTION**($R_{2\times 2}$) becomes clear, when we know the function of the constructed matrix Q_j is to convert the production of two matrices into an upper triangular matrix, that is:

$$\begin{bmatrix} c & s \\ s & -c \end{bmatrix} \times \begin{bmatrix} r_{j-1,j-1} & r_{j-1,j} \\ r_{j,j-1} & r_{j,j} \end{bmatrix} = \begin{bmatrix} \hat{r}_{j-1,j-1} & \hat{r}_{j-1,j} \\ 0 & \hat{r}_{j,j} \end{bmatrix}.$$

This problem can be easily solved by choosing s and c, such that $r_{j-1,j-1} \times c + r_{j,j-1} \times s = 0$. If the element $r_{j,j-1}$ happens to be 0, the plane reflection Q_j can just be set as an n dimensional identity matrix.

Another procedure **BLOCKDECREASE**(i, j) is involved in the forthcoming approximate Optimal-Reduction algorithm. It checks the the Size-Reduced condition-(5.2.1a) on the submatrix $[\mathbf{b}_k, \mathbf{b}_{k+1}, \ldots, \mathbf{b}_n]$, and updates the submatrix to make sure the condition (5.2.1a) is always fulfilled in the new constructed generator matrix. The procedure **DECREASE**(i, j) invoked in **BLOCKDECREASE**(i, j) is identical with the one that has been introduced in the chapter "LLL Reduction Algorithm".

Procedure BlockDecrease(k)

Input : R, Z

Output: Modified R, Z

```
1 for i ← n − 1 downto 1 do
2 |   for j ← n downto Max (i + 1, k) do
3 |   |   DECREASE(i, j) ;
4 return modified R, Z
```

Taking advantages of the two procedures above, we can finally give an approximate *Optimally-Reduced Algorithm* (we will state why it is an approximation algorithm in the next section).

Theorem 5.3.5 (**APPROXIMATE OPTIMALLY-REDUCED ALGORITHM**). *Given a lattice L and its generator matrix B, suppose the matrices Q and R are the outputted matrices of QR-Decompositions*-(5.1.1) *of B. The following algorithm constructs an unimodular matrix*

Z for the matrix B, such that BZ forms an approximation of an Optimally-Reduced basis that generates the original lattice L.

Algorithm 5: APPROXIMATE OPTIMAL REDUCTION Algorithm

Input : A basis B of a given lattlce L

Output: An unimodular matrix Z that transforms an approximate Optimally-Reduced basis

1 Initial QR-Decomposition, $B = QR$ and $Z \leftarrow I_n$;
2 **BLOCKDECREASE**(1) ;
3 $k \leftarrow 1$;
4 **while** $k < n$ **do**
5 Set $R_k \leftarrow [\mathbf{r}_k, \mathbf{r}_{k+1}, \ldots, \mathbf{r}_n]$;
6 Call algorithm-(3) **ENUMERATE**() to Find a vector $\mathbf{z} \in \mathbb{Z}^{n-k+1}$, such that $R_k\mathbf{z}$ forms a shortest nonzero vector in the sublattice generated by the columns of R_k ;
7 **TRANSFORM**$(k, \mathbf{z})$;
8 **BLOCKDECREASE**(k) ;
9 Search the smallest p such that $\|\mathbf{r}_p\|_2 > \|\mathbf{r}_q\|_2$, where $1 \le p \le k < q \le n$ and $\|\mathbf{r}_q\|_2 = \mathtt{Min}\,(\|\mathbf{r}_j\|_2)$, $k < j \le n$;
10 **if** $p \le k$ **then**
11 $k = p$;
12 **else**
13 $k = k + 1$;
14 **return** the matrix Z ;

Let $B_k = [\mathbf{b}_k, \mathbf{b}_{k+1}, \ldots, \mathbf{b}_n]$ be the submatrix of the generator matrix B without the first $k-1$ columns, and denotes $\bar{B}_k = [\mathbf{v}, \bar{\mathbf{b}}_{k+1}, \ldots, \bar{\mathbf{b}}_n]$ and $\bar{B} = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{k-1}, \bar{B}_k]$ be the modified submatrix B_k and the matrix B after calling the procedure **TRANSFORM**$(k, \mathbf{z})$, respectively. In the algorithm **APPROXIMATE OPTIMAL REDUCTION**, the **while** loop from line 4 to 13 works iteratively in every sublattice $L_k(B_k)$ generated by columns of B_k, which is created in line 5. An unimodular transformation matrix Z of an approximate

Optimally-Reduced basis will be constructed in the end of the **while** loop.

Line 6 is a key process of the algorithm, which finds a shortest vector $\mathbf{v}$ in the sublattice $L_k(B_k)$. Various external SVP solving and apprSVP solving algorithms can be used to discover the shortest vector $\mathbf{v}$. Sphere decoding methods is recommended in F. Luk and S. Qiaos' original paper[42], we will choose the algorithm **ENUMERATE()**-(3) to discover $\mathbf{v}$, which is plugged into the algorithm at line 6. In the end of line 6, it returns the integer coefficient $\mathbf{z}$ of the shortest vector $\mathbf{v}$ discovered.

After finding the shortest vector $\mathbf{v}$, the algorithm invokes the procedure **TRANSFORM()** to produce a new generator matrix $\bar{B}_k$ for the sublattice $L_k(B_k)$, which stores $\mathbf{v}$ as the first element. The corresponding transformation to the generator matrix B of the lattice L is that the vector $\mathbf{v}$ will be inserted into B as the kth vector, and leaves the first $k-1$ vectors $\{ \mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{k-1} \}$ unchanged. Line 8 checks the new basis with Size-Reduced condition(5.2.1a).

The shortest vectors $\mathbf{v}$ found in each sublattices $L_k(B_k)$ $(1 \leq k \leq n)$ may not be correctly ordered since it is an approximate algorithm. Therefore we have to consider sorting the vectors in the matrix $\bar{B}$ according to the definition of Optimally-Reduced bases (5.3.2). That is, after transforming the kth vector into the basis B, we will simply go through every single vector in $\bar{B}$. If there is a vector located after the kth vector whose length is shorter than a vector $\mathbf{b}_p$, $(p < k)$ which is located before $\mathbf{b}_k$, we restart the **while** loop from the incorrect-ordered index p of the vector $\mathbf{b}_p$.

5.3.3 Termination and Complexity Analysis

We have mentioned that the **APPROXIMATE OPTIMAL REDUCTION** algorithm-(5) can only produce an approximation of an Optimally-Reduced basis. Actually, the shortest vector

computed in line 6 is a shortest vector of a particular sublattice, not all possible sublattices. Hence, what the algorithm outputs can be only an approximate of an Optimally-Reduced basis.

The action of "Searching for the smallest p" in line 9 raises the question of termination for the approximate Optimal-Reduction algorithm. The value of the index k in the next loop starts either the integer $k+1$, or the number p that smaller than k. However, the algorithm-(5) guarantees termination. Because every time the index k goes back to a smaller p, the length of the shortest vector found in the sublattice $L_p(B_p)$ is at most the length of the vector that is shorter than $\mathbf{b}_p$. Hence the lengths of vectors found in algorithm-(5) will be shorter and shorter. Since the lower bounds of Euclidean Length for the columns in the generator matrix are Minkowski's Minimas, the algorithm will terminate in a finite step of swaps.

An interesting phenomenon happens if we use LLL as an algorithm to find an approximation of a shortest vector in line 6. Since the **LLL** algorithm-(4) only finds an approximate shortest vector with a factor of $2^{\mathcal{O}(n)}$. the swapping time will have to be exponential. The extreme case of invoking the **LLL** algorithm-(4) will lead the algorithm-(5) to never terminate at all. The reason is when we swap the index k with the index p of the longer column vector, the next shortest vector found in sublattice $L_p(B_p)$ might to remain the same as $\mathbf{b}_p$. This is because the **LLL** algorithm-(4) is an approximate algorithm that cannot guarantee find a vector shorter than $\mathbf{b}_p$. On the other hand, exact SVP solving algorithms like **ENUMERATE()** will reduce the swap times to polynomial time and guarantee termination.

The complexity dominates the algorithm-(5) is the SVP solving algorithm at line 6. The call of **ENUMERATE()** algorithm-(3) will cost $2^{\mathcal{O}(n^2)}$ composed operations[45], and

the complexity of each composed operation is a fixed polynomial of n. Other sphere decoding algorithms can reduce the complexity of the algorithm-(5) a little, for example, if we use Kannan's algorithm to discover a shortest vector, the worst-case complexity would be $n^{\frac{n}{2e}+o(n)} \approx n^{0.184n+o(n)}$[20].

5.4 Optimally Reduced Algorithm

The previous section has introduced an approximate Optimally-Reduction algorithm-(5), which works as finding a shortest vector in each sublattices and combining them together to construct a new basis. The procedure **BLOCKDECREASE()** involved the vectors of the basis that before the cursor, such that the sublattices are changed. However, The procedure **BLOCKDECREASE()** cannot be ignored because it is necessary for checking the Size-Reduced condition. Hence the algorithm-(5) can only produce an approximate Optimally-Reduced basis. Recently, Wen Zhang, Sanzheng Qiao, and Yimin Wei[48] present a novel algorithm that constructs an exact Optimally-Reduced basis.

Instead of searching a shortest vectors in particular sublattices[14], the new Optimally-Reduced algorithm discovers shortest vectors in all possible subspaces. The following lemma[6, 48] builds the theoretical foundation for the new Optimally-Reduced algorithm we are going to introduce.

