Revisiting Fully Homomorphic Encryption Schemes and Their Cryptographic Primitives

A thesis submitted in fulfillment of the
requirements for the award of the degree

Doctor of Philosophy

from

UNIVERSITY OF WOLLONGONG

by

Zhenfei Zhang

School of Computer Science and Software Engineering
April 2014

Dedicated to
My mother, my father and my wife

Declaration

This is to certify that the work reported in this thesis was done by the author, unless specified otherwise, and that no part of it has been submitted in a thesis to any other university or similar institution.

Zhenfei Zhang
April 3, 2014

Abstract

Lattice-based cryptography plays an important role in modern cryptography. Apart from being a perfect alternative of classic public key cryptosystems, should the quantum computers become available, the lattice-based cryptography also enables many applications that conventional cryptosystems, such as RSA encryption scheme, can not deliver. One of the most significant aspects from this point of view is the fully homomorphic encryption schemes.

A fully homomorphic encryption scheme allows one to arbitrarily operate on the encrypted messages, without decrypting it. This notion was raised in 1978, and it becomes a "holy grail" for the cryptographers for 30 years until 2009, Craig Gentry presented a framework to construct a fully homomorphic encryption using ideal lattice. The fully homomorphic encryption schemes, although they may be lacking of efficiency at its current stage, enable many important applications, such as secured cloud searching verifiable outsourced computing. Nevertheless, just like other cryptosystems, and perhaps all other inventions at the initial stage, the fully homomorphic encryption is young, prospective, and hence requires more research.

In this thesis, we focus on the security of fully homomorphic encryption schemes. The security of all known fully homomorphic encryption schemes can be reduced to some lattice problems. Therefore, our main tool, not surprisingly, is lattice. Previous work has shown that some of the fully homomorphic encryption schemes can be broken using lattice reduction algorithms. Indeed, there exist several lattice reduction algorithms, such as LLL and L^2, that run in polynomial time, that can break a homomorphic encryption scheme. However, the running time, even though it is a polynomial algorithm, is still beyond tolerance. Hence, our first step is to optimize those algorithms. In this thesis, we show three different improvements. To sum up, combining those techniques, we are able to accelerate the reduction form $O(d^{4+\varepsilon}\beta^2 + d^{5+\varepsilon}\beta)$ to $O(d^{2+\varepsilon}\beta^2 + d^{4+\varepsilon}\beta)$ when the algorithm is dedicated for those cryptosystems, where d is the dimension of the lattice, and β is the maximum bit-length of the norm of input

vectors. In practice, those techniques accelerate the reduction for approximately 10 times for moderate lattices, and when it is applied over the fully homomorphic encryption challenge, we can solve the challenge within 15.7 years, while the previous best result was 45 years.

We also analyzed the security of the fully homomorphic encryption scheme using integers under the chosen ciphertext attack model. The chosen ciphertext attack model is one of the classic security models as far as an encryption scheme is concerned. Indeed, we present a chosen ciphertext attack that breaks the security of the scheme. Further, in theory, the chosen ciphertext attack relies on the existence of a decryption oracle that might not always exist in practice. In this thesis, we also show that a decryption oracle can be constructed via a reaction attack for all fully homomorphic encryption schemes that are used in an outsourcing computing environment.

The last component of this thesis is a new fully homomorphic encryption scheme. To construct this scheme, we propose the notion of hidden lattices. We show that several hidden lattice problems, which are to be used in a fully homomorphic encryption scheme, are more difficult than the corresponding problems over a normal lattices. Hence, we base our scheme on a problem that is harder than all problems that existing fully homomorphic encryption schemes are based on. Since our problem is substantially harder to solve, our scheme can operate with smaller parameters, and hence be more efficient, compared to state-of-art ideal lattice based fully homomorphic encryption schemes.

Acknowledgement

My experience as a graduate student in the University of Wollongong has been wonderful. I am grateful to my principal supervisor Professor Willy Susilo for this opportunity. It was Willy who led me into the world of cryptography when I was doing my master's degree in Engineering over 5 years ago. Ever since then, Willy has been an excellent supervisor, who showed me the research direction and helped me to get started on the path to these degrees.

I would like to express my gratitude to my co-supervisor Dr. Thomas Plantard for his guidance. I am extremely grateful and indebted to him for his expert, sincere and valuable guidance and encouragement extended to me. He was always available for my questions and he was positive and gave generously of his time and vast knowledge. Everything I learned about lattices was from him.

I would also like to thank Dr. Man Ho Allen Au, for offering me an opportunity as a casual research assistant during the final year of my study, and for showing me a different angle in computer security.

In addition, a thank you to Dr. Craig Gentry and Professor Damien Stehlé for patiently explaining their schemes in details to me. Thank you to Professor Mihir Bellare for guiding me to the place to look for the answers. Thank you to Linda Taylor for proof reading this thesis.

This thesis was examined by Professor Josef Pieprzyk and Professor Damien Stehlé. I thank them for their invaluable suggestions improving this thesis.

Finally, my love and gratitude to my parents and my wife, for everything.

Publications

The following papers have been published or presented, and contain materials based on the content of this thesis. Note that the order of authors has been mainly done in an alphabetical order.

- **LLL for Ideal Lattice: Re-evaluation of the Security of Gentry-Halevi's FHE Scheme.**

  T. Plantard, W. Susilo and Z. Zhang

  Designs, Codes and Cryptography (DCC), 2014.

- **A Fully Homomorphic Encryption Scheme using Hidden Ideal Lattice.**

  T. Plantard, W. Susilo and Z. Zhang

  IEEE Transactions on Information Forensics and Security (TIFS), 2013.

- **Adaptive Precision Floating Point LLL**

  T. Plantard, W. Susilo and Z. Zhang

  18th Australasian Conference of Information Security and Privacy (ACISP 2013), Lecture Notes in Computer Science, Springer-Verlag, 2013.

- **Lattice Reduction for Modular Knapsack**

  T. Plantard, W. Susilo and Z. Zhang

  The Conference on Selected Areas in Cryptography (SAC 2012), Lecture Notes in Computer Science, Springer-Verlag, 2012.

- **On the CCA-1 Security of Somewhat Homomorphic Encryption over the Integers**

  Z. Zhang, T. Plantard and W. Susilo

The 8th International Conference on Information Security Practice and Experience (ISPEC 2012), Lecture Notes in Computer Science 7232, Springer-Verlag, pp. 353 - 368, 2012.

- **Reaction Attack on Outsourced Computing with Fully Homomorphic Encryption Schemes**

  Z. Zhang, T. Plantard and W. Susilo

  The 14th International Conference on Information Security and Cryptology (ICISC 2011), Lecture Notes in Computer Science, Springer-Verlag, 2011.

Contents

Chapter 1

Introduction

1.1 Fully Homomorphic Encryption

Cloud computing has been one of the biggest evolutions in the computer world in the past decade. Cloud computing allows access to highly scalable, inexpensive, on-demand computing resources that can execute the code and store the data that are provided to them. This feature is very attractive, as it alleviates most of the burden on Information Technology (IT) services from the consumer (or data owner). The data owner can outsource data to the cloud without having to maintain costly infrastructure.

Nevertheless, the adoption of cloud computing by business does have a major obstacle in that data owners are hesitant to allow untrusted cloud providers to have access to the data being outsourced. Merely encrypting the data prior to storing it on the cloud is not a viable solution, since encrypted data cannot be further manipulated. This means that if the data owner needs to, for example, search for particular information, then the data would need to be completely retrieved and decrypted - hence, a very costly operation.

Fully Homomorphic Encryption (FHE) is believed to be the solution for securing cloud computing. The problem of developing a fully homomorphic public key encryption scheme has been a long-standing open problem in the cryptography research community. Shortly after Shamir, Rivest, and Adleman invented RSA [RSA78], Rivest, Adleman and Dertouzos [RAD78] questioned whether a fully homomorphic encryption scheme, which they called *privacy homomorphism*, could be constructed. If such a scheme can be constructed, then essentially one can arbitrarily compute using encrypted data. Essentially, fully homomorphic encryption schemes enable one to apply homomorphic operations over an arbitrary number of given ciphertexts without the need to know the corresponding plaintexts.

Example 1.1 *Consider a situation where a user stores his/her colour images in the*

cloud in an encrypted form. The user wants to retrieve a greyscale version of one particular image from the cloud without leaking the information about the image itself.

- *Without fully homomorphic encryption, the user needs to retrieve the colour image from the cloud and then conduct the greyscaling process on their local machine.*
- *Using a fully homomorphic encryption, the cloud service can conduct the greyscaling algorithm in the cloud homomorphically (i.e. using the cloud's computing power), and then the user retrieves the image from the cloud and decrypts it to obtain a greyscale version of that image.*

Nevertheless, to construct such a scheme is challenging. The notion of fully homomorphic encryption was raised in 1978, and it became a "holy grail" for the cryptographers for 30 years until 2009, when Craig Gentry [Gen09b] successfully provided a framework for constructing fully homomorphic encryption schemes and furthermore provided a concrete construction in 2009 [Gen09a]. In addition, subsequent works based on his framework [SV10, vDGHV10, Gen10b, GHV10, SS10, GH11, BGV12, CMNT11, BV11b, LNV11, GHPS12, GHS12b, BV11a] have been proposed.

Fully homomorphic encryption schemes, although they may be lacking in efficiency at its current stage, enable many important applications, such as secured cloud searching and verifiable outsourced computing. Nevertheless, just like other cryptosystems, and perhaps all other inventions at the initial stage, the fully homomorphic encryption is young, prospective, and needs further research.

1.2 Lattice Theory and Cryptography

Lattice-based cryptography plays an important role in modern cryptography. It enables not only public key encryption schemes [GGH97, HPS98] but also digital signature schemes [LM08]. The security of those cryptosystems is based on hard lattice problems, in comparison with the factorization problem or the discrete logarithm problem that conventional encryption schemes are based on [RSA78, Kob87]. It is conjectured that lattice problems will remain difficult when quantum computers become available, while efficient algorithms have already been developed to solve the factorization problem and the discrete logarithm problem using quantum computers. Hence, it is believed that the lattice-based cryptosystem is one of the main candidates for post-quantum cryptography.

- **Classic Cryptography**: RSA, ECC, ElGamal,etc.

- **Post-quantum Cryptography**: Lattice-based, Multivariate equation-based [1], etc.

Nevertheless, lattice-based cryptography has drawn more and more attention in the past decade. As the research goes deeper and wider, amazing properties can be achieved with lattice-based cryptography, which we have never seen in classic encryption schemes.

Apart from being an alternative to classic public key cryptosystems, when the quantum computers become available, lattice-based cryptosystems also enable many applications that conventional cryptosystems, such as RSA encryption scheme, can not deliver. One of the most significant aspects from this point of view, is the fully homomorphic encryption scheme as we mentioned in the previous section.

Interestingly, when lattice first met cryptography, it was used to conduct cryptanalysis. To date, lattice, as a cryptanalysis tool, can be used against not only lattice-based cryptosystems, but also some classic cryptosystems. For instance, lattice reduction algorithms can be used to attack an RSA cryptosystem [Cop96a, Cop96b] when some of the most significant bits of the keys are known. To this end, many improvements towards lattice reduction algorithms [LLL82, Sch88, NS05a, NSV11, CN11] have been proposed. Some of them focus on reducing the running time of the algorithms, while others aim to improve the quality of the results. Nevertheless, achievements from both aspects affect cryptanalysis greatly.

1.3 Summary of Results

In this thesis, we focus on the security of fully homomorphic encryption schemes. The security of all known fully homomorphic encryption schemes can be reduced to some lattice problems. Therefore, our main tool, not surprisingly, is the lattice theory.

Previous work has shown that some of the fully homomorphic encryption schemes are broken using lattice reduction algorithms [Ngu11, CN11]. Indeed, there exists several lattice reduction algorithms, such as LLL [LLL82] and L^2 [NS05a, Ste10], that run in polynomial time, that break a homomorphic encryption scheme. However, the running time of the algorithms, even though it is polynomial, is still beyond tolerance. Therefore, our first step is to optimize those algorithms.

In the first part, we show three different improvements towards improving the running time of those algorithms. The main results are summarized in Table 1.1. In

[1] A public-key encryption scheme based on multivariate polynomials over finite fields.

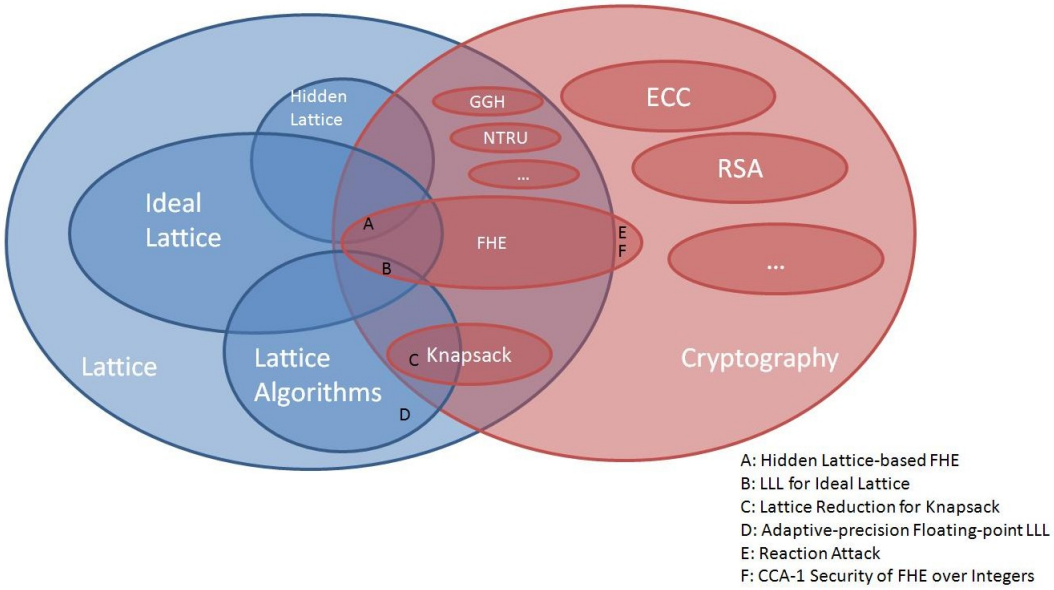

Figure 1.1: Summary of work in relation with cryptography and lattice theory

Table 1.1, ε is a real number between 0 and 1, ω is a valid exponent from matrix multiplications. For comparison, in this table, we also show the complexity of the LLL algorithm (the very first polynomial time lattice reduction algorithm), the L^2 algorithm (the best algorithm in practice) and the $\tilde{\mathrm{L}}^1$ algorithm (the best algorithm in theory).

Specifically, our results start with an adaptive-precision floating-point LLL algorithm (Figure 1.1, D) in Chapter 3. The proposed algorithm can be used for every purpose of lattice reduction. Although the proposed algorithm accelerates the procedure by 20% on average, the asymptotic complexity remains the same. Then, in Chapter 4, we present a recursive-reduction algorithm (Figure 1.1, C), that is more focused on knapsack-type bases, and we successfully improve the asymptotic complexity as shown in Table 1.1. The knapsack-type basis is somewhat a standard basis that is used in cryptanalysis. We shall discuss in more details in the corresponding chapter. To complete this part, in Chapter 5 we end with a dedicated algorithm, the LLL algorithm for ideal lattice (Figure 1.1, B), that can only be applied over an ideal lattice basis and bases with similar structures. These bases are mainly used in fully homomorphic encryption schemes and NTRU encryption schemes, etc. The improvement of our approach is both theoretical and practical.

To sum up, combining those techniques, in theory, we are able to accelerate the reduction from $O(d^{4+\varepsilon}\beta^2+d^{5+\varepsilon}\beta)$ to $O(d^{2+\varepsilon}\beta^2+d^{4+\varepsilon}\beta)$ when the algorithm is dedicated for those cryptosystems, where d is the dimension of the lattice, and β is the maximum bit-length of the norm of input vectors. In practice, those techniques accelerate the

Algorithms	Arbitrary Basis	Specific Basis	Comments
LLL	$O(d^{5+\varepsilon}\beta^{2+\varepsilon})$	$O(d^{4+\varepsilon}\beta^{1+\varepsilon})$	Classic Result
L^2	$O(d^{4+\varepsilon}\beta^2 + d^{5+\varepsilon}\beta)$	$O(d^{3+\varepsilon}\beta^2 + d^{4+\varepsilon}\beta)$	Best Practical Result
$\tilde{L}^1$	$O(d^{2+\varepsilon+\omega}\beta^{1+\varepsilon} + d^{5+\varepsilon}\beta)$	$O(d^{1+\varepsilon+\omega}\beta^{1+\varepsilon} + d^{4+\varepsilon}\beta)$	Best Theoretical Result
Ap-fplll	$O(d^{4+\varepsilon}\beta^2 + d^{5+\varepsilon}\beta)$	$O(d^{3+\varepsilon}\beta^2 + d^{4+\varepsilon}\beta)$	Better Practical Results
Rec-Red	N/A	$O(d^{2+\varepsilon}\beta^2 + d^{4+\varepsilon}\beta)$	Better Theoretical Bound
iLLL	N/A	$O(d^{2+\varepsilon}\beta^2 + d^{4+\varepsilon}\beta)$	Theory and Practice

Table 1.1: Comparison of time complexity

Rec-Red algorithm is dedicated to knapsack-type basis and the iLLL algorithm is dedicated to ideal lattice basis. The complexity of previous algorithms for those two specific bases are shown correspondingly. For more details of the complexy, see next chapter.

The Gentry-Halevi's Challenge	dim 512	dim 2048
The Previous Best Results/Prediction[CN11]	30 days	45 years
LLL implementation @2.66GHz	32 days	25.8 years
iLLL implementation @2.66GHz	24 days	23.6 years
iLLL prediction @4.0GHz	16 days	15.7 years

Table 1.2: Practical Result on Gentry-Halevi's Challenge

reduction for approximately 10 times for moderate lattice. Hence, those results are of independent interest for many other purposes beyond cryptanalysis of the fully homomorphic encryption challenges.

In the second part of the thesis, we show our cryptanalysis over the existing fully homomorphic encryptions. In chapter 6, we firstly show our results of the fully homomorphic encryption challenge using the improved lattice reduction algorithms proposed in the previous chapter. When applying those algorithms over the fully homomorphic encryption challenge [GH], where the dimension of the lattice is huge, we can solve the challenge within 15.7 years, while the previous best result required 45 years [Ngu11].

Another approach in terms of cryptanalysis is to analyze the security of the scheme under a certain security model. The two classic security models as far as an encryption scheme is concerned, are the chosen plaintext attack (CPA) model [GM84] and the chosen ciphertext attack (CCA) model [BDPR98]. While the security towards the chosen plaintext is always guaranteed by some computationally hard problem, the security towards the chosen ciphertext attack is still a concern. Indeed, it has been shown that

the level 2 of chosen ciphertext attack (CCA-2) security is impossible for fully homomorphic encryption schemes, and the level 1 security of the chosen ciphertext attack (CCA-1), which is a substantially weaker level, of a variant of fully homomorphic encryption scheme based on principal ideal lattice is actually compromised [LMSV11]. In Chapter 7, we also break the CCA-1 security of another variant of fully homomorphic encryption schemes using integers (Figure 1.1, F). Further, in theory, the chosen ciphertext attack relies on the existence of a decryption oracle that might not exist in practice. For this reason, we also show that a decryption oracle can indeed be constructed via a reaction attack (Figure 1.1, E), when a fully homomorphic encryption scheme is used in outsourced computing.

Finally, we present a new fully homomorphic encryption scheme (Figure 1.1, A) in the last part. To construct this scheme, we propose the notion of hidden lattices in Chapter 8. We show that several hidden lattice problems, that are to be used in a fully homomorphic encryption scheme, are harder than the corresponding problems over a normal lattice. We present a somewhat homomorphic encryption scheme in Chapter 9 and show how to convert it into a fully homomorphic encryption scheme in Chapter 10. We base our scheme on a problem that is harder than all problems that existing fully homomorphic encryption schemes are based on. Since our problem is substantially harder to solve, our scheme can operate with smaller parameters, compared with state-of-art ideal lattice based fully homomorphic encryption schemes.

1.4 Notations

The following notation is used throughout the rest of this thesis.

a, b, c	integers
x, y, z	variables
$\mathbb{Z}$, $\mathbb{R}$	set of integers, reals
$\mathbb{Z}_p$	integers modulo p
$x \leftarrow \mathbb{Z}$, $x \leftarrow \mathbb{R}$	sample x uniformly randomly from $\mathbb{Z}$, $\mathbb{R}$
$\vec{v} = \langle v_1, v_2, \ldots, v_n \rangle$	vector consist of elements $\langle v_1, v_2, \ldots, v_n \rangle$
$\mathbf{B} = (\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_n)$	matrix consist of vectors $(\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_n)$
$\mathcal{L}$, $\mathcal{L}(\mathbf{B})$	lattice, lattice spanned by a basis $\mathbf{B}$
$\det(\mathcal{L})$	determinant of a lattice $\mathcal{L}$
$\textsc{Dist}(\vec{v}, \mathcal{L})$	the distance between a vector and a lattice
λ_i	the i-th minima of a lattice
$\|\cdot\|$, $\|\cdot\|_2$	the Euclidean norm
$\|\cdot\|_\infty$	the infinity norm
$\lvert\cdot\rvert$	the absolute value
$[b]_a$	b mod a
$\lfloor a \rceil$	closest integer of a
$f(x)$, $g(x)$	polynomials
$\textsc{Rot}(\vec{v}, f(x))$, $\mathrm{Rot}(\vec{v})$	matrix from cyclic rotation of $\vec{v}$ modulo $f(x)$
$f(x) = O(g(x))$	there exists a positive real ε such that $\lvert f(x)\rvert \leq \varepsilon \lvert g(x)\rvert$ for all x
$\mathcal{M}(d)$	cost of multiplication of two d-bits integers
$\mathcal{O}$	oracle
λ	security parameter
$\binom{n}{k}$	choose k out of n

Chapter 2

Background

2.1 Modern Cryptography

2.1.1 Public Key Cryptography

The public key cryptography was proposed by Shamir, Rivest, and Adleman [RSA78] over 30 years ago. In comparison to the previous encryption schemes, where the same key is used to encrypt and decrypt, the public key cryptography uses two keys, a public one and a secret one. For this reason, these kinds of encryption schemes are also known as asymmetric encryption schemes.

Suppose Alice is the owner of a pair of keys **pk** and **sk**, where **pk** is known to everybody. Then, Bob can use Alice's public key to encrypt a message, which can only be decrypted by Alice herself. To construct a public key encryption scheme, one usually starts with building a certain one-way trapdoor function as follows:

Definition 2.1 (One-way Trapdoor Function) *A one-way trapdoor function, refer to as $y \leftarrow f(x)$, is a function that is easy to compute in one direction, for instance, given x and f it is relatively easy to compute y, while it is computationally hard to compute in the opposite direction (finding its inverse $f^{-1}(x)$) without a trapdoor. Further, the function can be easily inverted once the trapdoor is given.*

If one views x as the message and $y \leftarrow f(x, \mathbf{pk})$ as the encryption function with **pk**, then this one-way function defines an encryption algorithm. If one knows the trapdoor, then one can efficiently compute $x \leftarrow f^{-1}(y, \mathbf{sk})$. However, without a trapdoor (for example, **sk**), to compute $x \leftarrow f^{-1}(y)$ is not feasible. For instance, given y and f it would be hard to compute x.

Example 2.1 *Considering the following problem. Let p and q be two prime numbers. Let $N = pq$. Let e be an integer such that e and $(p-1)(q-1)$ are co-prime. The*

function $y \leftarrow x^e \bmod N$ *is a one-way trapdoor function. Given* e *and* N *one can efficiently compute* y *from* x*. But to compute* x *from* y *one seems to need* p *and* q*.*

To date, many complex problems are used to build the one-way trapdoor function for different cryptosystems. The classic RSA cryptosystem [RSA78] is based on the factorization problem of large integers. The Elliptic Curve Cryptography (ECC) [Kob87] is based on the discrete logarithm problem over a group. The NTRU cryptosystem [HPS98] is based on a problem of finding the closest vector of certain vector to a lattice. All those problems are relatively hard unless a certain piece of information is given.

However, with the rise of quantum computers, the factorization problem and the discrete logarithm problem will be easy to solve [Sho94]. As a result, the corresponding cryptosystems become insecure. Fortunately, lattice problems are conjectured to be hard even with quantum computers. For this reason, lattice-based cryptosystem is drawing more and more attention.

2.1.2 Security Modeling

In this thesis, we focus on the security of the some cryptosystems. The security of a cryptosystem is usually defined under certain security models. In the following, we describe briefly both Chosen-Plaintext Attack (CPA) [GM84] and Chosen-Ciphertext Attack (CCA) [BDPR98] for completeness.

The IND-CPA security game is defined as follows:

1. The challenger runs KeyGen algorithm and outputs a secret key **sk** and a public key **pk**;
2. The attacker is given the public key so that he/she can compute Encrypt(m, **pk**) locally;
3. The attacker then generates two ciphertexts m_0 and m_1;
4. The challenger generates $c = \text{Encrypt}(m_b, \mathbf{sk})$, where b is chosen uniformly randomly from $\{0, 1\}$;
5. The attacker outputs b'.