Lemma 5.4.1. *Let* $B = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n] \in \mathbb{R}^{m\times n}$ *be an integer full-column rank matrix and* L *be the lattice generated by columns of* B. *For a vector* $\mathbf{v} = \sum_{i=1}^{n} z_i\mathbf{b_i}$, $(z_i \in \mathbb{Z})$, *there exists a basis for* L *containing* $\{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{p-1}, \mathbf{v}\}$ *if and only if:*

$$\gcd(z_p, z_{p+1}, \ldots, z_n) = 1.$$

Proof. See [6, 48]. □

The **Lemma** -(5.4.1) points out a very important property for candidate vectors of a basis in the lattice L, that if the vector can be inserted into the given generator matrix B as the pth column, the greatest common divider of the coefficients of the vectors $\{ \mathbf{b}_p, \mathbf{b}_{p+1}, \ldots, \mathbf{b}_n \}$ has to be 1. According to the **Lemma**-(5.4.1), we can construct a basis by searching all the vectors that fulfil the above property, and at the same time, we have to make those vectors as short as possible.

5.4.1 Sphere Decoding Algorithm

In the approximate Optimally-Reduced algorithm-(5) proposed in the previous section, we use the **ENUMERATE()** algorithm-(3) to discover a shortest vector in a *sublattice.* In order to identify the searching ranges, the **ENUMERATE()** algorithm-(3) projects a vector recursively to the orthogonal complement of a hypersphere spanned by a group of vectors. In this new exact Optimally-Reduced algorithm, a minor modified version of sphere decoding method[48] will be introduced which works in *subspaces* to find the vector mentioned in the **Lemma**-(5.4.1).

Let L be an n dimensional lattice generated by the columns of the matrix $B \in \mathbb{R}^{m \times n}$, and let $\mathbf{v} \in \mathbb{R}^m (\mathbf{v} \notin L)$ be a vector to be decoded, the decoding method is to find an integer coefficient vector $\mathbf{z}$ with respect to a vector $\hat{\mathbf{v}} \in L$, such that that coefficients $\mathbf{z}$ minimizes the Euclidean Length from the vector $\hat{\mathbf{v}}$ to the vector $\mathbf{v}$, that is:

$$\textbf{MIN}\{ \|B\mathbf{z} - \mathbf{v}\|_2 \mid \forall \mathbf{z} \in \mathbb{Z}^n \}. \tag{5.4.1}$$

Various algorithms are proposed to solve the sphere decoding problem, such as the Exhaustive Search algorithm **ENUMERATE()**-(3), which is based on Kannan's strategy[25,

26], the Sieving algorithm based on Ajtai's strategy[2], and so on. The general idea of sphere decoding algorithms is to identify a range that includes $B\mathbf{z}$ and exhaustively search all points within this range. Hence the closest vector will be discovered.

Differing from the **ENUMERATE()** algorithm-(3) which works in sublattices, the new decoding algorithm we will describe searches the points and constructs the new generator matrix in *subspaces*. The given lattice generator matrix B of the lattice L in equation-(5.4.1) can be rewritten as:

$$B = [B_{m\times(n-1)},\ \mathbf{b}_n] \tag{5.4.2}$$

where $\mathbf{b}_n$ is the last column of the matrix B and $B_{m\times(n-1)}$ is an m-by-$(n-1)$ submatrix including the first $n-1$ vectors of B. By splitting the matrix B in (5.4.2), the lattice L can be then separated into a sequence of sublattices indexed by listing all possible coefficients to the vector $\mathbf{b}_n$. Each sublattice has the same dimension $(n-1)$. The decomposition of the lattice L can be represented as:

$$L(B) = \bigcup_{u_n=-\infty}^{+\infty} \{\, \mathbf{w} + u_n\mathbf{b}_n \mid \mathbf{w} \in L(B_{m\times(n-1)}),\ u_n \in \mathbb{Z} \,\} \tag{5.4.3}$$

The parameter u_n is called the *index* of the sublattice $L(B_{m\times(n-1)})$, because the points of those sublattices shown in equation-(5.4.3) are identical but with different offset $u_n\mathbf{b}_n$. Let Span(B) represent the *Vector Space* spanned by the columns of the matrix B, and define the *subspace* $\mathrm{Span}_{u_n}(B)$ as

$$\mathrm{Span}_{u_n}(B) = \{\, \mathbf{w} + u_n\mathbf{b}_n(n) \mid \mathbf{w} \in \mathrm{Span}(B_{m\times(n-1)}) \,\} \tag{5.4.4}$$

where $\mathbf{b}_n(n)$ is the projection of $\mathbf{b}_n$ on the orthogonal complement of the vector

space $\text{Span}(B_{m\times(n-1)})$ according to the **Formula**-(4.2.3), see the Chapter "Lattice Projection" for more details. Therefore, the orthogonal distance between the vector $\mathbf{v}$ of the equation-(5.4.1) to the subspace $\text{Span}_{u_n}(B)$ is

$$d(\mathbf{v}, \text{Span}_{u_n}(B)) = |u_n - \hat{u}_n| \cdot \|\mathbf{b}_n(n)\|_2, \quad \text{where } \hat{u}_n = \frac{\langle \mathbf{v}, \mathbf{b}_n(n)\rangle}{\|\mathbf{b}_n(n)\|_2^2} \tag{5.4.5}$$

If $\hat{\mathbf{v}}$, the closest vector of $\mathbf{v}$, lies in a certain subspace $\text{Span}_{u_t}(B)$ with an index u_t, and if the upper bound ρ_n, such that $\|\mathbf{v} - \hat{\mathbf{v}}\| \leq \rho_n$ is given, then the orthogonal distance between $\mathbf{v}$ to the subspace $\text{Span}_{u_t}(B)$ must be not longer than the orthogonal distance between $\mathbf{v}$ and $\hat{\mathbf{v}}$, which means

$$d(\mathbf{v}, \text{Span}_{u_t}(B)) \leq \rho_n.$$

Combing the above inequation of the given distance upper bound ρ_n with the equation (5.4.5), we can calculate the range of the index u_t of the subspace $\text{Span}_{u_t}(B)$ in which the closest vector $\hat{\mathbf{v}}$ lies, that is:

$$\hat{u}_t - \frac{\rho_n}{\|\mathbf{b}_n(n)\|_2} \leq u_t \leq \hat{u}_t + \frac{\rho_n}{\|\mathbf{b}_n(n)\|_2}, \quad (u_t \in \mathbb{Z}) \tag{5.4.6}$$

By enumerating each integer u_t within the range of above equation-(5.4.6) and setting $\rho_{n-1} = \sqrt{\rho_n^2 - (\|u_t \times \mathbf{b_n}(n)\|_2)^2}$, we have successfully transformed the original closest vector problem in the n dimensional lattice $L(B)$ to $\lfloor \frac{\rho_n}{\|\mathbf{b}_n(n)\|_2} \rfloor$ closest vector problems of one dimension less lattice $L(B_{m\times(n-1)})$, where $\lfloor x \rfloor$ represents the biggest integer z, such that $z \leq x$. The problem (5.4.1) will be solved, if we recursively apply the above process until the dimension of subspaces reaches 1,

Summing up all the equations and processes above, the sphere-decoding algorithm **M-DECODE**(R, p) can finally be introduced to find the pth vector mentioned in the

Lemma-(5.4.1). This algorithm also solves the integer least square problem-(5.4.1). To start the decoding process, the equation-(5.4.6) requires an initial distance ρ_n, which will be set as the length of the pth vector $\mathbf{b}_p$ of the current generator matrix B of the lattice L. Since at least we can guarantee that $\mathbf{b}_p$ can be found as a candidate, which makes the equation $\gcd(z_p, z_{p+1}, \ldots, z_n) = 1$ satisfied. Therefore, the algorithm never fails. The input parameter R is the upper triangular matrix of the QR-Decomposition-(5.1.1), p is the index of the vector we are looking for.

Algorithm 6: M-DECODE

input : R, p

output: An integer coefficient vector $\mathbf{z}$, which minimizes $\|R\mathbf{z}\|_2$ and $\gcd(z_p, z_{p+1}, \ldots, z_n) = 1$

1 Run **LLL** algorithm-(4) on R ;
2 Set the initial size $\rho \leftarrow \|R(:, p)\|_2$ as the upper bound of searching region;
3 Call M-Search-I(R, ρ, p) to get an integer coordinate $\mathbf{z}_{min}$;
4 **return** the $\mathbf{z}_{min}$;

The algorithm **M-DECODE()** invokes the **LLL** algorithm as a preprocessor to speed up the searching precess at line 1. The upper bound for the length of the target vector is set at line 2. Line 3 is a key statement which calls another method **M-SEARCH-I()** and gives us the integer coefficient $\mathbf{z}_{min}$ of the vector fulfilled conditions mentioned in the **Lemma**-(5.4.1). In the end of algorithm, the $\mathbf{z}_{min}$ returned in line 3 will be returned.

The procedure **M-SEARCH-I()** returns an integer coordinate vector $\mathbf{z}_{min}$, which minimizes $R\mathbf{z}$ for the given index p. Depth-First search is designed as the main approach to enumerate all possible coordinates in subspaces. The **while** loop runs from the last vector of B to the first vector with a cursor variable i. The variable $\mathbf{u}$ records the branches of searching coordinates, which varies in the ranges identified by equation-(5.4.6) for

Procedure M-Search-I

input : R, ρ, p

output: An integer coordinate $\mathbf{z}_{min}$

```
1  Set z_min = [0, ... , 0]^T, u = [0, ... , 0]^T, l = [0, ... , 0]^T ;
2  Set i ← n ;
3  while i ≤ n do
4      Compute l ← Ru ;
5      if ||l||_2 < ρ then
6          Set z = ConvertLLL (u) ;
7          if i = 1 & gcd(z_p, z_{p+1}, ... , z_n) = 1 then
8              Set ρ ← ||l||_2 ;
9              Set z_min ← z ;
10             u_i = u_i + 1 ;
11         else if i > 1 then
12             Set the center b ← b_{1:i-1} − u_i R(1 : i − 1, i) ;
13             Compute ρ_{i-1} = sqrt(ρ^2 − ||l||_2^2) as the upper bound of searching region
               in the subspace ;
14             i ← i − 1 ;
15             Set u_i ← new lower index by the equation-(5.4.6) ;
16     else
17         i ← i + 1 ;
18         u_i = u_i + 1 ;
19 return the z_min ;
```

each index i. However, the equation-(5.4.6) is designed to solve the CVP for a given $\mathbf{v}$. Hence in our method, we can set the parameter $\mathbf{v}$ to be $\mathbf{0}$ as the initial vector in order to apply equation-(5.4.6) to start the algorithm. In terms of the length of the vector with current coordinates $\mathbf{u}$, the variable $\mathbf{l}$ will store the current vector computed from $R\mathbf{u}$ and its length can be calculated by the Euclidean Norm formula.

The upper bound ρ for the length of the pth vector is reduced within the procedure whenever it is possible. Notice that the matrix R sent to **M-SEARCH-I()** is not the original R right after the QR-Decomposion algorithm, because the LLL algorithm has been applied to the matrix R in the algorithm **M-DECODE()**. Therefore, the method $\gcd(\cdots)$ cannot be used directly to the coordinates $\mathbf{u}$, which has to be converted back to another variable $\mathbf{z}$ with an LLL transformation function in line 6. If the length of the current vector is less than the upper bound ρ and the condition equation $\gcd(z_p, z_{p+1}, \ldots, z_n) = 1$ is satisfied at the same time, this vector will become a candidate of the target vector we are looking for. Hence the parameter ρ will be updated in line 8 and the coefficient vector $\mathbf{z}$ will be recorded as $\mathbf{z}_{min}$.

Line 12 and 13 are actually compute the upper bound ρ_{n-1} for the equation-(5.4.6) of the next Depth-First-Search level. The variable $\mathbf{l}$ stores the coordinates of current vector, which is the computed result of $R\mathbf{u}$, and $\mathbf{l}$ is also referred as the offset of the upper bound ρ_n in equation-(5.4.6) during iterative branches, that is $\rho_{i-1} = \sqrt{\rho^2 - \|\mathbf{l}\|_2^2}$ in line 13. In the end of procedure **M-SEARCH-I()**, the integer coefficient vector $\mathbf{z}_{min}$ corresponding to the parameter index p is returned, and $R\mathbf{z}_{min}$ is minimized among all possible integral coefficients.

5.4.2 Optimally-Reduced Algorithm

Having the theoretical foundation **LEMMA**-(5.4.1) and the sphere decoding algorithm **M-DECODE()**-(6), constructing an Optimally-Reduced basis is straightforward[48]. Given a lattice L of arbitrary dimension n generated by the columns of a generator matrix B, there are two main problems need to be solved in order to construct an Optimally-Reduced generator matrix M, they are:

1. Find n independent vectors that fulfil the conditions required in **Lemma**-(5.4.1).

2. Combine the n vectors found in step one to construct an Optimally-Reduced basis.

By the sphere decoding algorithm-(6) **M-DECODE()**, the pth vector candidate in the matrix M can be discovered without any difficulties. Hence, the problem-1 can be solved by applying **M-DECODE()** with the cursor p varies from 1 to n.

However, solving the second problem to form an Optimally-Reduced basis is not as simple as putting the n vectors found in step one together. Recall the procedure **TRANSFORM**(k,**z**) in the **APPROXIMATE OPTIMALLY REDUCTION** algorithm-(5), which is invoked to transform the given generator matrix into a new generator matrix that includes the shortest vector discovered as a column. Meanwhile, we have to maintain the properties of the lattice generated by the new constructed basis, such that it will be the same as the original given lattice. We will borrow this idea from the **APPROXIMATE OPTIMALLY REDUCTION** algorithm-(5) to transform vectors found in step one into the given generator matrix to construct a new Optimally-Reduced basis.

Since the algorithm **ENUMERATE()** embedded in the **APPROXIMATE OPTIMALLY REDUCTION** algorithm-(5) works in sublattices, but the algorithm **M-DECODE()** finds the pth vector in the original lattice L all the time, we can see the difference between the two

vectors, $\bar{\mathbf{v}}_p$ found in the algorithm **ENUMERATE()** and $\mathbf{v}_p$ discovered in the algorithm **M-DECODE()**, is that $\bar{\mathbf{v}}_p$ involves only the columns of the generator matrix with index larger than p, yet $\mathbf{v}_p$ can be linear combination of all columns of the matrix.

More precisely, let $B_p = [\mathbf{m}_1, \ldots, \mathbf{m}_{p-1}, \mathbf{b}'_p, \ldots, \mathbf{b}'_n]$ be the currently constructed generator matrix of index p in process. Denote $\mathbf{m}_p$ the pth Optimally Reduced vector discovered in the algorithm **M-DECODE()** with coefficients $\mathbf{z} = [z_1, z_2, \ldots, z_n]^T$. Therefore,

$$\mathbf{m}_p = B_p \mathbf{z}.$$

According to the **LEMMA**-(5.4.1), the vector $\mathbf{m}_p$ satisfies the following two conditions:

$$\mathbf{m}_p = \underbrace{z_1\mathbf{m}_1 + \cdots + z_{p-1}\mathbf{m}_{p-1}}_{\mathbf{m}_{part1}} + \underbrace{z_p\mathbf{b}'_p + \cdots + z_n\mathbf{b}'_n}_{\mathbf{m}_{part2}} \tag{5.4.7a}$$

$$\gcd(z_p, z_{p+1}, \ldots, z_n) = 1 \tag{5.4.7b}$$

Out goal is to find an unimodular matrix Z, such that

$$B_{p+1} = B_p Z$$

where the pth column of the matrix B_{p+1} is $\mathbf{m}_p$, and the first $p-1$ columns of B_{p+1} and B_p are identical.

Therefore, the pth column of the unimodular matrix Z has to be the coefficient vector $\mathbf{z}$, and the first $p-1$ columns of Z must be unit vectors. Thus after multiplying B_p with the matrix Z, the pth vector in B_{p+1} will be $\mathbf{m}_p$, and leaves the first $p-1$ columns of B_p unchanged.

We will construct two unimodular matrices Z_1 and Z_2, such that $Z = Z_1 Z_2$[48], by:

$$Z_1 = \begin{bmatrix} I_{p-1} & 0 \\ 0 & M_1 \end{bmatrix}, \quad Z_2 = \begin{bmatrix} I_{p-1} & M_2 \\ 0 & I_{n-p+1} \end{bmatrix} \tag{5.4.8}$$

where M_1 of Z_1 is the unimodular matrix, which is produced by applying the procedure **TRANSFORM**$(p, [z_p, z_{p+1}, \ldots, z_n]^T)$, whose first column is $[z_p, z_{p+1}, \ldots, z_n]^T$, and hence Z_1 is an unimodular matrix. See Chapter "Approximate Optimally-Reduced Algorithm" or [14, 48] for more details. M_2 is composed by forcing $[z_1, z_2, \ldots, z_{p-1}]^T$ as the first column and **0**s for all other vectors. From the structure of Z_2, it is an unimodular matrix.

For the condition (5.4.7a), we can see that $\mathbf{m}_{part1}$ and $\mathbf{m}_{part2}$ are the products of B_p with the corresponding columns of Z_1 and Z_2, respectively. Thus $\mathbf{m}_p$ is the pth column of the multiplication of $B_p Z_1 Z_2$. After that, $B_{p+1} = B_p Z$ is obtained.

Theorem 5.4.2 (OPTIMALLY-REDUCED ALGORITHM). *Given an integer full-column rank matrix $B \in \mathbb{R}^{m \times n}, (m \geq n)$, and a lattice L generated by the columns of B, let the matrices Q and R be the matrices of QR-Decompositions-(5.1.1). The following algorithm constructs an unimodular matrix Z, such that BZ forms an Optimally-Reduced basis whose columns generate the original lattice L.*

The cursor p for the new generator matrix of the **OPTIMAL REDUCTION** algorithm-(7) is set in line 3. The algorithm terminates when p arrives at n. Whenever a new coefficient vector $\mathbf{z}$ for the pth vector is found in line 4, the algorithm transforms the currrent generator matrix into a new generator matrix that includes the vector corresponding to the coordinates $\mathbf{z}$ as the pth column. Line 6 applies the matrix Z_2 of equation-(5.4.8) to R and line 7 synchronizes the process with the unimodular matrix Z, therefore in the

Algorithm 7: **OPTIMAL REDUCTION** algorithm

Input : A generator matrix B of a given lattlce L

Output: An unimodular matrix Z, such that BZ forms an Optimally-Reduced basis

```
1 Initial QR Decomposition, such that B = QR ;
2 Set Z ← I_n ;
3 for p ← 1 to n do
4 |   Call M-DECODE(p) to Find a vector z fulfilled Lemma-(5.4.1) ;
5 |   TRANSFORM (p, [z_p, z_{p+1}, ... , z_n]^T) ;
6 |   Set R ← RZ_2 ;
7 |   Set Z ← ZZ_2 ;
8 return the matrix Z ;
```

end of the algorithm, the new basis will be the multiplication of B and the unimodular matrix Z.