We say that an encryption scheme is CPA secure if the advantage of the attacker to win the game ($Pr[b = b'] - 1/2$) is negligible.

The IND-CCA-1/2 security game is defined as follows:

1. The challenger runs KeyGen algorithm and outputs a secret key **sk** and a public key **pk**;
2. The attacker is given two oracles, an encryption oracle and a decryption oracle;
3. The attacker then generates two ciphertexts m_0 and m_1;
4. The challenger generates $c = \text{Encrypt}(m_b, \mathbf{sk})$, where b is chosen uniformly randomly from $\{0, 1\}$;
5. (Only for CCA-2) The attacker is given the two oracles again, but it cannot query on c;
6. The attacker outputs b'.

We say that an encryption scheme is CCA-1/2 secure if the advantage of the attacker to win the game $(Pr[b = b'] - 1/2)$ is negligible.

2.2 Lattice Theory

2.2.1 Lattice Basics

In this section, we review some concepts of lattice theory that will be used throughout this thesis. The lattice theory, also known as the geometry of numbers, was introduced by Minkowski in 1896 [Min96]. We refer readers to [Lov86, MG02] for a more detailed account.

Definition 2.2 (Lattice) *A lattice $\mathcal{L}$ is a discrete sub-group of $\mathbb{R}^n$, or equivalently the set of all the integral combinations of $d \leq n$ linearly independent vectors over $\mathbb{R}$.*

$$\mathcal{L} = \mathbb{Z}\vec{b}_1 + \mathbb{Z}\vec{b}_2 + \cdots + \mathbb{Z}\vec{b}_d, \vec{b}_i \in \mathbb{R}^n$$

$\mathbf{B} = (\vec{b}_1, \ldots, \vec{b}_d)$ is called a basis of $\mathcal{L}$ and d is the dimension of $\mathcal{L}$, denoted as $\dim(\mathcal{L})$. $\mathcal{L}$ is a full rank lattice if d equals n.

Definition 2.3 (Determinant) *Let $\mathcal{L}$ be a lattice. Its determinant, denoted as $\det(\mathcal{L})$, is a real value, such that for any basis $\mathbf{B}$ of $\mathcal{L}$, $\det(\mathcal{L}) = \sqrt{\det(\mathbf{B} \cdot \mathbf{B}^T)}$, where $\mathbf{B}^T$ is the transpose of $\mathbf{B}$.*

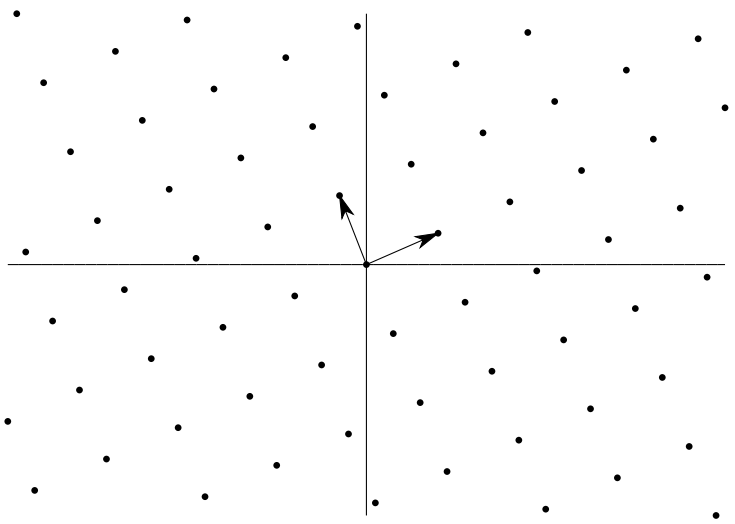

Figure 2.1: A 2-dimensional lattice and its basis

Definition 2.4 (Hermite Normal Form) *Let $\mathcal{L}$ be an integer lattice of dimension d and $\mathbf{H}$ be a basis of $\mathcal{L}$. Then the basis $\mathbf{H}$ is the Hermite Normal Form (HNF) basis of $\mathcal{L}$ if and only if*

$$\forall 1 \leq i, j \leq d \qquad \mathbf{H}_{i,j} \begin{cases} = 0 & if \quad i < j; \\ \geq 0 & if \quad i \geq j; \\ < \mathbf{H}_{j,j} & if \quad i > j. \end{cases}$$

For a given lattice $\mathcal{L}$, there exists an infinite number of bases. However, its determinant and its HNF basis is unique. Looking ahead, in terms of cryptanalysis, generally speaking, the HNF basis is believed to leak the least information of the corresponding lattice, since it can be obtained by any of its basis in polynomial time $O(d^3\beta)$ [MW00].

For a vector $\vec{b} = \langle b_1, b_2, \ldots, b_d \rangle$, its Euclidean norm, denoted by $\|\vec{b}\|$, is $\sqrt{\sum_{i=1}^{d} b_i^2}$. The distance of two vectors $\vec{b}_1$ and $\vec{b}_2$, referred to as $\textsc{Dist}(\vec{b}_1, \vec{b}_2)$, is defined by $\|\vec{b}_1 - \vec{b}_2\|$, while the distance between a vector $\vec{b}$ and a lattice $\mathcal{L}$, denoted by $\textsc{Dist}(\vec{b}, \mathcal{L})$, is the minimum value of $\textsc{Dist}(\vec{b}, \vec{v})$ for any $\vec{v} \in \mathcal{L}$.

Definition 2.5 (Successive Minima) *Let $\mathcal{L}$ be an integer lattice of dimension d. The i-th successive minima with respect to $\mathcal{L}$, denoted by λ_i, is the smallest real number, such that there exist i non-zero linearly independent vectors $\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_i \in \mathcal{L}$ with*

$$\forall i, \quad \|\vec{b}_1\|, \|\vec{b}_2\|, \ldots, \|\vec{b}_i\| \leq \lambda_i,$$

where $\|\cdot\|$ denotes the Euclidean norm of the corresponding vector.

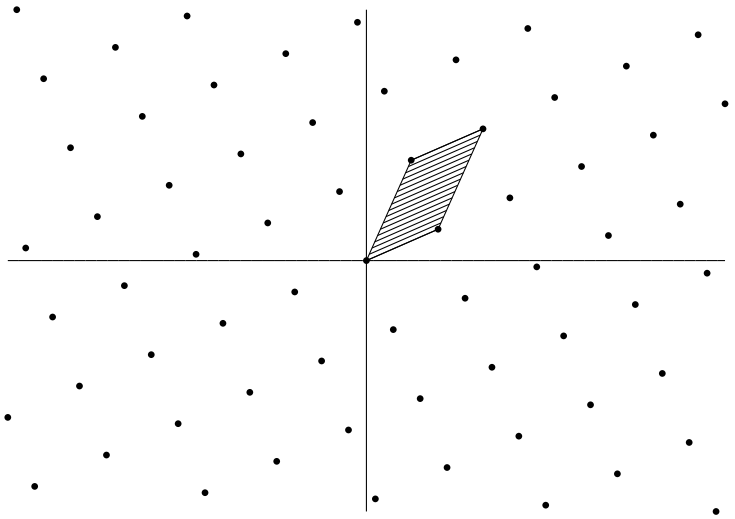

Figure 2.2: The volume of a lattice

In addition, if the lattice is random (see Theorem 2.1), then the value of i-th minima is estimated by the Gaussian heuristic as follows:

$$\forall i, \quad \lambda_i(\mathcal{L}) \sim \sqrt{\frac{d}{2\pi e}} \det(\mathcal{L})^{\frac{1}{d}}. \tag{2.1}$$

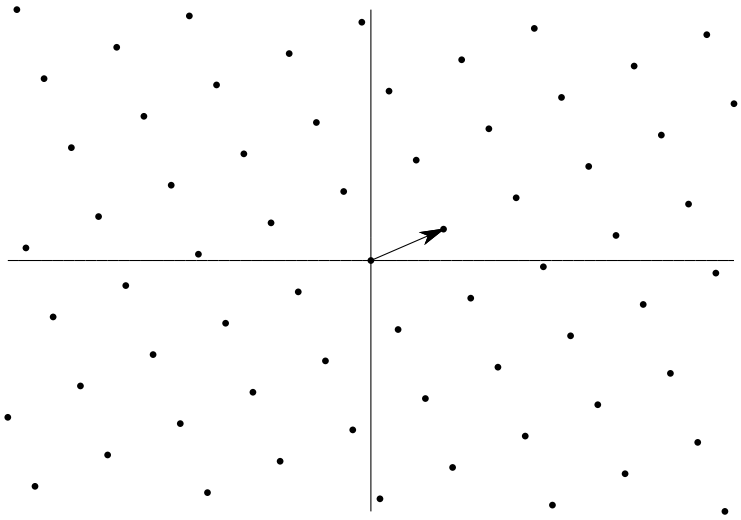

Figure 2.3: A shortest non-zero vector of a lattice

Definition 2.6 (Hermite factor) *Let* $\mathbf{B} = (\vec{b}_1, \ldots, \vec{b}_d)$ *a basis of* $\mathcal{L}$*. The Hermite factor with respect to* $\mathbf{B}$ *is defined as* $\frac{\|\vec{b}_1\|}{\det(\mathcal{L})^{\frac{1}{d}}}$.

Note that the Hermite factor is a good indicator of the quality of a reduced basis (see next subsection for definitions of lattice reductions).

Example 2.2 *Figure 2.1, 2.2 and 2.3 give the same example of a 2-dimensional lattice. The basis, as shown in Figure 2.1, is*

$$\mathbf{B} = \begin{pmatrix} 8 & 5 \\ -3 & 11 \end{pmatrix}.$$

The determinant of the lattice is 103 *(Figure 2.2). The Hermite Normal Form basis of this lattice is*

$$\mathbf{H} = \begin{pmatrix} 103 & 0 \\ 65 & 1 \end{pmatrix}.$$

A shortest non-zero vector is given in Figure 2.3, hence, the first minima $\lambda_1 = \| \langle 8, 5 \rangle \| = \sqrt{8^2 + 5^2} = \sqrt{89}$. *The Hermite factor of* $\mathbf{B}$ *is* $\sqrt{\frac{89}{103}}$, *while the Hermite factor of* $\mathbf{H}$ *is* $\sqrt{103}$.

For any vector $\vec{v} = \langle v_1, \ldots, v_d \rangle$, denote $v(x) = \sum_{i=1}^{d} v_i x^{i-1}$ the polynomial form of $\vec{v}$. Let $\text{ROT}(\vec{v}, f)$ be the matrix consisting of vectors from $\{x^i v(x) \bmod f(x)\}$ for $1 \leq i \leq d$. Then $\text{ROT}(\vec{v}, f)$ forms a rotation basis. For instance, if $f(x) = x^n + 1$ and $\vec{v} = \langle v_1, v_2, \ldots, v_d \rangle$, then the rotation basis $\mathbf{B}$ is of the following form:

$$\mathbf{B} = \begin{pmatrix} v_1 & v_2 & v_3 & \ldots & v_d \\ -v_d & v_1 & v_2 & \ldots & v_{d-1} \\ -v_{d-1} & -v_n & v_1 & \ldots & v_{d-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -v_2 & -v_3 & -v_4 & \ldots & v_1 \end{pmatrix}$$

Then, we have the definition of principal ideal lattices.

Definition 2.7 (Ideal Lattice [Mic07]) *Let* R *be a polynomial ring* $\mathbb{Z}[X]/f(x)$, *where* $f(x) \in \mathbb{Z}[X]$ *is a monic irreducible polynomial of degree* n. *The ideal lattice over* R, *is a lattice* $\mathcal{L}$ *such that for any* $\vec{v} \in \mathcal{L}$, *the vector corresponding to the polynomial* $x \times v \bmod f(x)$ *also belongs to* $\mathcal{L}$.

Definition 2.8 (Principal Ideal Lattice) *Let* R *be a polynomial ring* $\mathbb{Z}[X]/f(x)$, *where* $f(x) \in \mathbb{Z}[X]$ *is a monic irreducible polynomial of degree* n. *Let* $\vec{v} \in \mathbb{Z}^n$. *The ideal lattice over* R *with respect to* $\vec{v}$, *denoted by* $\mathcal{L}(\text{ROT}(\vec{v}, f))$ *is the set of all integral linear combinations of* $\vec{v}$ *and its rotation vectors.*

Note that for ideal lattices, we have $n = d$ for all bases. The principal ideal lattice is used to construct efficient encryption schemes [GH11, SV10], since it can be represented

by three integers $\{\alpha, \delta, d\}$ rather than a full basis, where δ is the determinant of the lattice, and α is an integer such that $f(\alpha) = 0 \pmod{\delta}$, i.e., $\alpha^d + 1 = 0 \pmod{\delta}$ if $f(x) = x^n + 1$. As shown in [GH11] with those three integers, one is able to construct a basis $\mathbf{H}$, that is the Hermite Normal Form basis of the lattice. Then $\mathcal{L}(\mathbf{H})$ defines an principal ideal lattice.

$$\mathbf{H} = \begin{pmatrix} \delta & 0 & 0 & \dots & 0 & 0 \\ (-\alpha) \bmod \delta & 1 & 0 & \dots & 0 & 0 \\ (-\alpha^2) \bmod \delta & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ (-\alpha^{d-1}) \bmod \delta & 0 & 0 & \dots & 0 & 1 \end{pmatrix}$$

Looking ahead, from the point of view of cryptography, one sometimes also uses $\mathbf{H}'$ which shares a similar property with $\mathbf{H}$ in terms of lattice reduction, since they have same dimension and the coefficients are of approximately same length.

$$\mathbf{H}' = \begin{pmatrix} \delta & 0 & 0 & \dots & 0 & 0 \\ -\alpha & 1 & 0 & \dots & 0 & 0 \\ 0 & -\alpha & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -\alpha & 1 \end{pmatrix}.$$

We proof that $\mathbf{H}$ and $\mathbf{H}'$ span a same lattice as follows: firstly, all vectors in $\mathbf{H}'$ can be obtained by cyclic rotation of vectors in $\mathbf{H}$. That is, any vector that is in $\mathcal{L}(\mathbf{H}')$ is also in $\mathcal{L}(\mathbf{H})$. Then, since $\mathcal{L}(\mathbf{H})$ and $\mathcal{L}(\mathbf{H}')$ have same determinant and rank, they are the same lattice.

Theorem 2.1 (Random Lattice [GM06]) *Let* $\mathbf{B}$ *be a basis as follows:*

$$\mathbf{B} = \begin{pmatrix} X_1 & 0 & 0 & \dots & 0 \\ X_2 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ X_d & 0 & 0 & \dots & 1 \end{pmatrix}$$

$\mathbf{B}$ *spans a randomly lattice, if* X_1 *is a large prime and* X_i*-s* ($i \neq 1$) *are chosen uniformly between* 0 *and* X_1.

Indeed, $\mathbf{B}$ is a modular knapsack-type basis (see next subsection). It gains its great importance in lattice theory because it spans a random lattice, as defined in [GM06].

Further, these bases are somewhat standard to analyze lattice reductions. They are believed to leak the least information for the corresponding lattice, since it can be obtained by any lattice basis within polynomial time by performing an HNF transformation over the basis. They are also adopted in [GN08, NS06] where the behavior of lattice reduction algorithms is thoroughly analyzed. In addition, when setting $\beta \sim 10d$, these bases are also used for the shortest vector problem (SVP) challenges in [svp].

2.2.2 Lattice Problems

In this subsection we review some basic problems of lattice theory. For many years, lattice was used for cryptanalysis. It is one of the most powerful tools to break certain cryptography. It was not until recently that researchers successfully build a cryptosystem with it, where the security of the cryptosystem relies on the hardness of certain lattice problems. Among all lattice problems, the Shortest Vector Problem (SVP) and the Closest Vector Problem (CVP) are the core problems.

Definition 2.9 (Shortest Vector Problem) *Given an arbitrary basis* $\mathbf{B}$ *of a lattice* $\mathcal{L}$*, find a shortest non-zero vector* $\vec{v} \in \mathcal{L}$*.*

Definition 2.10 (Closest Vector Problem) *Given an arbitrary basis* $\mathbf{B}$ *of a lattice* $\mathcal{L}$ *and a vector* $\vec{v}_0 \in \mathbb{R}^n$*, find a vector* $\vec{v}_1 \in \mathcal{L}$ *such that* $\|\vec{v}_0 - \vec{v}_1\| \leq \textsc{Dist}(\mathcal{L}, \vec{v}_0)$*.*

Generally speaking, the SVP problem is to find a shortest non-zero vector within a lattice, while the CVP problem is to find a closest point in the lattice for a certain vector. Both problems are hard to solve, except for some trivial dimensions, i.e. 1 dimensional lattice, while solving a SVP is no easier than solving a CVP. For a recent survey of SVP and CVP problem, we refer the reader to [HPS11a].

Moreover, quantum algorithms solving CVP or SVP have not yet been found. Hence, lattice-based cryptography is one of the best candidates for post-quantum use. Indeed, even the approximated version (see below) of those problems are believed to be NP-hard and quantum resistant, with small approximation factors.

Definition 2.11 (Approximated Shortest Vector Problem (γ-SVP)) *Given a lattice* $\mathcal{L}$*, let* λ_1 *be the first minima of* $\mathcal{L}$*, the Approximated Shortest Vector Problem, denoted by* γ*-SVP, is to find a non-zero vector* $\vec{v} \in \mathcal{L}$ *such that* $\|\vec{v}\| \leq \gamma\lambda_1$*.*

Definition 2.12 (Approximated Closest Vector Problem (γ-CVP)) *Given a lattice* $\mathcal{L}$ *and a vector* $\vec{v}_1$*, the Approximated Closest Vector Problem, denoted by* γ*-CVP, is to find a non-zero vector* $\vec{v}_2 \in \mathcal{L}$ *such that* $\textsc{Dist}(\vec{v}_1, \vec{v}_2) \leq \gamma\textsc{Dist}(\vec{v}_1, \mathcal{L})$*.*

Those two problems derives problems that cryptosystems are based on, for instance, the bounded distance decoding (BDD) problem. The BDD problem is another very important lattice problem. In the reminder of the thesis, we shall refer to this problem repetitively, since it enables the fully homomorphic encryption schemes over ideal lattices.

Definition 2.13 (Bounded Distance Decoding problem (γ-BDD)) *Let $\gamma \in \mathbb{R}^+$ be a positive real. Let $\mathcal{L}$ be an n dimensional (ideal) lattice, and $\vec{v} \in \mathbb{Z}^n$, such that there exists a unique vector $\vec{u} \in \mathcal{L}$ satisfying $\textsc{Dist}(\vec{v}, \vec{u}) \leq \gamma$. The γ-Bounded Distance Decoding problem over (ideal) lattice, denoted by γ-BDD (γ-BDDi, resp.), is to find $\vec{u}$, given a basis of $\mathcal{L}$ and $\vec{v}$.*

Usually, to break a cryptosystem, one only need to solve a decisional version of the above problem. The decisional version of the BDD is defined as follows.

Definition 2.14 (Decisional BDD Problem) *Let $\gamma \in \mathbb{R}^+$ be a positive real. Let $\mathcal{L}$ be an n dimensional (ideal) lattice, and $\vec{v} \in \mathbb{Z}^n$. The Decisional γ-Bounded Distance Decoding problem over (ideal) lattice, denoted by Dec γ-BDD$_n$ (Dec γ-BDDi$_n$, resp.), is to decide if there exists a unique vector $\vec{u} \in \mathcal{L}$ satisfying $\textsc{Dist}(\vec{v}, \vec{u}) \leq \gamma$ or not, given a basis of $\mathcal{L}$ and $\vec{v}$.*

There have been several definitions of BDD (see [LM09, GH11] for comparison), due to the simplification of their proof reductions. In our case, we subsequently define BDD with slight modification to achieve the same goal. Nevertheless, it is clear that all these definitions capture the same notion.

2.2.3 Lattice-related Problems

Now we shall describe some other problems related to lattice. We remark that lattice theory can be used to analysis all problems discussed in this subsection.

As mentioned earlier, there are mainly three types of fully homomorphic encryption schemes. The schemes are based on three different problems. In the last subsection, we have described the bounded distance decoding (BDD) problem that enables lattice-based schemes. Now, we introduce the other two problems, namely, the approximate greatest common divisor problem and the learning with error problem.

Definition 2.15 (AGCD Problem) *Let $c_i \in \mathbb{Z}$, τ integers such that there exist some unique integers $r_i \in \mathbb{Z}$ and a unique integer $p \in \mathbb{N}$ such that $\forall i, p|(c_i - r_i)$ and $\forall i, |r_i| \leq$*

$\gamma < p/2$. Then, the Approximate Greatest Common Divisor problem, denoted by γ-AGCD, is to find p, given c_i.

The above definition describes the general version of AGCD problem. By setting $r_1 = 0$ one obtains the partial version (P-AGCD). For the rest of this thesis, we are only concerned with the general version of this problem. As in the case of BDD, the problem of AGCD has been defined slightly differently in the literature (see [vDGHV10, HG01] for comparison). Again, we apply the same principle to achieve our goal. Nevertheless, all of these definitions still capture the same notion.

The classic way to attack this problem makes use of lattices [vDGHV10]. Nevertheless, there are also attacks like [CN12] which do not use lattices.

The Learning With Error (LWE) problem, and its ring version, the Learning With Error over the Ring (R-LWE) problem, were introduced by Regev [Reg05, Reg10]

Definition 2.16 (Learning With Error problem) *Given an integer q and an error distribution Ξ over $\mathbb{Z}_q$, Let $\vec{s}$ be a random vector in $\mathbb{Z}_q^n$. Let $(\vec{a}_1, \ldots, \vec{a}_n)$ and $(\vec{e}_1, \ldots, \vec{e}_n)$ be two sets of vectors, where $\vec{a}_i$ is randomly chosen from $\mathbb{Z}_n^q$, and $\vec{e}_i$ is chosen according to Ξ. The Learning With Error problem, denoted by LWE, is given as many independent pairs of $\vec{a}_i \cdot \vec{s} + \vec{e}_i$ and $\vec{a}_i$, find $\vec{s}$.*

In addition, the ideal lattice-based fully homomorphic encryption schemes and the integer-based fully homomorphic encryption schemes all uses the squashing technique, which, as we shall discuss in the next section, relies on a Sparse Subset Sum Problem (SSSP). It is derived from the (modular) knapsack problem as follows:

Definition 2.17 (Knapsack Problem) *Let $\{X_1, X_2, \ldots, X_d\}$ be a set of positive integers. Let $c = \sum_1^d s_i X_i$, where $s_i \in \{0, 1\}$. A knapsack problem is given $\{X_i\}$ and c, find each s_i.*

The density of a knapsack, denoted by ρ, is d/β, where β is the maximum bit length of X_i-s.

Definition 2.18 (Modular Knapsack Problem) *Let $\{X_0, X_1, \ldots, X_d\}$ be a set of positive integers. Let $c = \sum_1^d s_i X_i \bmod X_0$, where $s_i \in \{0, 1\}$. A modular knapsack problem is given $\{X_i\}$ and c, find each s_i.*

The knapsack problem is also known as the Subset Sum Problem (SSP) [LO85]. The Sparse Subset Sum Problem (SSSP) that the squashing technique relies on, is a special case of knapsack problem with $\sum s_i \ll d$.

The decisional version of the knapsack problem is NP-complete [Kar72]. However, if its density is too low, there is an efficient reduction to the problem of finding a shortest non-zero vector from a lattice (refer to [Lai01, NS05b, CJL$^+$92]).

In practice, to solve those problems, one runs lattice reduction algorithms (We will provide more details on lattice reductions in the next subsection.) over the following basis:

$$\mathbf{B}_K = \begin{pmatrix} X_1 & 1 & 0 & \dots & 0 \\ X_2 & 0 & 1 & \dots & 0 \\ X_3 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ X_d & 0 & 0 & \dots & 1 \end{pmatrix}$$

We refer to $\mathbf{B}_K$ as the *knapsack-type basis*, and $\mathbf{B}_M$ as the *modular knapsack-type basis*.

$$\mathbf{B}_M = \begin{pmatrix} X_0 & 0 & 0 & \dots & 0 \\ X_1 & 1 & 0 & \dots & 0 \\ X_2 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ X_d & 0 & 0 & \dots & 1 \end{pmatrix}$$

In the rest of the thesis, for simplicity, we focus on knapsack-type basis, although the adoption over a modular knapsack-type basis is straightforward. We also note that a principal ideal lattice basis is a modular knapsack-type basis.

For now, we use lattice reduction algorithms as a black box and illustrate an example of how to recover $\{s_i\}$ via lattice reduction.

Example 2.3 *Considering the following knapsack problem, where* $X = \sum_{i=1}^{9} s_i X_i$.