Proof. See [48] for the full proof. □

Chapter 6

Experimental Results

We have presented the GGH cryptosystem in chapter 3, and we also introduced the lattice reduction algorithms that can be applied to attack the GGH cryptosystem in chapter 5.

They are:

1. The **LLL REDUCTION** algorithm-(4);
2. The **APPROXIMATE OPTIMAL REDUCTION** algorithm-(5);
3. The **OPTIMAL REDUCTION** algorithm-(7).

In this chapter, we will show the results of two parts experiments by applying the three lattice reduction algorithms. The first part of the experiments focuses on the qualities of the bases produced by the algorithms, and the second part presents the ciphertexts decrypted by those bases in the first part.

In the first part of this chapter, we show the experimental environment, including the hardware, the software and the programming tools we used. Then the detailed organizations of experiment steps will be introduced. The second part compares the qualities

of the bases produced by the three algorithms in terms of the Hadamard Ratio, in which cases, the dimension of the GGH cryptosystem varies from 2 to 20. In the last part, we present the plaintexts decrypted by the bases produced by the algorithms with missing ratios.

6.1 Experimental Environments

We choose Ubuntu as the main Operating System, which is a branch of Linux operating system based on the Debian GNU/Linux distribution . The three lattice reduction algorithms are implemented using the 32-Bits version MATLAB 2010b.

The details of experimental environments are shown in the **Table**-(6.2) below.

Experimental Environments		
Hardware	CPU	3.0GHz Quad-Core Intel Core i3
	Memory	4GB 1066MHz
	Graphics Card	ATI Radeon HD 5450
	Hard Disc	500GB 7200-RPM
Software	Operating System	Ubuntu 11.04 32-Bits Kernel Linux 2.6.38-8-Generic-Pae
	MATLAB	2010b 32-Bits
Algorithms	The **LLL algorithm**-(4)	
	The **APPROXIMATE OPTIMALLY-REDUCED algorithm**-(5)	
	The **OPTIMALLY-REDUCED algorithm**-(7)	

Table 6.2: The experimental environments

6.2 The Organization of the Experiments

In our experiments, all elements in vectors and bases are **integers**. After implementing the algorithms with Matlab 2010b, our experiments were designed as follows:

1. Determine the dimension of the GGH cryptosystem;

   The dimension of GGH cryptosystems in our experiments was set from 2 to 20, that is, our experiments ran in lattices, whose dimensions varies from 2 to 20.

2. Choose a range of the integers for the entries in the Private Key;

   We chose 4 different ranges I_k, $(1 \leq k \leq 4)$ as the integer intervals for the elements $p_{i,j}$, $(1 \leq i, j \leq n)$ of the Private Key.

   They were:

$$I_k = \begin{cases} -999 \leq p_{i,j} \leq 999, & (k=1) \\ -9{,}999 \leq p_{i,j} \leq 9{,}999, & (k=2) \\ -99{,}999 \leq p_{i,j} \leq 99{,}999, & (k=3) \\ -999{,}999 \leq p_{i,j} \leq 999{,}999, & (k=4) \end{cases}$$

   For the *Ephemeral Key* **r**, each of its coordinates was randomly chosen as an integer uniformly distributed between −99 and 99. The integer range of **r** is fixed, independent of I_k, $(1 \leq k \leq 4)$.

   Compare with the **Example**-(3.2.1) of the GGH Cryptosystem, this perturbation vector **r** was relatively larger than that of the real world. We chose this large **r** because the dimensions of our lattice candidates were relatively small. To enhance the effects of the three lattice reduction algorithms, we had to make the distance

between the lattice point representing encrypted message **e** and the exact closest point **v** as large as possible. We will see from the experimental result that the distance was so large, that sometimes even the Private Key did not decrypt the message **e** successfully.

3. Run the lattice reduction algorithms;

   For each given number n as the dimension of a lattice, we first randomly generated a Private Key V, such that its Hadamard Ratio fulfilled the requirement of the GGH cryptosystem. After this, we constructed the Public Key W based on this Private Key, and its Hadamard Ratio was very small, 1.0×10^{-4} for example. Then the plaintext **m** and the Ephemeral Key **r** were generated randomly. The ciphertext **e** was encrypted by the **Formula**-(3.2.1).

   To make the result in our experiments as accurate as possible, we ran the above process for 100 times in each dimension n.

4. Compare the average Hadamard Ratios of the bases produced;

   We applied the **LLL REDUCTION** algorithm-(4), the **APPROXIMATE OPTIMAL REDUCTION** algorithm-(5), and the **OPTIMAL REDUCTION** algorithm-(7) on the Public Key W. For each algorithm, the Hadamard Ratios of the reduced bases it produced were accumulated together, and the average value of the Hadamard Ratio was shown in the end of the experiments of each dimension n.

   We compared the difference of the average Hadamard Ratios for the three algorithms in various dimension lattices in tables and figures.

5. Compare the decoded message by the bases produced;

The main purpose of experiments was to compare the decrypted message of the three algorithms. We decrypted the ciphertexts using the bases produced by the lattice reduction algorithms, and compared them to illustrate their capabilities of decryption.

6. Output of the experimental results.

   In the end of experiments, the result of average Hadamard Ratios and the missing ratios of decryption for each algorithm were output.

The experimental data will be shown in two parts in this chapter: the average Hadamard Ratio of 20 lattice dimentions, and the times of decrypting the ciphertext successfully.

6.3 Experimental Results of Hadamard Ratio

We have introduced in the chapter 5 that the *Hadamard Ratio* is proposed to measure the qualities of the *Private Key* and the *Public Key*, which are the bases of a lattice that the GGH cryptosystem works on. The Private Key is a good basis for the lattice that has a relatively large Hadamard Ratio, and the Hadamard Ratio of the Public Key is very small.

In this part, we used the Hadamard Ratio to measure the qualities of the bases produced by the three lattice reduction algorithms. The Hadamard Ratios are shown in figures. We attached the detailed experimental results of the Hadamard Ratios in **Appendix A.**

To start the experiments, we chose the dimension of the GGH cryptosystem from 2 to 20, iteratively. After selecting the dimension, the Private Key was created randomly, and the Public Key was generated such that its Hadamard Ratio was less than 1.0×10^{-4}. Then the three lattice reduction algorithms were applied to reduce the Public Key, and

we recorded the Hadamard Ratios of the bases produced algorithm by algorithm. The above process run 100 times for each fixed dimension. Therefore, the Hadamard Ratio shown in figures and table were the average value of 100 samples.

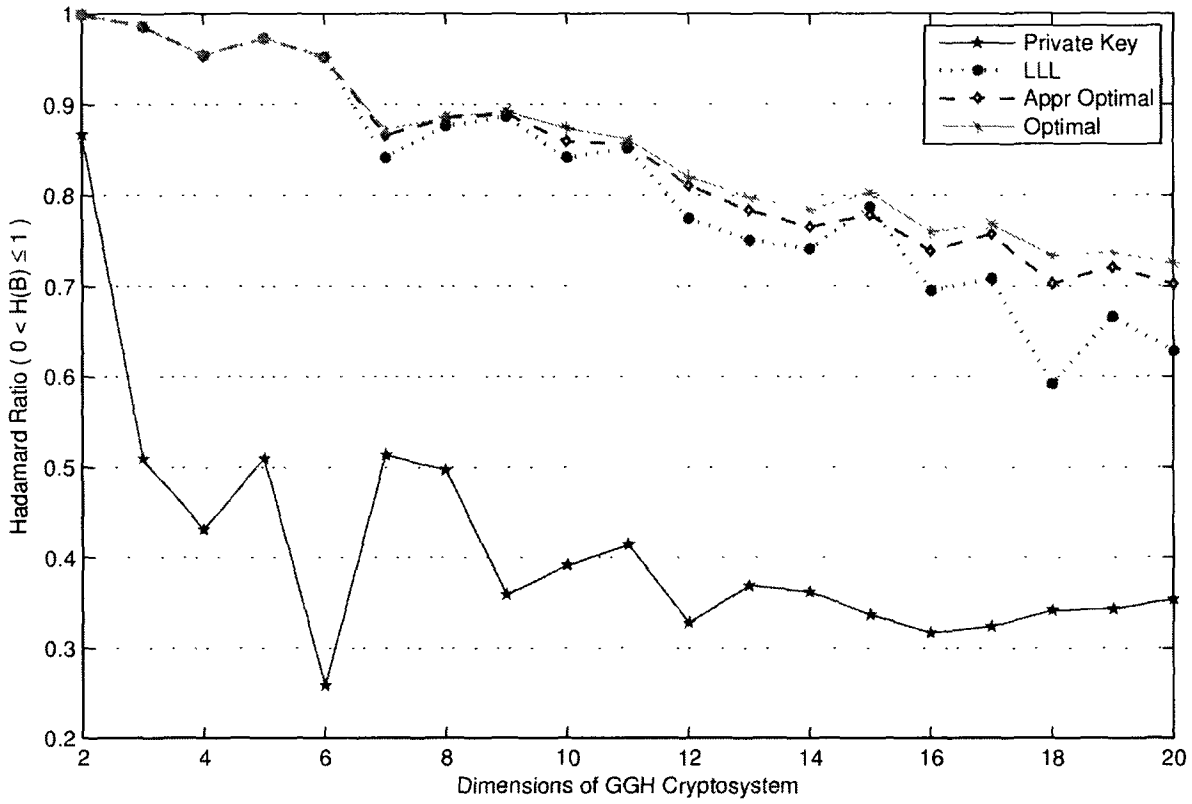

Figure 6.6: Hadamard Ratios of bases of integers within range [−999, 999]

Figure-(6.6) illustrates the average Hadamard Ratios of the bases which were produced by applying the three lattice reduction algorithms on the Public Key respectively. In this case, each element of the Private Key was randomly generated between −999 to 999. The bigger that the Hadamard Ratio was, the better that the corresponding bases were.