$X = 1911310173$ $X_1 = 437491759$ $X_2 = 128552629$ $X_3 = 972127522$ $X_4 = 711069765$
$X_5 = 125617110$ $X_6 = 812891076$ $X_7 = 44057509$ $X_8 = 376073782$ $X_9 = 340284326$

To solve this problem, one firstly needs to build a lattice with the following basis:

$$\mathbf{B} = \begin{pmatrix} 1911310173 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 437491759 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 128552629 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 972127522 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 711069765 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 125617110 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 812891076 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 44057509 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 376073782 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 340284326 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Then, by performing a lattice reduction, one obtains a reduced basis. In this example, we use the classic LLL algorithm.

$$\mathsf{LLL}(\mathbf{B}) = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 5 & -3 & -1 & 1 & 0 & 0 & 3 & 1 & 3 & 2 \\ -1 & -5 & 4 & 5 & -1 & 4 & 1 & 0 & -5 & 0 \\ 3 & -4 & 3 & 2 & -4 & 3 & -6 & 4 & -1 & -2 \\ 1 & 2 & -1 & 1 & 7 & 4 & 4 & 3 & -6 & -2 \\ -1 & -4 & -4 & -5 & 3 & 3 & -5 & 1 & 5 & 3 \\ -3 & 1 & -4 & 7 & 5 & -6 & 1 & 3 & -3 & -5 \\ 8 & -1 & 5 & -1 & 4 & 1 & -3 & -4 & 2 & -1 \\ 4 & 2 & 4 & 4 & 4 & -6 & -4 & 5 & 1 & 3 \\ 3 & 2 & -6 & 1 & -2 & 4 & 1 & -4 & -9 & 2 \end{pmatrix}$$

Note that the first vector contains only binary coefficients. Indeed, it indicates that $X = X_1 + X_3 + X_5 + X_8$*, and the problem is solved.*

2.2.4 Lattice Reductions

A lattice is usually given in the form of a basis. However, for a given lattice, there exist an infinite number of bases. Among those bases, some are "good", where the vectors within the basis are almost orthogonal (see Figure 2.4), while some are "bad", where the vectors within the basis are almost parallel (see Figure 2.5). Given a bad basis, to obtain a good basis, is known as lattice reduction [LLL82]. The quality of a basis can be indicated from the Gram-Schmidt orthogonalization (GSO).

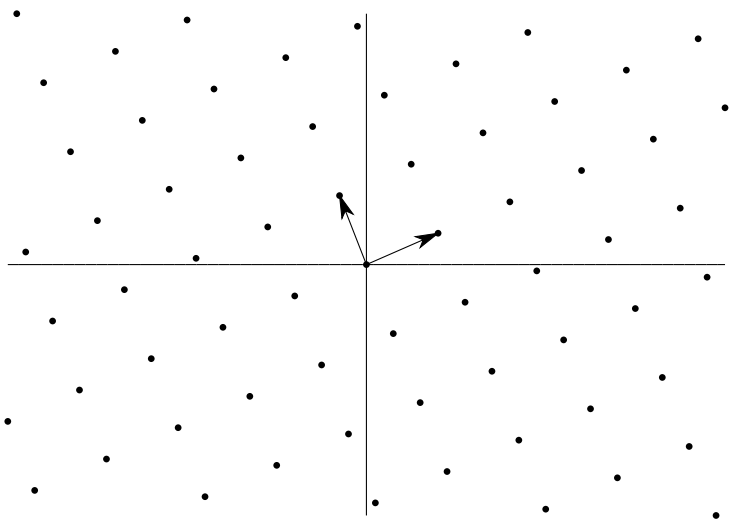

Figure 2.4: A good basis of a 2-dimensional lattice

Definition 2.19 (Gram-Schmidt orthogonalization) *Let* $\mathbf{B} = (\vec{b}_1, \ldots, \vec{b}_d)$ *be a basis of* $\mathcal{L}$*. The Gram-Schmidt orthogonalization (GSO) of* $\mathbf{B}$ *is* $\mathbf{B}^* = (\vec{b}_1^*, \ldots, \vec{b}_d^*)$*:*

$$\begin{aligned}
&\vec{b}_1^* = \vec{b}_1, \\
&\vec{b}_i^* = \vec{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \vec{b}_j^*, && (2 \le i \le d), \\
&\mu_{i,j} = \frac{\vec{b}_i \cdot \vec{b}_j^*}{\vec{b}_j^* \cdot \vec{b}_j^*}.
\end{aligned}$$

Example 2.4 *For the good basis*

$$\mathbf{B}_A = \begin{pmatrix} 8 & 5 \\ -3 & 11 \end{pmatrix}.$$

Its GSO are

$$\mathbf{B}_A^* = \begin{pmatrix} 8 & 5 \\ \frac{-515}{89} & \frac{824}{89} \end{pmatrix}, \qquad \mu_A = \begin{pmatrix} 1 & 0 \\ \frac{31}{89} & 1 \end{pmatrix}.$$

For the bad basis

$$\mathbf{B}_B = \begin{pmatrix} 29 & 31 \\ 21 & 26 \end{pmatrix}.$$

Its GSO are

$$\mathbf{B}_B^* = \begin{pmatrix} 29 & 31 \\ \frac{-3193}{1802} & \frac{2987}{1802} \end{pmatrix}, \qquad \mu_B = \begin{pmatrix} 1 & 0 \\ \frac{1415}{1802} & 1 \end{pmatrix}.$$

$\mathbf{B}_A$ *is a better reduced basis because the maximum is smaller in* $\mathbf{B}_A^*$ *are smaller.*

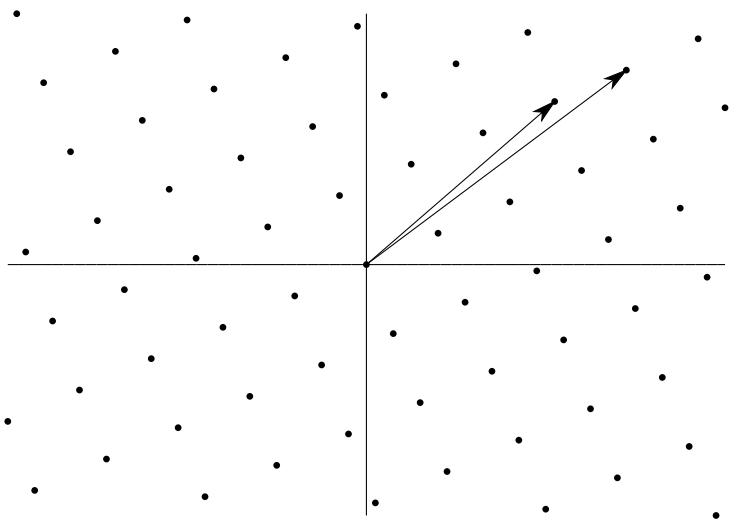

Figure 2.5: A bad basis of a 2-dimensional lattice

Lattice reduction is a very powerful tool, and has many variants. There are polynomial time algorithms, that find vectors that are an exponential approximation of a shortest non-zero vector, and there are also exponential time algorithms, that find the exact shortest vector, or a polynomial approximation of it.

Hence, depending on the quality of the reduced basis, one is able to solve some of the lattice problems listed in the last subsection. And therefore, the cryptosystems that are based on those problems are under concern. The fully homomorphic encryption scheme over ideal lattice is based on a bounded distance decoding problem, while the knapsack problem is based on an approximate shortest vector problem. Both problems adopt very large approximation factors. As a result, one is able to solve the problems using polynomial time algorithms, such as the LLL algorithm.

The LLL Algorithm

In this subsection, we give a general overview of the LLL algorithm. We refer the reader to the book by Nguyen and Vallée [NV09] for a more detailed account.

The LLL algorithm is described in Algorithms 1 and 2. Algorithm 1 is for size reduction, and it is sometimes referred to as the Gram-Schmidt reduction. Algorithm 2 outputs a (δ, η)-reduced basis. The basis is also referred to as the LLL-reduced basis.

Definition 2.20 (η-size reduced) *Let* $\mathbf{B} = (\vec{b}_1, \ldots, \vec{b}_d)$ *be a basis of* $\mathcal{L}$. $\mathbf{B}$ *is* η*-size reduced, if* $|\mu_{i,j}| \leq \eta$ *for* $1 \leq j < i \leq d$. $\eta \geq 0.5$ *is the reduction parameter.*

Definition 2.21 ((δ, η)-reduced basis) *Let* $\mathbf{B} = (\vec{b}_1, \ldots, \vec{b}_d)$ *be a basis of* $\mathcal{L}$. $\mathbf{B}$ *is*

(δ, η)-reduced, if the basis is η-size reduced and it satisfies Lovász condition as follows: $\delta\|\vec{b}_{i-1}^*\|^2 \leq \|\vec{b}_i^* + \mu_{i,i-1}^2\vec{b}_{i-1}^*\|^2$ for $2 \leq i \leq d$. $\frac{1}{4} < \delta \leq 1$ and $\frac{1}{2} \leq \eta < \sqrt{\delta}$ are two reduction parameters.

Algorithm 1 Size Reduction

Input: $\mathbf{B} = (\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_d)$, its GSO, an index κ and a reduction parameter η.
Output: A new basis $\mathbf{B}$, where $\vec{b}_\kappa$ is size reduced, and the updated GSO.

```
for i = (κ − 1) → 1 do
    if μ ≤ η then
        b_κ ← b_κ − ⌈μ_{κ,i}⌋ · b_i;
        Update GSO;
    end if
end for
return B.
```

It is quite straightforward to see that if Algorithm 2 terminates, then its output basis satisfies the Lovász condition. Hence, the basis is (δ, η)-reduced. Note that in the classic LLL, $\eta = 0.5$, while in L^2 (see next subsection), it is essential that η is slightly greater than 0.5.

Algorithm 2 LLL

Input: $\mathbf{B} = (\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_d)$ and reduction parameters (δ, η).
Output: A (δ, η)-reduced basis $\mathbf{B}$.

```
Compute GSO;
κ ← 2;
while κ ≤ d do
    size reduce (b_1, ..., b_κ) with parameter η;
    if δ‖b*_{κ−1}‖² ≤ ‖b_κ‖² + μ²_{κ,κ−1}‖b*_{κ−1}‖² (Lovász condition) then
        κ ← κ + 1;
    else
        Exchange b_κ and b_{κ−1};
        κ ← max(κ − 1, 2);
        Update GSO;
    end if
end while
return B.
```

With respect to the running time of the algorithm, we firstly recall the following two definitions.

Definition 2.22 (Gram determinants) *Let $\mathbf{B} = (\vec{b}_1, \ldots, \vec{b}_d)$ be a basis of $\mathcal{L}$. Let $\mathbf{B}^* = (\vec{b}_1^*, \ldots, \vec{b}_d^*)$ be the corresponding GSO. The Gram determinants of $\mathbf{B}$, noted*

$\{\Delta_1^*, \ldots, \Delta_d^*\}$ *is defined as:*

$$\Delta_i^* = \det(\vec{b}_1^*, \ldots, \vec{b}_i^*) = \prod_{i=1}^{d} \|\vec{b}_i\|.$$

Definition 2.23 (Loop Invariant) *The loop invariant is defined as the product of all Gram determinants as:* $D = \prod_{i=1}^{d-1} \Delta_i^*$.

For any basis, the upper bound of D is $2^{\beta d(d-1)}$, while for principal ideal lattice HNF bases, D is further bounded by $2^{\beta(d-1)}$.

It has been shown [NS06] that the loop invariant D is unchanged except during the exchange procedure, while during the exchange, D is decreased by a factor of δ. Hence, the total number of exchanges cannot exceed $\left|\frac{\beta d(d-1)}{\log_2 \delta}\right|$, which implies there are maximum $O(d^2\beta)$ loop iterations. In addition, since the size reduction algorithm requires $O(d^2)$ operations, the total number of operations is $O(d^4\beta)$. Finally, each operation involves integer multiplications with a cost of $\mathcal{M}(d\beta)$ due to rational arithmetics.[1] Hence, the original LLL algorithm terminates in polynomial time $O(d^6\beta^3)$.

With respect to the quality of a reduced basis for an arbitrary lattice, the following theorem provides an upper bound.

Theorem 2.2 *[NS05a, Ste10] For a lattice* $\mathcal{L}$*, if* $\mathbf{B} = (\vec{b}_1, \ldots, \vec{b}_n)$ *form an LLL-reduced basis of* $\mathcal{L}$*, then,*

$$\forall i, \quad \|\vec{b}_i\| \leq 2^{\frac{d-1}{2}} \lambda_i(\mathcal{L}). \tag{2.2}$$

Hence, if $\mathbf{B} = (\vec{b}_1, \ldots, \vec{b}_d)$ forms an (δ, η)-reduced basis, then $\|\vec{b}_i\| < 2^d \det(\mathcal{L})^{\frac{1}{d}}$ for $1 \leq i \leq d$.

The L^2 algorithm

The most costly part in an LLL procedure is the size reduction. When one performs a size reduction, the GSO needs to be regularly updated. During the update, the classic LLL needs to operate on integers with the length of $O(d\beta)$. As a result, the multiplication of those integers incurs a cost of $\mathcal{M}(d\beta)$.

In [Sch88] and [SE94], Schnorr showed that using floating points instead of integers for LLL can reduce the cost of multiplications from $\mathcal{M}(d\beta)$ to $\mathcal{M}(d+\beta)$. To the best of our knowledge, this is the first time where floating points make a significant

[1] Here, we follow the LLL algorithm by using $\mathcal{M}(d)$ to be $O(d^2)$ assuming a naive integer multiplication, although one can replace it with $O(d^{1+\varepsilon})$ to obtain the exact bit complexity using fast integer arithmetics.

difference in the LLL algorithm. However, it is obseved that the hidden constant in the bit complexity remains huge.

To further improve the efficiency, the L^2 algorithm was proposed by Nguyen and Stehlé [NS05a] in 2005. It is the first variant whose worst-case time complexity is quadratic with respect to β. L^2 uses floating point arithmetics where the multiplication can be carried out with precision $O(d)$. Hence, it reduces the cost of integer multiplication from $\mathcal{M}(d\beta)$ to $\mathcal{M}(d)$. It uses a worst-case time complexity of $O(d^5\beta^2 + d^6\beta)$ to produce a (δ, η)-reduced basis for $\frac{1}{4} < \delta < 1$ and $\frac{1}{2} < \eta < \sqrt{\delta}$.

The L^2 algorithm incorporates the *lazy reduction* as follows:

- one is required to perform $O(1 + \frac{\beta}{d})$ *fp*-reductions to ensure that the vectors is size reduced, since each *fp*-reduction may be incomplete.
- the size reduction consists of $O(1 + \frac{\beta}{d})$ floating point reductions (*fp*-reduction).
- within each *fp*-reduction, one works on floating point whose precision is $O(d)$. As a consequence, the multiplication cost is reduced to $\mathcal{M}(d)$.
- The factor within $O(\cdot)$ is influenced by the reduction parameters. A default setting in the *fplll* is approximately $1.6d$.

As for a principal ideal lattice HNF basis or a knapsack-type basis, it is proved that L^2 terminates in $O(d^4\beta^2 + d^5\beta)$, since there are $O(d\beta)$ loop iterations for these bases instead of $O(d^2\beta)$ for bases of random lattices (see Remark 3, [NS05a]).

In practice, the *fplll* library [PSC] and MAGMA [BCP97] are two well known implementations. Within MAGMA. there exist two main versions, "L2" and "FP" (it is known as "LM_WRAPPER" in the *fplll*). In the *fplll* library, two more variants, known as LM_HEURISTIC" and LM_FAST", are available. However, those two variants do not deliver proved results. For this reason, in this thesis we focus on the first two methods.

The L2 is described as above. It is the proved version of L^2. Meanwhile, in practice, one can further improve the average performance with some heuristics. To the best of our knowledge, the most efficient implementation of L^2 is FP. As far as the floating point is concerned, FP is L2 plus some early reductions.

In FP, the basis is early reduced as follows: the algorithm will choose several fixed precisions subject to the following conditions:

- The arithmetics are fast with those precisions, for instance, 53 for C double precision.

- Reductions with those precisions are likely to produce a correct basis, for instance, d rather than $1.6d$ (see Remark 4, [NS06]).

Reductions with above precisions are cheaper, while they produce somewhat reduced bases. So the algorithm will try all early reductions with different fixed precisions, and finally perform an L2 to ensure the quality of reduction. We note that those early reductions do not change the overall complexity, since in theory the last L2 is still the most costly one. Nevertheless, in practice, the early reductions are very effective to accelerate the whole procedure.

Other LLL Variants

To complete this subsection, we list some of the LLL-type algorithms that improve complexity with respect to β. For other improvements with respect to d, we refer readers to [MSV09, Sch06, KS01].

In [vHN10], van Hoeij and Novocin proposed a gradual sub-lattice reduction algorithm based on LLL that deals with knapsack-type bases. Unlike other LLL-type reduction algorithms, it only produces a basis of a sub-lattice. This algorithm uses a worst-case $O(d^7 + d^3\beta^2)$ time complexity.

In 2011, Novocin, Stehlé and Villard [NSV11] proposed a new improved LLL-type algorithm that is quasi-linear in β. This led to the name $\tilde{\mathrm{L}}^1$. It is guaranteed to terminate in time $O(d^6\beta + d^{\omega+2}\beta^2)$ for any basis, where ω is a valid exponent from matrix multiplications. To bound ω, we have $2 < \omega \leq 3$. A typical setting in [NSV11] is $\omega = 2.3$.

Other Reduction Algorithms

As mentioned earlier, there exist two main types of lattice reduction algorithms. The LLL-type algorithms we have described in the previous subsection use polynomial time and find vectors that are an exponential approximation of a shortest non-zero vector. There are also some stronger lattice reduction algorithms, for instance, HKZ and BKZ, that runs in (sub-)exponential time and find shorter vectors. The cryptanalysis we shall describe in this thesis concerns mainly polynomial time lattice reduction algorithms. Hence, we will only briefly recall those stronger lattice reductions.

The HKZ reduction algorithm, which was proposed by Hermite in 1850, and refined by Korkine and Zolotareff in 1873, is one of the strongest reduction algorithm in terms of the reduced basis its providing. Given an input basis, it starts with finding the exact

shortest within the lattice spanned by the input basis. Then it projects the lattice over a shortest non-zero vector, to obtain a sub-lattice of the original lattice. Note that after the projection, the dimension of the lattice will be reduced by 1. Therefore, after $d-1$ loop iterations, one obtain a strongly reduced basis. However, since it uses many SVP solvers, the running time is exponential in d.

In 1988, Schnorr proposed a Block Korkine-Zolotareff (BKZ) reduction algorithm. Instead of reducing the whole basis as in HKZ, the BKZ algorithm deals with blocks of k vectors. As a result, BKZ allows some exponential computations in terms of k. We remark that LLL can been seen as BKZ with block length of $k = 2$. We refer readers to [HPS11b] for a recent complexity analysis of BKZ and [CN11] for recent development of BKZ.

2.2.5 Lattice-based Cryptography

Now we briefly review some of the most important lattice-based cryptosystems. We refer the readers to [BBD08, NS01] for a more detailed account.

GGH Cryptosystem

GGH [GGH97] cryptosystem is one of the first lattice-based public key cryptosystem. A GGH scheme uses a good basis and a bad basis of a lattice as its secret/public keys. As shown in the figures, basis in Figure 2.4 and 2.5 represents same 2-dimensional lattice. Given a good basis, one is able to generate a bad basis, but not vice versa. This gives us a one-way trapdoor function. Hence, generally speaking, the GGH cryptosystem relies on the following assumption: given a bad basis, there is no efficient algorithm that produces a good basis.

The KeyGen algorithm takes as follows:

- Given security parameters λ, it generates a n-dimensional lattice $\mathcal{L}$ in the form of a good basis $\mathbf{B_{sk}}$.
- Set $\mathbf{B_{pk}} = \mathbf{U} \times \mathbf{B_{sk}}$ that forms a bad basis of $\mathcal{L}$, where $\mathbf{U}$ is a $n \times n$ uni-modular matrix.
- Generate $\mathbf{B_{sk}^{-1}}$.
- Output $\mathbf{sk} = \{\mathbf{B_{sk}^{-1}}\}$ and $\mathbf{pk} = \{\mathbf{B_{pk}}, \epsilon\}$, where ϵ is the permitted error bound.

The Encrypt Algorithm takes as follows:

- For input message $\vec{m} = \langle m_1, m_2, m_3, ..., m_n \rangle$, compute $\vec{v} = \sum_i^n m_i \times \vec{b}_i$, where $\mathbf{B_{pk}} = (\vec{b}_1, \vec{b}_2, \vec{b}_3, \ldots, \vec{b}_n)$.
- Randomly generate a noise vector $\vec{e}$, where $\|\vec{e}\| \leq \epsilon$.
- Output the ciphertext $\vec{c} = \vec{v} + \vec{e}$.

The Decrypt Algorithm takes as follows:

- Compute $\vec{m}' = \vec{c} \times \mathbf{B_{sk}^{-1}} = \vec{m} \times \mathbf{U} + \vec{e} \times \mathbf{B_{sk}^{-1}}$.
- Round off the permitted error $\vec{e} \times \mathbf{B_{sk}^{-1}}$.
- Output $\vec{m} = \lfloor \vec{m}' \rceil \times \mathbf{U}^{-1}$.

Ideally, retrieve a message from a ciphertext without the secret key in GGH is equivalent to solving a CVP problem. However, in 1999, Nguyen [Ngu99] showed that the GGH encryption scheme has a flaw in the design of the schemes. He showed that every ciphertext reveals information about the plaintext and that the problem of decryption could be turned into a special CVP much easier to solve than the general CVP.

Although GGH is broken, it shows us a method to build a one-way trapdoor function using lattice. For instance, in NTRU [HPS98], a good basis is used as a secret, while a bad basis is the public key. To encrypt a message, it maps the message to a point not in the lattice, but within a very short distance less than a permitted error rate. To retrieve the message without a good basis is equivalent to solving a CVP. Hence it is infeasible in polynomial time. However, if one possess such a good basis, then one can

simply decrypt it using a rounding off procedure.

2.3 Fully Homomorphic Encryption

The idea of fully homomorphic encryption was raised by Rivest, Adleman and Dertouzos [RAD78], shortly after the invention of RSA [RSA78]. A homomorphic encryption scheme ξ consists of four algorithms: KeyGen, Encrypt, Decrypt and Eval.

- KeyGen(λ): Input a security parameter λ, it outputs public key **pk**, secret key **sk**.
- Encrypt(m, **pk**): Input a message m and the public key **pk**, it outputs a corresponding ciphertext c.
- Decrypt(c, **sk**): Input a ciphertext c and the secret key **sk**, it outputs a corresponding message m.
- Eval($\mathbf{pk}, c_1, c_2, \ldots, c_n, \mathcal{C}^n$): Input a public key **pk**, n ciphertext $c_1, c_2, \ldots, c_n$ and a permitted circuit $\mathcal{C}^n$, it outputs $\mathcal{C}^n(c_1, c_2, \ldots, c_n)$.

The first three algorithms follow the definition of a public key encryption scheme, while the last one is defined as follows: input a public key **pk**, a set of ciphertexts $\{c_i\}$ whose corresponding messages are $\{m_i\}$, and a circuit $\mathcal{C}$, output another ciphertext c. This evaluation is correct if the following holds:

$$\text{Decrypt}(\text{Eval}(\mathcal{C}, \{c_i\}, \mathbf{pk}), \mathbf{sk}) = C(m_1, \ldots, m_t). \tag{2.3}$$

Definition 2.24 (Homomorphic Encryption) *The scheme ξ = (KeyGen, Encrypt, Decrypt, Eval) is homomorphic for a class $\mathcal{C}$ of circuits if it is correct according to Equation 2.3 for all circuits $C \in \mathcal{C}$. ξ is fully homomorphic if it is correct for all boolean circuits. Further, ξ is compact, if for any circuit $C \in \mathcal{C}$ with a number of inputs polynomial in λ, the size of ciphertexts output by* Eval *is bounded by a fixed value which is polynomial in λ.*

Following this notion, schemes that support partial homomorphism have been proposed. Recently, Gentry [Gen09b, Gen09a] successfully provided a framework for constructing homomorphic encryption schemes (referred to as the Gentry scheme) and,

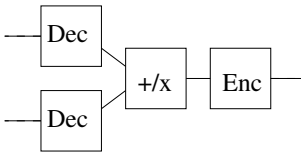

Figure 2.6: $f_{+,\times}$: the two basis operations

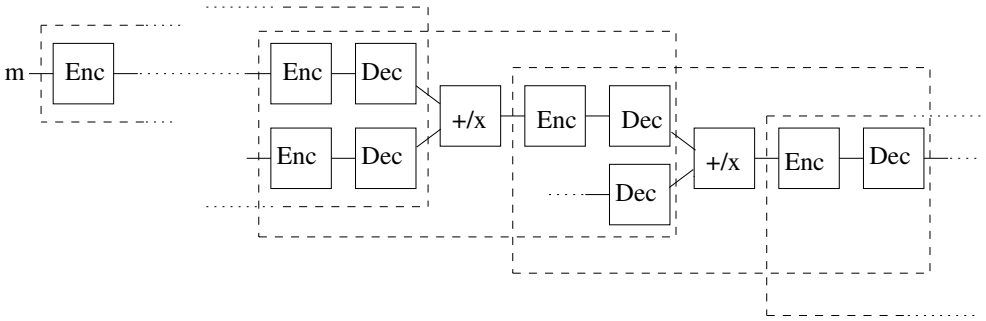

Figure 2.7: Gentry's fully homomorphic encryption model

further, he provided a concrete construction. In addition, subsequent works based on his framework have been proposed recently (such as [SV10, SS10, vDGHV10]). For instance, in [GH11] (referred to as Gentry-Halevi scheme), the authors optimized the performance of the Gentry scheme, while in [vDGHV10] (referred to as vDGHV scheme), the author proposed an integer variant of the Gentry scheme. In the following, for clarity, we will review Gentry's framework.