We can see in **Figure**-(6.6), the bases produced by the lattice reduction algorithms were better then the Private Key. There was no difference between the bases produced

by the algorithms when the dimension was less and equal than 6. From the dimension 7, the quality of the bases produced by the **OPTIMAL REDUCTION** algorithm-(7) is better than all other bases. The average Hadamard Ratios of bases of dimension 7 were:

$$\mathscr{H}(B) \approx \begin{cases} 0.5135, & \text{(the Private Keys)} \\ 0.8418, & \text{(the LLL-Reduced bases)} \\ 0.8661, & \text{(the Approximate Optimally-Reduced bases)} \\ 0.8722, & \text{(the Optimally-Reduced bases)} \end{cases}$$

This is reasonable because the definition of Optimally-Reduced basis is much stronger than the definition of LLL basis, hence the approximation of an Optimally-Reduced basis is likely better than a LLL-Reduced basis in a high possibility. In most of the GGH cryptosystems whose dimensions is larger than 7, the bases produced by the **APPROXIMATE OPTIMAL REDUCTION** algorithm-(5) were better than the bases created by the **LLL** algorithm-(4). However, in dimension 15, the average Hadamard Ratio of the LLL-Reduced bases was slightly greater than the approximate Optimally-Reduced bases. This fact illustrated that the algorithm-(5) was only an approximation of the Optimal Reduction algorithm, such that it could not guarantee to produce a basis which was better than the one produced by the LLL algorithm.

Figure-(6.7) illustrates the average Hadamard Ratios of the bases produced by the three lattice reduction algorithms. In this case, each integer in the Private Key was randomly generated between $-9,999$ to $9,999$.

Compare with the Hadamard Ratios shown in **Figure**-(6.6), we can see that the Hadamard Ratios of bases generated by the three lattice reduction algorithms were different in the

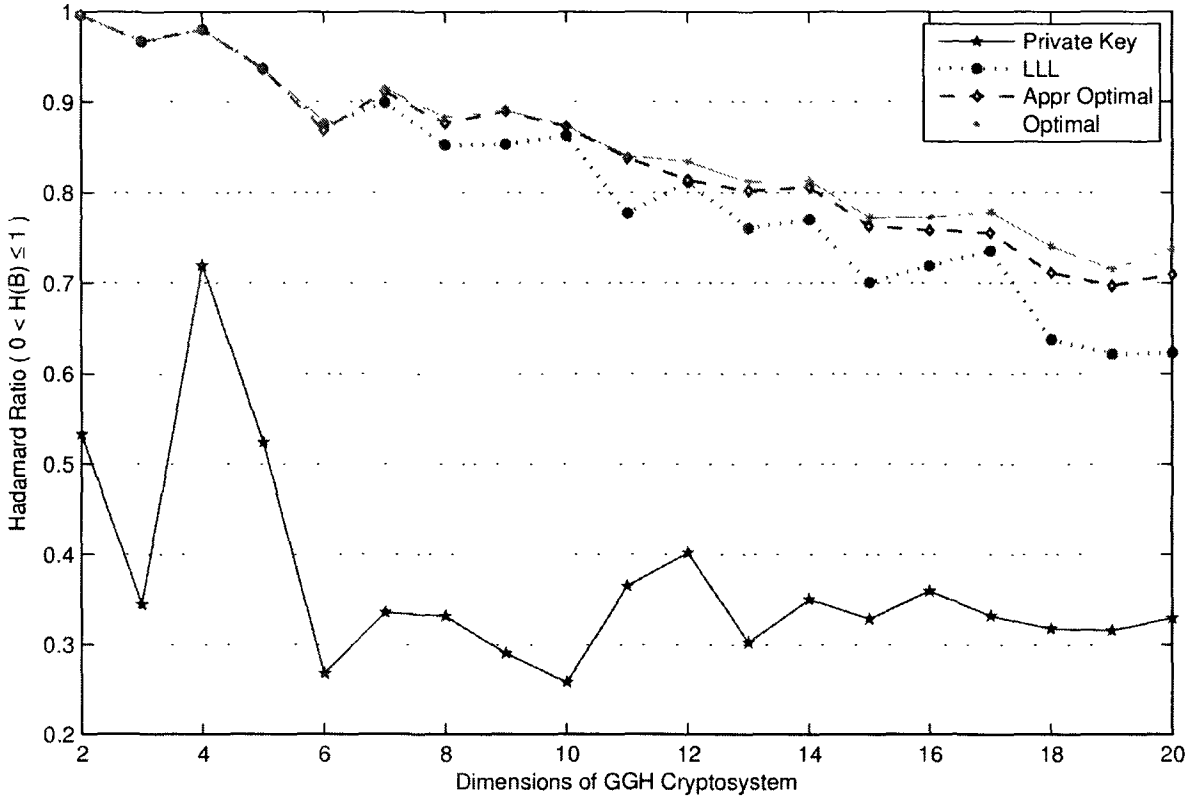

Figure 6.7: Hadamard Ratios of bases of integers within range [−9,999, 9,999]

GGH cryptosystem of dimensions larger than 5. That was one dimension less the previous sample. The average Hadamard Ratios of bases of dimension 6 were:

$$\mathcal{H}(B) \approx \begin{cases} 0.2679, & (\text{ the Private Keys }) \\ 0.8741, & (\text{ the LLL-Reduced bases }) \\ 0.8685, & (\text{ the Approximate Optimally-Reduced bases }) \\ 0.8788, & (\text{ the Optimally-Reduced bases }) \end{cases}$$

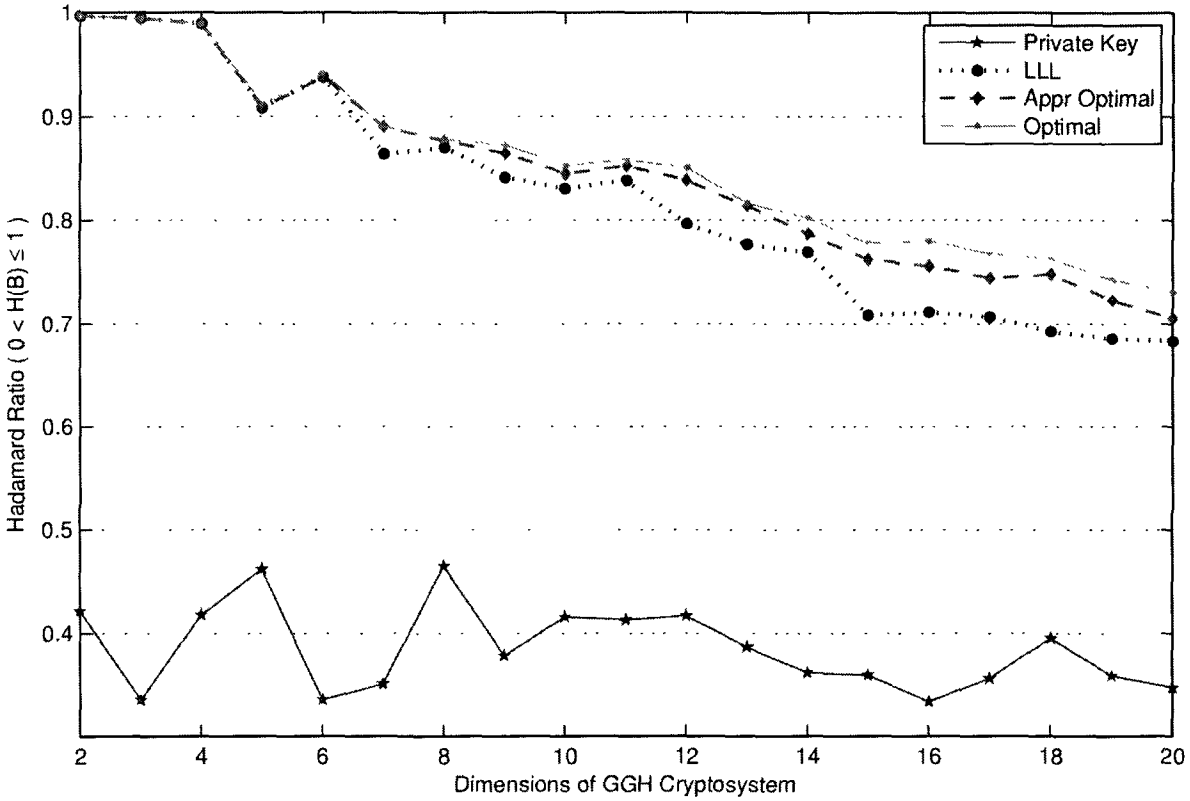

Figure 6.8: Hadamard Ratios of bases of integers within range [−99,999, 99,999]

Now we increased the range of the elements of the Private Key to [−99,999, 99,999], which was approximately 10 times larger than the bases shown in **Figure**-(6.7). The **Figure**-(6.8) illustrates the average Hadamard Ratios of the bases produced by the three

lattice reduction algorithms.