2.3.1 Gentry's Framework

Gentry's framework for constructing fully homomorphic encryption schemes is based on creating a function to perform two atomic operations which will allow the user to build any kind of circuit. Effectively, any circuit can be built with two atomic functions, namely addition $+$ and multiplication $\times$ over $\mathbb{F}_2$. Therefore, to evaluate any circuit, we are only required to be able to add and multiply over $\mathbb{F}_2$ two encrypted bits.

We note that, to ensure security, such an encryption function is required to be indistinguishable, namely $\text{Enc}(m_0) \neq \text{Enc}(m_1) \not\Rightarrow m_0 \neq m_1$. To build such a function, $\oplus$ and $\otimes$, Gentry used a simple model. Gentry defined the two functions f_+ and $f_\times$ which are equivalent to decrypting both encrypted bits, adding or multiplying such decrypted bits and then encrypting the resulting bits (See Figure 2.6).

However, if f_+ and $f_\times$ return the desired result for $\oplus$ and $\otimes$, the bits are clearly

readable and therefore they do not maintain the intended security requirement.

To achieve this required property, Gentry used an encryption scheme which allows evaluation of short circuits. Therefore, it encrypts the ciphertext with a second cryptosystem. Hence, it can remove the first encryption securely to perform the addition or the multiplication (See Figure 2.7). This technique is named *bootstrapping*.

Definition 2.25 (Bootstrappable Encryption) *Let scheme* $\xi = (\text{KeyGen}, \text{Encrypt}, \text{Decrypt}, \text{Eval})$ *be a compact homomorphic encryption scheme, and let* $\mathcal{C}_\xi$ *be the class of circuits regarding to which the scheme is correct. Denote* D_ξ *its decryption circuit.* ξ *is bootstrappable if* $D_\xi \in \mathcal{C}_\xi$.

Remark 2.1 *Gentry has shown that if a bootstrappable scheme can correctly evaluate bitwise additions and multiplications over two ciphertexts, then this scheme is fully homomorphic [Gen09a].*

Using such a technique, Gentry simplified the quest of constructing a fully homomorphic encryption that can evaluate any circuit on encrypted data by finding an encryption system that can evaluate only some short circuits, namely f_+ and $f_\times$. This encryption system is referred to as a *somewhat homomorphic encryption* system. In formally speaking, to achieve fully homomorphic, we need

$$\text{FHE} = \text{SHE} + \text{Squash(optional)} + \text{boostrapping},$$

where the second step is optional for some SHE schemes whose decryption circuit depth is low.

2.3.2 Gentry's Initial Construction

Here we highlight the important techniques used in the Gentry scheme, and we refer the readers to [Gen09a, Gen09b, Gen10a] for a more detailed account.

The Somewhat Homomorphic Encryption Scheme

We will first recall the somewhat homomorphic encryption scheme as follows. The somewhat homomorphic encryption scheme is a GGH-type cryptosystem [GGH97], i.e. the secret key/public key are "good"/"bad" basis of the lattice, and the underlying lattice problem is a Bounded Distance Decoding problem over ideal lattice (BDDi, see Definition 2.13).

The encryption is to map a message to a vector close to the lattice using the bad basis. To be more specifically, to encrypt a message one uses the following one-way trapdoor function:

$$\vec{c} \leftarrow \vec{m} + \vec{r} \times \mathcal{I} + \vec{g} \times \mathcal{J}$$

where $\vec{r}$ and $\vec{g}$ are randomly chosen. $\mathcal{I}$ and $\mathcal{J}$ are two ideal lattices that are co-prime.

Then, with the good basis, one can perform the vector reduction to recover the message with the following equation:

$$\vec{m} \leftarrow \vec{c} \bmod \mathcal{I} \bmod \mathcal{J}. \tag{2.4}$$

Generally speaking, the encryption maps a message $\vec{m}$ to a residue group of $\mathcal{J}$, with a certain level of noise from $\mathcal{I}$. Given a good basis of $\mathcal{J}$ (as a trapdoor), it is easy to retrieve $\vec{m}$ from $\vec{c}$, if the noise is properly bounded. Meanwhile, the addition or multiplication of elements of the residue group will still fall into the same group, *i.e.* $\vec{c}_1 + \vec{c}_2 = \vec{m}_1 + \vec{m}_2 + (\vec{r}_1 + \vec{r}_2)\mathcal{I} + (\vec{g}_1 + \vec{g}_2)\mathcal{J}$. Therefore, the requirement for "somewhat homomorphic encryption" is fulfilled.

To be more specifically, let $\mathbf{B}_\mathcal{I}$ be a basis of lattice $\mathcal{I}$, $\mathbf{B}^{\mathbf{pk}}_\mathcal{J}$ and $\mathbf{B}^{\mathbf{sk}}_\mathcal{J}$ be a bad and a good basis of lattice $\mathcal{J}$, respectively. The secret key $\vec{v}^{\mathbf{sk}}_\mathcal{J}$ is constructed from $(\mathbf{B}^{\mathbf{sk}}_\mathcal{J})^{-1}$. A message can be encrypted as follows:

$$\vec{c} = \vec{m} + \vec{r} \cdot \mathbf{B}_\mathcal{I} + \vec{g} \cdot \mathbf{B}^{\mathbf{pk}}_\mathcal{J} \tag{2.5}$$

where r and g are randomly selected by the encrypter. A decrypter can therefore retrieve the message using Eq. 2.6:

$$\vec{m} = \vec{c} - \lfloor (\vec{v}^{\mathbf{sk}}_\mathcal{J}) \cdot \vec{c} \rceil \pmod{\mathbf{B}_\mathcal{I}}. \tag{2.6}$$

Theoretically, the matrix-vector multiplications have the same computational depth as integer multiplications. Hence, the decryption maintains the same circuit depth as Eq. 2.4. However, as the tweaked scheme uses a vector instead of a matrix as the secret key, the size of the secret key is reduced and therefore, it reduces the computational complexity of the decryption.

Bootstapping

The proposed encryption scheme is not fully homomorphic yet, since the decryption uses at least one multiplication[2] during the evaluations, the noise (i.e. the distance

[2]Note that $\pmod{\mathbf{B}_\mathcal{I}}$ will incur no computational cost if $\mathcal{I}$ is selected to be all even integers.

between a ciphertext vector and the lattice) grows, and when it exceeds a threshold, the ciphertext cannot be decrypted correctly. As a result, the evaluation circuit depth of the decryption algorithm exceeds the capability of the somewhat homomorphic encryption scheme. To solve this problem, one can to squash the decryption circuit from 1 multiplication to many additions.

The squashing procedure consists of a set of vectors, i.e. $S = \{\vec{t}_0, \vec{t}_1, \ldots, \vec{t}_n\}$. There exists a subgroup of S, namely S', whose sum equals to $\vec{v}_{\mathcal{J}}^{\mathbf{sk}}$, i.e. $\vec{v}_{\mathcal{J}}^{\mathbf{sk}} = \sum_{\vec{v}_i \in S'} \vec{v}_i$. Then, one obtains a new ciphertext $\{c'_i\}_{i=0}^{n}$, where $c'_i = c \times \vec{t}_i$, while the decrypter keeps a new secret key, t, a bit stream which records the sequence of S' from S. The hardness of retrieving the secret key from the set S is based on a Sparse Subset Sum Problem (SSSP). Now the decryption consists only additions as follows:

$$\vec{m} \leftarrow \vec{c} - \lfloor \sum\nolimits_t c'_i \rceil \pmod{\mathbf{B}_{\mathcal{I}}}.$$

Hence, the decryption circuit is simplified.

Since the algorithm now is bootstrappable, one is able to refresh a ciphertext by evaluating its own decryption circuit. Then, the original noise is eliminated and a new noise (much smaller) is induced. By doing this repetitively, one is able to evaluate circuit with any depth, therefore, a fully homomorphic encryption scheme is achieved.

2.3.3 Overview of Other Schemes

In previous sections, we have recalled Gentry's framework and the Gentry scheme based on ideal lattice. Following Gentry's framework, a few FHE schemes are proposed, which can mainly be divided into three categories:

Ideal Lattice-based schemes: One of the first variants proposed shortly after the initial construction was made by Smart and Vercauteren [SV10]. They used the "principal ideal lattice" instead of general ideal lattice. Therefore, they maintained a smaller key size and a simpler encryption/decryption algorithm. However, one major obstacle of this scheme is its inefficient key generation algorithm. Indeed, one is required to find a lattice with a prime determinant, and this criteria is impractical with a large dimension, for instance, 2048, which will lead to a larger determinant, and hence making the probability of the determinant being prime to be smaller. Later, Gentry and Halevi presented an alternative solution to avoid this issue in the Gentry-Halevi implementation [GH11], together with some other optimizations, which is by far the most efficient fully homomorphic encryption scheme using ideal lattice.

Other optimizations on fully homomorphic encryption schemes based on ideal lattice have been proposed by Stehlé and Steinfeld in [SS10]. They improved the efficiency of Gentry's original scheme. Part of their techniques are adopted in [GH11] as well. There is also an improvement proposed by Loftus et al. in [LMSV11], which deals with the CCA-1 security of Gentry-Halevi scheme. Nevertheless, this variant does not improve the efficiency of the system and therefore, the Gentry-Halevi scheme is still regarded as the most efficient fully homomorphic encryption scheme based on ideal lattice.

Integer-based schemes: In [vDGHV10], van Dijk et al. proposed another fully homomorphic encryption scheme (refer to as DGHV scheme) where the somewhat homomorphic scheme is based on the general version of Approximate Greatest Common Divisor of integers (AGCD, see Definition 2.15). In [CMNT11], Coron et al. showed an optimization of this scheme, where the security is based on a partial version of the AGCD problem. The hardness of the partial version was soon re-evaluated in [Ngu11, CN12]. In this thesis, we mainly focus on the first variant of AGCD problem, since it is in general harder to solve.

So far, the best implementation of integer based FHE scheme was presented in [CNT12]. We note that homomorphic encryption schemes based on AGCD problems provide an interesting alternative to the use of ideal lattice, but none of them is as efficient as Gentry and Halevi's scheme.

LWE-based schemes: The state-of-the-art of fully homomorphic encryption schemes are based on the Learning With Error problem (LWE) [BV11a] and Ring-LWE problem [BV11b]. The work in [GHS12a, GHS12b] delivers the best efficiency among all fully homomorphic encryption schemes to date.

The major advantage of using LWE is that one can freely select the moduli [BGV12] and the ring [GHPS12]. By using some moduli with a special form (i.e. $2^n + 1$, where n is an integer), one can use some noise control techniques [BGV12], or omit the second step in the framework [GHS12a]. Unfortunately, to apply this technique over lattice-based schemes is not quite straightforward, since the moduli is the determinant of the lattice, hence, it does not possess such freedom. We also note that there is no known reductions between the Bounded Distance Decoding (BDD) problem over ideal lattice and the learning with error problem yet.

2.3.4 Example I: the Integer-based FHE Scheme

Now we describe the fully homomorphic encryption scheme in more details using the interger-based scheme (referred to as the vDGHV scheme), as an example, since this scheme uses integers rather than ideal lattice, and therefore, it is easier to demonstrate and explain, and later incorporate our idea into.

Following Gentry's frameworks, the vDGHV scheme consists of a somewhat homomorphic encryption scheme (SHE) that supports limited additions and multiplications, and the bootstrapping technique to break such limitation.

The Somewhat Homomorphic Encryption Scheme

We firstly recall the somewhat homomorphic encryption scheme. The somewhat homomorphic encryption scheme consists of four algorithms: KeyGen, Encrypt, Decrypt and Eval.

- KeyGen(λ): Input a security parameter λ, it firstly generates parameters $\{\alpha, \beta, \gamma, t, n\}$ in function of λ. It then generates a secret *odd* integer $p \in (2^{\beta}, 2^{\beta+1})$, n different integers $\{r_i \in [-2^{\alpha}, 2^{\alpha})\}$ and another n different integers $\{g_i \in [0, 2^{\gamma-\beta})\}$, respectively. It finally outputs the public key $\mathbf{pk} = \{x_i = g_i p + 2r_i\}$ and secret key $\mathbf{sk} = \{p\}$.