In this case, the average Hadamard Ratios differed from the dimension 5. The average Hadamard Ratios of bases of 5-dimensional GGH cryptosystems were:

$$\mathscr{H}(B) \approx \begin{cases} 0.4618, & \text{(the Private Keys)} \\ 0.9084, & \text{(the LLL-Reduced bases)} \\ 0.9084, & \text{(the Approximate Optimally-Reduced bases)} \\ 0.9118, & \text{(the Optimally-Reduced bases)} \end{cases}$$

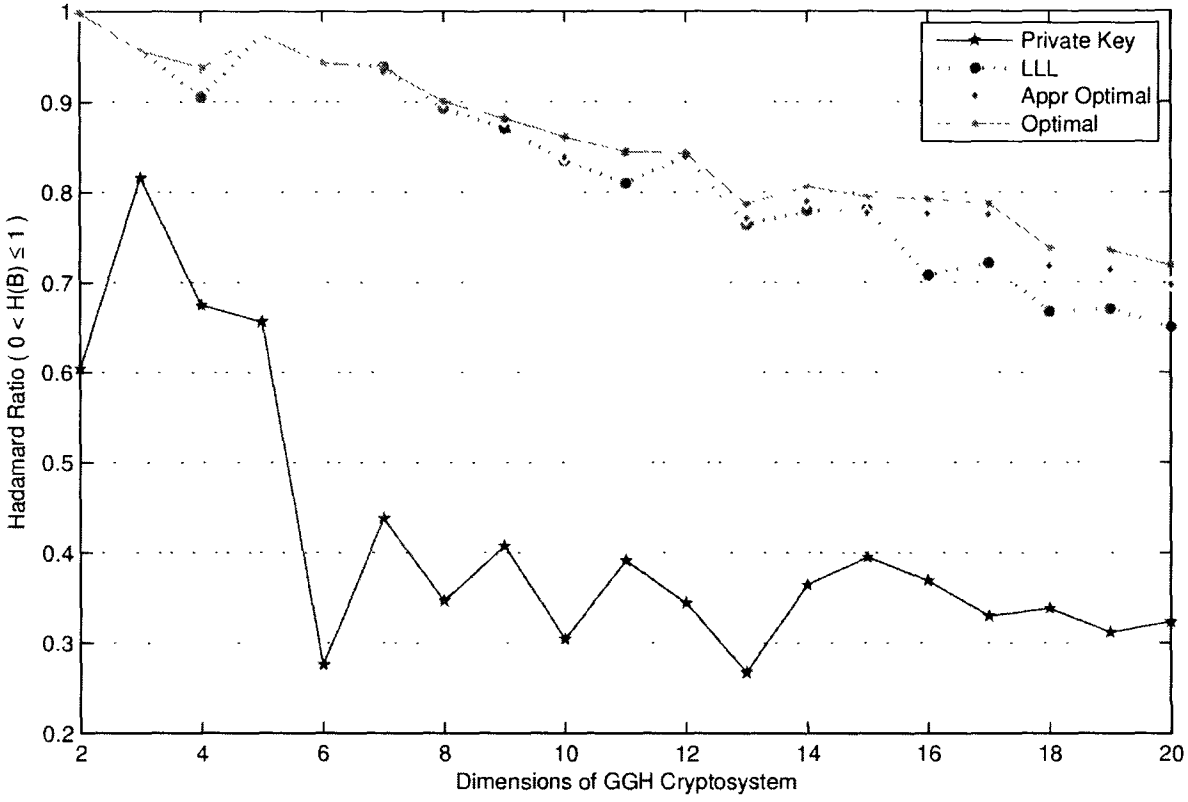

Figure 6.9: Hadamard Ratios of bases of integers within range [−999,999, 999,999]

In the last experiments, we increased the integer range to [−999,999, 999,999].

In this case, the average Hadamard Ratios showed a big difference in the GGH cryptosystems of dimension 4. The average Hadamard Ratios of bases in dimension 4 were:

$$\mathcal{H}(B) \approx \begin{cases} 0.6749, & \text{(the Private Keys)} \\ 0.9056, & \text{(the LLL-Reduced bases)} \\ 0.9376, & \text{(the Approximate Optimally-Reduced bases)} \\ 0.9376, & \text{(the Optimally-Reduced bases)} \end{cases}$$

To sum up, from the four **Figures**-(6.6), (6.7), (6.8), (6.9), we can see that the qualities of the bases that the three lattice reduction algorithm produced are influenced by two aspects:

1. The dimension of the GGH cryptosystem;

   When the dimension of the GGH cryptosystem increases, the Hadamard Ratios of the bases produced by the the three algorithms decreases.

2. The lengths of vectors in the Private Key;

   Comparing the four figures, we can see that when the lengthes of the vectors in the Private Key increases, the qualities of the bases produced by the **LLL**-(4) decreased. Whereas, the increase of elements in the Private Key did not influence the qualities of bases produced by the **APPROXIMATE OPTIMALLY-REDUCED** algorithm-(5) and the **OPTIMALLY-REDUCED** algorithm-(7), which means these two algorithms are more stable than the **LLL**-(4).

6.4 Experimental Results of Attacking GGH

The main purpose of this thesis is to attack the GGH cryptosystem using lattice reduction algorithms. We decrypted the ciphertexts using the bases produced by the three lattice reduction algorithms and the **BABAI'S** algorithm-(1). To distinguish the results of decryption , the *missing* ratios of attacking the GGH cryptosystem by the bases were accounted. We call it a *miss* if the basis cannot decrypt the ciphertext correctly.

In this part, the integer range we chose for the elements in the Private Key V as [−999, 999], and forced the Hadamard Ratio $\mathscr{H}(W)$ of the Public Key W no larger than 1.0×10^{-4}. The entries of perturbation vector **r** were randomly selected between −99 and 99. We used the same integer range for the plaintext **m**, which was also randomly generated. The plaintext **m** was encrypted according to the **Formula**-(3.2.1). The ciphertext **e** was decrypted by the **Formula**-(3.2.2) using the bases produced by the three algorithms. The dimension of the GGH cryptosystem varied from 1 to 20. In each dimension, the encryption and decryption processes ran 100 times to get the total missing ratios for each lattice reduction algorithm.

Figure-(6.10) illustrates the results of attacking the GGH cryptosystem.

When the dimension was less and equal than 9, the ciphertexts could be decrypted by all reduced bases and the Private Key V. For dimension 10, the Private Key started missing decryption. In the GGH cryptosystems of dimension 16, the bases produced by the **LLL REDUCTION** algorithm-(4) and the **APPROXIMATE OPTIMAL REDUCTION** algorithm-(5) could not always decrypted the ciphertext correctly. For the 17 dimensional GGH cryptosystem, the **OPTIMAL REDUCTION** algorithm-(7) started missing.

To sum up, we can see from the **Figure**-(6.10) that generating reasonable orthogonal

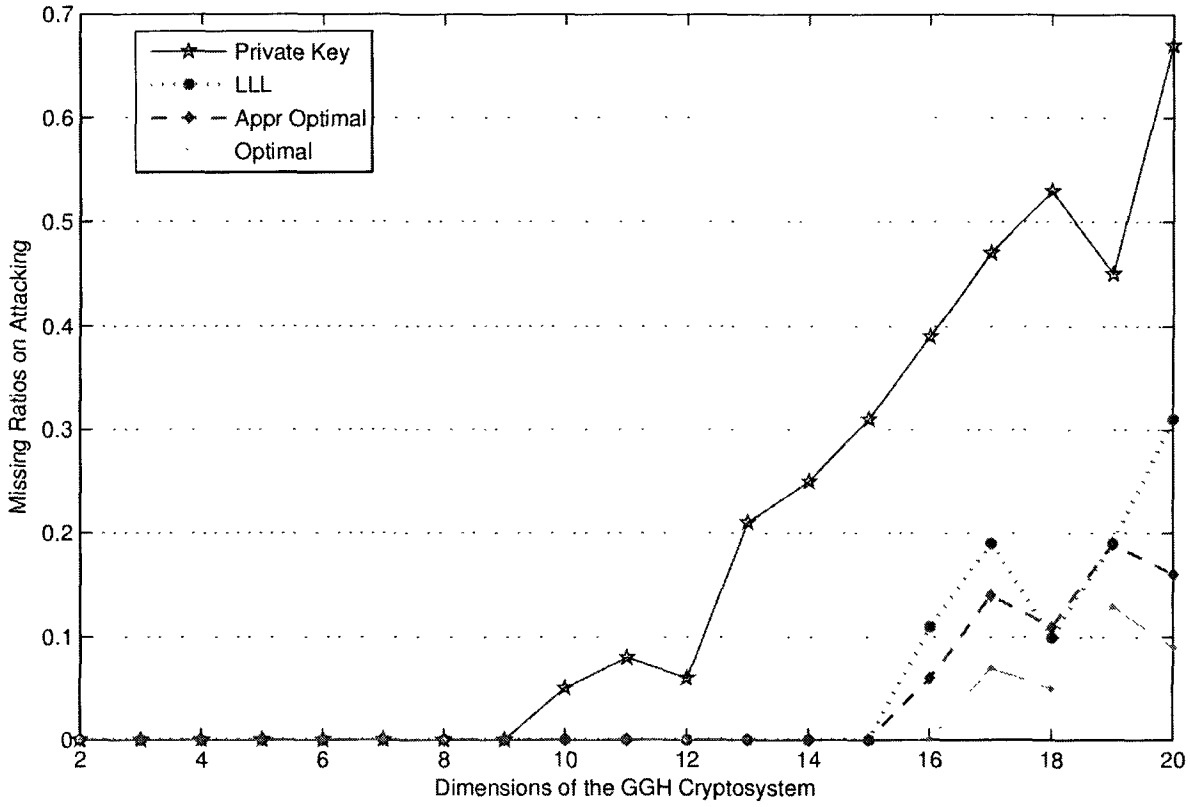

Figure 6.10: Missing ratios of attacking the GGH cryptosystem

bases by the lattice reduction algorithms is a possible way to attack the GGH cryptosystem.

Chapter 7

Conclusion and Future Works

In this thesis, we proposed a novel method to attack the GGH cryptosystem. That is, using the lattice reduction algorithm to generate a better basis based on the Public Key, then decrypted the ciphertext using this new produced basis.

We first introduced the concepts of lattices and bases in chapter 2. Then the GGH cryptosystem was presented in chapter 3. The GGH cryptology is a lattice based cryptosystem, which takes advantages of the hardness of finding the closest vector in a lattice. Other related definitions were introduced along side the GGH cryptosystem, including the Hadamard Ratio, the Fundamental Domain and the **BABAI'S** algorithm-(1), which was a widely used algorithm in order to discover the closest vector in a lattice. The **BASIC ENUMERATION** algorithm-(3) introduced in chapter 4 is an SVP solving algorithm, which compares all candidates within a certain range and finds the shortest vector.

Chapter 5 is the main part of this thesis. It introduced three lattice reduction algorithms, which are the **LLL REDUCTION** algorithm-(4), the **APPROXIMATE OPTIMAL REDUCTION** algorithm-(5), and the **OPTIMAL REDUCTION** algorithm-(7). The **BASIC ENUMERATION** algorithm-(3) was embedded into the **APPROXIMATE OPTIMAL REDUCTION**

algorithm-(5) and the **OPTIMAL REDUCTION** algorithm-(7).

The experimental results were shown in chapter 6. The experimental data we gave includes two aspects: the Hadamard Ratios of the produced bases and the missing ratios of decryption process including private key.

7.1 Conclusion

We discussed two other methods for attacking the GGH cryptosystem in chapter 3. The first method is invented by Phong Nguyen[38]. This method works by carefully choosing n linearly independent lattice vectors which form a basis for a sublattice of the original lattice. Then it attacks this sublattice to identify the potential properties of the original lattice. If the ciphertext fulfils the properties discovered, the plaintext may be decrypted, as least partial information could be discovered. The Phong Nguyen's GGH attacking method can be applied to GGH cryptosystems with dimensions up to 350.

The second method is introduced by Han, Daewan and Kim, Myung-Hwan and Yeom, Yongjin, who construct a *Paeng-Jung-Ha cryptosystem*[19]. They use lattice reduction algorithms to attack the encrypted signatures according to this constructed *Paeng-Jung-Ha cryptosystem.* In some special situations, the signature encrypted using the GGH cryptosystem can be broken by this method. This method can attack the signatures produced by GGH cryptosystem with dimensions up to 1000.

However, the correctness of the plaintexts decrypted by the above two GGH cryptosystem attacking methods depend on "special" situations[38, 19]. Therefore, they are not suitable for general cases.

The GGH attacking method introduced in this thesis works in general cases. Taking advantages of the lattice reduction algorithms, a more orthogonal basis can be produced

from the Public Key, which will be invoked to find the closest vector to the ciphertext. We claim that *for a GGH cryptosystem of arbitrary Public Key, the basis produced by the* **OPTIMAL REDUCTION** *algorithm-(7) is good enough to decrypt the ciphertext.*

For a randomly generated Private Key and the Public Key W with a very small Hadamard Ratio ($\mathscr{H}(W) \leq 1.0 \times 10^{-4}$), the experiments showed that the bases produced by the three lattice reduction algorithms were even better than the Private Key. The **LLL REDUCTION** algorithm-(4) produced a LLL-Reduced basis in polynomial time. Therefore, it was called as a pre-procedure by the other two lattice reduction algorithms, which are the **APPROXIMATE OPTIMAL REDUCTION** algorithm-(5) and the **OPTIMAL REDUCTION** algorithm-(7). These two algorithms also benefit from SVP solving algorithms, such that the bases produced by the two algorithm were better than the LLL-Reduced bases.

However, a main drawback of the two lattice reduction algorithms that should be mentioned is the performance. Their complexities are all exponential (with respects to the dimension of the GGH cryptosystem). In our experiments, it took an hour to generate an Optimally-Reduced basis for a 24-dimensional GGH cryposystem on average, and much longer in a GGH cryptosystem with the dimension large than 24. Therefore, for the high-dimensional GGH cryptosystems, attacking using lattice reduction algorithms is impractical.

7.2 Future Works

We proposed three lattice reduction algorithms in this thesis. They can produce the bases which are better than the Public Key. However, we did not give an accurate complexity analysis for the two lattice reduction algorithms, the **APPROXIMATE OPTIMAL**

REDUCTION algorithm-(5) and the **OPTIMAL REDUCTION** algorithm-(7). What we mentioned was that their complexities were exponential with respect to the dimension of the GGH cryptosystem. The performance of them were only estimated according to the experiments, hence they were very rough.

The future tasks includes:

1. The static analysis of complexities for the two algorithms;

   The further research will analyse the complexity of them. The full complexity analysis will be related to the dimension of the GGH cryptosystem and the Euclidean Lengths of the vectors in the Public Key.

2. Substitute the *Basis Enumeration* algorithm-(3) in the two algorithms;

   The SVP solving algorithm we chosen in this thesis is the *Basis Enumeration* algorithm-(3), which was embedded in the **APPROXIMATE OPTIMAL REDUCTION** algorithm-(5) and the **OPTIMAL REDUCTION** algorithm-(7). In term of the complexity, the *Basis Enumeration* algorithm-(3) is the major cost of the two algorithms. Its performance is worse than the Kannan's algorithm[25, 26] and the Ajtai's algorithm[2]. We will substitute the procedure *Basis Enumeration* with other SVP solving algorithms to increase the performance of the two lattice reduction algorithms in the future.

Appendix A

Experimental Result of Hadamard Ratio

The **Table**-(A.3) shows the details numbers of the experiments corresponding to the **Figure**-(6.6).

Average Hadamard Ratios of Bases				
Dimension	Private Key	LLL Reduced	Appr-Optimally Reduced	Optimally Reduced
2	0.8671	0.9987	0.9987	0.9987
3	0.5092	0.9858	0.9858	0.9858
4	0.4308	0.9537	0.9537	0.9537
5	0.5096	0.9727	0.9727	0.9727
6	0.2583	0.9517	0.9517	0.9517
7	0.5135	0.8418	0.8661	0.8722
8	0.4965	0.8765	0.8852	0.8868
9	0.3587	0.8868	0.8899	0.8926
10	0.3911	0.8417	0.8594	0.8742
11	0.4142	0.852	0.8562	0.8618
12	0.3284	0.7744	0.811	0.8208
13	0.369	0.7504	0.7829	0.798
14	0.3621	0.741	0.7653	0.7835
15	0.3373	0.7869	0.7782	0.8025
16	0.317	0.6951	0.7385	0.7595
17	0.3241	0.7081	0.7568	0.768
18	0.3419	0.5915	0.7026	0.7333
19	0.3434	0.6667	0.7204	0.7364
20	0.3539	0.6281	0.7022	0.7249

Table A.3: Hadamard Ratios of integer range $[-999, 999]$

The **Table**-(A.4) shows the details numbers of the experiments corresponding to the **Figure**-(6.7).

Average Hadamard Ratios of Bases				
Dimension	Private Key	LLL Reduced	Appr-Optimally Reduced	Optimally Reduced
2	0.5329	0.9959	0.9959	0.9959
3	0.3443	0.9669	0.9669	0.9669
4	0.7195	0.9802	0.9802	0.9802
5	0.5238	0.9365	0.9365	0.9365
6	0.2679	0.8741	0.8685	0.8788
7	0.3355	0.8987	0.9108	0.9149
8	0.3315	0.8526	0.8762	0.8826
9	0.2906	0.8533	0.8895	0.8901
10	0.2581	0.8627	0.8723	0.8736
11	0.3647	0.777	0.8382	0.8405
12	0.4017	0.8113	0.8138	0.8342
13	0.3023	0.7608	0.8015	0.8115
14	0.35	0.7705	0.8062	0.8132
15	0.3286	0.7007	0.763	0.7724
16	0.3592	0.719	0.7582	0.7726
17	0.3321	0.7355	0.7554	0.7785
18	0.3177	0.6382	0.7114	0.7407
19	0.3159	0.6219	0.6969	0.7154
20	0.3297	0.6836	0.7094	0.7373

Table A.4: Hadamard Ratios of integer range $[-9,999, 9,999]$

The **Table**-(A.5) shows the details numbers of the experiments corresponding to the **Figure**-(6.8).

Average Hadamard Ratios of Bases				
Dimension	Private Key	LLL Reduced	Appr-Optimally Reduced	Optimally Reduced
2	0.422	0.9968	0.9968	0.9968
3	0.3351	0.9945	0.9945	0.9945
4	0.4177	0.9894	0.9894	0.9894
5	0.4618	0.9084	0.9084	0.9118
6	0.3352	0.9377	0.9377	0.9412
7	0.3506	0.864	0.8904	0.8907
8	0.465	0.8702	0.8769	0.8783
9	0.3782	0.8414	0.8646	0.8724
10	0.4159	0.8305	0.8449	0.8532
11	0.4131	0.8383	0.8525	0.8583
12	0.417	0.7963	0.8385	0.8507
13	0.3868	0.7767	0.8138	0.8166
14	0.3617	0.7693	0.7869	0.802
15	0.3592	0.7084	0.7622	0.7776
16	0.3334	0.7114	0.7559	0.7801
17	0.3561	0.7065	0.7442	0.7677
18	0.3953	0.6926	0.7478	0.7627
19	0.3588	0.6855	0.7227	0.7427
20	0.347	0.6832	0.7053	0.7308

Table A.5: Hadamard Ratios of integer range $[-99,999,99,999]$

The **Table**-(A.6) shows the details numbers of the experiments corresponding to the **Figure**-(6.9).

Average Hadamard Ratios of Bases				
Dimension	Private Key	LLL Reduced	Appr-Optimally Reduced	Optimally Reduced
2	0.604	0.9977	0.9977	0.9977
3	0.8162	0.9563	0.9563	0.9563
4	0.6749	0.9056	0.9376	0.9376
5	0.6562	0.9735	0.9735	0.9735
6	0.2755	0.9427	0.9427	0.9427
7	0.4378	0.9383	0.9327	0.9383
8	0.347	0.8926	0.9004	0.9004
9	0.4071	0.871	0.8799	0.882
10	0.3043	0.8348	0.8386	0.8612
11	0.3914	0.81	0.8431	0.8451
12	0.3449	0.8397	0.8397	0.8433
13	0.2668	0.7644	0.7714	0.787
14	0.365	0.7792	0.7892	0.8061
15	0.3951	0.7805	0.7767	0.7951
16	0.3695	0.7082	0.7764	0.7924
17	0.3302	0.7219	0.7751	0.7873
18	0.3386	0.6676	0.7186	0.7387
19	0.3117	0.6702	0.7144	0.7359
20	0.3241	0.6504	0.6981	0.7198

Table A.6: Hadamard Ratios of integer range $[-999,999,999,999]$

Bibliography

[1] Dorit Aharonov and Oded Regev. Lattice problems in NP and co-NP. *J. ACM*, 52:749–765, September 2005.

[2] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, STOC '01, pages 601–610, New York, NY, USA, 2001. ACM.

[3] L. Babai. On lovasz lattice reduction and the nearest lattice point problem. *Combinatorica*, 6:1–13, 1986. 10.1007/BF02579403.

[4] M. Bessenyei. The Hermite–Hadamard inequality on simplices. *Amer. Math. Monthly*, 115(4):339–345, 2008.

[5] Dan Boneh, The Rsa Cryptosystem, Invented Ron Rivest, Adi Shamir, Len Adleman, and Was Rst. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46:203–213, 1999.

[6] Ludwig BrÃűcker. Zur theorie der quadratischen formen Ãijber formal reellen kÃűrpern. *Mathematische Annalen*, 210:233–256, 1974. 10.1007/BF01350587.

[7] P.M. Cohn. *Basic algebra: groups, rings, and fields.* Springer, 2003.

[8] Don Coppersmith. Finding a small root of a univariate modular equation. In *Proceedings of the 15th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'96, pages 155–165, Berlin, Heidelberg, 1996. Springer-Verlag.

[9] Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. Cryptology*, 10(4):233–260, 1997.

[10] Özgür Dagdelen and Michael Schneider. Parallel enumeration of shortest lattice vectors. In *Proceedings of the 16th international Euro-Par conference on Parallel processing: Part II*, Euro-Par'10, pages 211–222. Springer-Verlag, Berlin, Heidelberg, 2010.

[11] Glenn Durfee, Dan Boneh, and Ramarathnam Venkatesan. Cryptanalysis of RSA using algebraic and lattice methods. 2002.

[12] J Edmonds. Systems of distinct representatives and linear algebra. *Journal of Research of the National Bureau of Standards*, 71(B):241–245, 1967.

[13] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[14] Wen Zhang Franklin T. Luk, Sanzheng Qiao. A lattice basis reduction algorithm. Technical report, Institute for Computational Mathematics Hong Kong Baptist University, 2010.

[15] Nicolas Gama, Nick Howgrave-Graham, Henrik Koy, and Phong Q. Nguyen.

Rankin's constant and blockwise lattice reduction. In *CRYPTO*, pages 112–130, 2006.

[16] O. Goldreich, S. Goldwasser, and S. Halevi. Public-key cryptosystems from lattice reduction problems. Technical report, Cambridge, MA, USA, 1996.

[17] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, 3rd edition, 1996.

[18] Benqi Guo and Ivo Babuska. Mathematical framework for lattice problems. *Int. J. Numer. Anal. Mod.*, 4:307–341, 2007.

[19] Daewan Han, Myung-Hwan Kim, and Yongjin Yeom. Cryptanalysis of the Paeng-Jung-Ha cryptosystem from PKC 2003. In *Proceedings of the 10th international conference on Practice and theory in public-key cryptography*, PKC'07, pages 107–117, Berlin, Heidelberg, 2007. Springer-Verlag.

[20] Guillaume Hanrot and Damien Stehlé. Improved analysis of kannan's shortest lattice vector algorithm. In *Proceedings of the 27th annual international cryptology conference on Advances in cryptology*, CRYPTO'07, pages 170–186, Berlin, Heidelberg, 2007. Springer-Verlag.

[21] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers.* Oxford University Press, fifth edition, 1979.

[22] Johan Hastad. Solving simultaneous modular equations of low degree. *SIAM J. Comput.*, 17:336–341, April 1988.

[23] Bettina Helfrich. Algorithms to construct Minkowski reduced and Hermite reduced lattice bases. *Theoretical Computer Science*, 41:125 – 139, 1985.

[24] J. Hoffstein, J.C. Pipher, and J.H. Silverman. *An introduction to mathematical cryptography.* Undergraduate texts in mathematics. Springer, 2008.

[25] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, STOC '83, pages 193–206, New York, NY, USA, 1983. ACM.

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

[27] Louis Kruh and Cipher Deavours. The commercial enigma: beginnings of machine cryptography. *Cryptologia*, 26:1–16, January 2002.

[28] L.S. Leff. *PreCalculus the Easy Way.* Barron's E-Z Series. Barron's, 2005.

[29] A.K. Lenstra, H.W.jun. Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[30] Franklin T. Luk and Sanzheng Qiao. A pivoted LLL algorithm. *Linear Algebra and its Applications*, In Press, Corrected Proof:–, 2010.

[31] Franklin T. Luk and Daniel M. Tracy. An improved LLL algorithm. *Linear Algebra and its Applications*, 428(2-3):441 – 452, 2008. Special Issue devoted to the Second International Conference on Structured Matrices, Second International Conference on Structured Matrices.

[32] Romanos Malikiosis. An optimization problem related to Minkowski's successive minima. *Discrete Comput. Geom.*, 43:784–797, June 2010.

[33] J. Martinet. *Perfect Lattices in Euclidean Spaces.* Springer-Verlag, Berlin, 2003.

[34] Daniele Micciancio and Shafi Goldwasser. *Complexity of Lattice Problems: a cryptographic perspective*, volume 671 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, Massachusetts, March 2002.

[35] A. Ray Miller. The cryptographic mathematics of Enigma. *Cryptologia*, 19(1):65–80, January 1995.

[36] Wa Ho Mow and Key Words. Universal lattice decoding: Principle and recent advances. *Wireless Communications and Mobile Computing*, 3:553–569, 2003.

[37] P. Q. Nguyen and T. Vidick. Sieve algorithms for the shortest vector problem are practical. *J. of Mathematical Cryptology*, 2(2), 2008.

[38] Phong Nguyen. Cryptanalysis of the goldreich-goldwasser-halevi cryptosystem from crypto '97, 1999.

[39] Phong Q. Nguyen and Damien Stehlé. An LLL algorithm with quadratic complexity. *SIAM J. Comput.*, 39:874–903, August 2009.

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

[41] Xavier Pujol and Damien Stehlé. Rigorous and efficient short lattice vectors enumeration. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '08, pages 390–405, Berlin, Heidelberg, 2008. Springer-Verlag.

[42] Sanzheng Qiao. Integer least squares: sphere decoding and the LLL algorithm. In

Proceedings of the 2008 C3S2E conference, C3S2E '08, pages 23–28, New York, NY, USA, 2008. ACM.

[43] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26:96–99, January 1983.

[44] C. P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, August 1987.

[45] C. P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66:181–199, 1994. 10.1007/BF01581144.

[46] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. SIAM: Society for Industrial and Applied Mathematics, 1997.

[47] Paul M. B. Vitányi. Turing machine. *Scholarpedia*, 4(3):6240, 2009.

[48] Sanzheng Qiao Wen Zhang and Yimin Wei. Practical algorithms for constructing HKZ and Minkowski reduced bases. Technical report, McMaster University, 2011.