- Encrypt(m, $\mathbf{pk}$): Input the public key $\mathbf{pk}$ and a message $m \in \{0, 1\}$, it chooses a random subset $\mathbf{s} \subseteq \mathbf{pk}$ and output $c = m + 2r + \sum_{x_i \in \mathbf{s}} x_i \bmod x_0$, where x_0 is the greatest in $\{x_i\}$, $r \in [-2^{\alpha}, 2^{\alpha})$ is a random noise.

- Decrypt(c, $\mathbf{sk}$): Input the secret key $\mathbf{sk} = \{p\}$ and a ciphertext c, it outputs $m = (c \bmod 2) \oplus (\lfloor c/p \rceil \bmod 2)$, where $\lfloor c/p \rceil$ returns the closest integer of c/p.

- Eval(c_1, c_2, ..., c_k, $\mathcal{C}$, $\mathbf{pk}$). It outputs $\mathcal{C}(c_1, c_2, ..., c_k)$, where $\mathcal{C}$ is a k-inputs evaluation polynomial whose circuit depth is lower than the maximum circuit depth allowed by this SHE.

Example 2.5 *This SHE scheme naturally supports homomorphic additions and multiplications, when $\alpha \ll \beta$. We use multiplication as an example.*

- $c_1 = \text{Encrypt}(m_1) = m_1 + g_1 p + 2r_1$ *for centain* g_1, r_1*;*

- $c_2 = \text{ENCRYPT}(m_2) = m_2 + g_2 p + 2r_2$ *for centain* g_2, r_2*;*
- $c_1 c_2 = m_1 m_2 + 2(r_1 m_2 + r_2 m_1 + 2r_1 r_2) + p(g_1 m_2 + 2g_1 r_2 + g_2 m_1 + 2g_2 r_1 + g_1 g_2 p)$*;*
- $\text{DECRYPT}(c_1 c_2) = c_1 c_2 \bmod p \bmod 2 = m_1 m_2$*, as long as* $2(r_1 m_2 + r_2 m_1 + 2r_1 r_2) \in (-p/2, p/2]$.

Therefore, the above SHE scheme is homomorphic.

However, the homomorphic circuit depth is limited, *i.e.*, the noise grows after each operation $2(r_1 m_2 + r_2 m_1 + 2r_1 r_2)$ compared with r_1 or r_2, and eventually it is possible that the absolute value of the noise will be greater than $p/2$ and a decryption error is then generated. The bootstrapping technique is used to break this limitation.

Bootstrapping

Suppose we want to evaluate a circuit whose depth is greater than this SHE permits, we break the circuit into several sub-circuits. For each sub-circuit, the absolute value of resultant noise is less than the threshold $(p/2)$. Then we refresh the resultant ciphertext using the bootstrapping technique. We describe the bootstrapping technique in general. We refer the readers to their original scheme [vDGHV10] for more details.

To bootstrap, firstly, they modify the decryption circuit. As we mentioned earlier, the noise grows significantly faster in a multiplication than in an addition. Therefore, a squashing method is adopted to break the decryption circuit from one multiplication into several additions. The squashing technique is as follows:

- Generate $x = \lfloor 2^\kappa / p \rceil$, where κ is a parameter in λ that is greater than $\beta + 1$.
- Build a bit sequence $\vec{s} = \langle s_1, s_2, \ldots, s_\eta \rangle$, $s_i \in \{0, 1\}$, with $\sum s_i = \theta$. $\vec{s}$ becomes the new secret key.
- Choose n random integers u_i between 0 and $2^{\kappa+1}$, such that $\sum_i^n s_i u_i = x \bmod 2^{\kappa+1}$.
- Set $y_i = u_i / 2^\kappa$. Then $\sum s_i y_i = 1/p + \epsilon$, where ϵ is negligible compared with $1/p$.
- New ciphertext is a vector $\vec{z} = \langle z_1, z_2, \ldots, z_\eta \rangle$, generated by $z_i = [c \times y_i]_2$.
- New decryption circuit becomes $m = [c - \lfloor \sum s_i \times z_i \rceil]_2$

As a result, the decryption circuit now consists only of additions, hence, the growth of noise in additions becomes extremely slow. Then, because the modified decryption circuit depth is relatively low, now it is possible to carry out the decryption circuit homomorphically, through the proposed somewhat homomorphic encryption scheme.

To do so, one encrypts ciphertexts, denoted by $\{\textsc{Encrypt}(z_i)\}$ and the secret keys, denoted by $\{\textsc{Encrypt}(s_i)\}$. Let $\mathcal{C}_D$ be the decryption circuit, then

$$\textsc{Decrypt}(\mathcal{C}_D, \{\textsc{Encrypt}(z_i)\}, \{\textsc{Encrypt}(s_i)\}) = \textsc{Encrypt}(m).$$

This is because firstly $\mathcal{C}_D(\{z_i\}, \{s_i\}) = m$ and secondly, $\mathcal{C}_D$ can be carried out homomorphically. Therefore, we obtain a new ciphertext $\textsc{Encrypt}(m)$.

The new ciphertext, $\textsc{Encrypt}(m)$ has a refreshed noise level (less than 2^α), which means $\textsc{Encrypt}(m)$ can be evaluated again. By doing this repeatedly, we can evaluate circuit with any depth homomorphically. Therefore, a fully homomorphic encryption scheme is achieved.

2.3.5 Example II: The ideal lattice-based SHE and the CCA-1 attack

In this example we aim to recall the CCA-1 attack [LMSV11] against Gentry and Halevi's SHE. For completeness, we recall SHE schemes first. We note that the actual procedure to generate the ideal lattice is much more complicated than this example. We omit the details for simplicity.

KeyGen(λ)

- Let $\{\alpha, \delta, d\}$ be a principal ideal lattice generated by cyclic rotation of $\vec{v}$ over $\mathbb{Z}/f(x)$, $f(x) = x^d + 1$ and d is a power of 2.
- Find the polynomial $w(x)$, such that $w(x) \times v(x) = \delta \bmod f(x)$;
- $\mathbf{sk} \leftarrow w$, where w is one of odd coefficients of $w(x)$;
- $\mathbf{pk} \leftarrow \{\alpha, \delta, \rho\}$, where ρ is the maximum length allowed for noise during encryption.

Encrypt(m, $\mathbf{pk}$)

- Generate a degree $d-1$ polynomial $r(x)$, with coefficients r_i randomly chosen from 0 and ρ);
- $c(x) \leftarrow m + 2 \times r(x)$, where $m \in \{0, 1\}$;
- $c \leftarrow [m + c(\alpha)]_\delta$.

Decrypt(c, $\mathbf{sk}$)

- $m \leftarrow [c \times w]_\delta \bmod 2$.

Now we describe the CCA-1 attack. The Decrypt algorithm for Gentry-Halevi SHE is to evaluate $m \leftarrow [c \times w]_\delta \bmod 2$. This decryption will be valid as long as $[c \times w/\delta] \leq 1/2$. Therefore, for a certain key set (w, δ), the maximum value c' allowed is a fixed integer. The adversary picks several different "ciphertexts", and pass them to the decryption oracle to check if they can be decrypted correctly. Eventually, the attacker will recover the threshold c' which is the maximum integer that can be decrypted correctly. This c' in return gives the attacker w, the secret key.

To stop this attack, Loftus et al. [LMSV11] proposed a ciphertext check procedure. The ciphertext that is to be decrypted, will be "disassembled" into the generating polynomial $c(x)$. Recall that $c(x) = 2 \times r(x) + m$, hence, for valid ciphertexts, $\|c(x)\|_\infty$ is bounded by a certain threshold smaller than T, while for invalid "ciphertexts" (i.e., integers picked by attacker), the corresponding $c(x)$ can have arbitrary coefficients. Therefore, in the latter case, an error $\perp$ is generated, and the decryption stops.

GH Challenge	Lattice Dimension (d)	Hermite Factor	Previous Results[CN11]
Toy	512	1.67^d	30 days
Small	2048	1.14^d	45 years
Medium	8192	1.03^d	68582 years
Large	32768	1.0081^d	None-polynomial

Table 2.1: The Previous Best Results/Prediction on Gentry Halevi's Challenge

2.3.6 The Fully Homomorphic Encryption Challenges

We recall that the fully homomorphic encryption schemes using ideal lattices are based on a well known lattice problem, the Bounded Distance Decoding problem over ideal lattice (BDDi). As stated earlier, this problem can be reduced to some hard lattice problems such as the shortest vector problem or the closest vector problem. Further, the BDDi problem used in lattice-based FHE schemes also relies on a special case of lattice, the ideal lattice, while it is unclear that if a problem over an ideal lattice is as difficult as the corresponding problem over normal lattices. To this end, Gentry and Halevi propose the fully homomorphic challenges [GH].

In the fully homomorphic encryption challenges [GH], the authors published four sets of public keys with respect to different parameters (toy, small, medium and large). To solve the challenge, one needs to recover the secret keys from the public keys. In toy, small and medium challenges, this can be achieved by performing an LLL reduction over the principal ideal lattice bases [CN11]. Since in those lattices, the corresponding Hermite factor is much greater than the upper bound of the LLL algorithm. Therefore, performing an LLL reduction will solve the challenge. As a result, the remaining problem is the running time of the LLL algorithm. The previous best results/prediction for the running time of LLL can be found in table 2.1. In next chapter, we will show three different techniques to improve the performance of the LLL algorithm.

Part I

Improving Lattice Algorithms for Cryptanalysis

Chapter 3

Adaptive Precision Floating Point LLL

To date, the LLL algorithm is one of the most studied lattice basis reduction algorithms in the literature. Among all of its variants, the floating point version, also known as L^2, is the most popular one, due to its efficiency and its practicality. In its classic setting, the floating point precision is a fixed value, determined by the dimension of the input basis at the initiation of the algorithm. We observe that a fixed precision overkills the problem, since one does not require a huge precision to handle the process at the beginning of the reduction.

To this end, we present an adaptive precision floating point LLL algorithm, the *ap-fplll*. We consider both the proven version, L2, and the most efficient version, FP of the L^2 algorithm.

We test our *ap-fplll* with random lattices. In practice, it always faster than the standard version of L^2. When the dimension and/or determinant are sufficiently large, it is also faster than the fastest implementation of L^2. In general, we accelerate the reduction by 20%.

3.1 The algorithm

The LLL algorithm uses a stepping method. For a basis $\mathbf{B} = (\vec{b}_1, \ldots, \vec{b}_d)$, it starts with the first 2 vectors, and then adds 1 vector into the procedure during each step. We notice that, one does not require a floating point precision of $O(d)$ to reduce in the first $d-1$ steps. In fact, for any k vectors, one only requires $O(k)$ precisions. Hence, a possible improvement is to adaptively select the precision according to the number of vectors that are involved. This leads to the adaptive precision floating point LLL algorithm (*ap-fplll*) as shown in Algorithm 3.

Algorithm 3 describes the L2 version of our *ap-fplll* algorithm. k indicates the current vector the algorithm is working on, while k_{max} indicates the maximum number of the vectors that are involved. When k_{max} changes, one is required to reset the

Algorithm 3 Adaptive precision floating point LLL algorithm

Input: $\mathbf{B} = (\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_d)$, reduction parameters (δ, η) and a starting index γ.
Output: An (δ, η)-reduced basis $\mathbf{B}$.

$k \leftarrow 2$, $k_{max} \leftarrow \gamma$.
SetPrecision(γ) and Compute GSO accordingly.
while $k \leq d$ **do**
  Size reduce $(\mathbf{B}, k)$;
  if $\delta\|\vec{b}^*_{k-1}\|^2 \leq \|\vec{b}_k + \mu_{k,k-1}\|\vec{b}^*_{k-1}\|^2$ (Lovász condition) **then**
    $k \leftarrow k + 1$;
    if $k > k_{max}$ **then**
      $k_{max} \leftarrow k$;
      if $k_{max} > \gamma$ **then**
        SetPrecision(k_{max}) and Compute GSO accordingly.
      end if
    end if
  else
    Swap $\vec{b}_k$ and $\vec{b}_{k-1}$;
    $k \leftarrow \max(k - 1, 2)$;
    Update GSO;
  end if
end while
return $\mathbf{B}$.

precision. To obtain the FP version of the algorithm, one conducts early reductions as in *fplll* when k_{max} increases.

We also introduce an index parameter γ due to an implementation issue. For some of the library, there exists a minimum precision for floating point. If the required precision is smaller than this bound, the algorithm will automatically use the bound. In this case, the precision is not $O(d)$, rather it is a fixed value subject to the system. Hence, using an adaptive precision will not reduce the cost of multiplication, rather it will repetitively recompute the GSO. We set γ such that when more than γ vectors are involved, the algorithm will need to use a precision subject to the dimension.

Remark 3.1 *In our algorithm, we follow the* L2 *by setting the precision to be the exact value that is required, i.e.,* $1.6d$*, to deliver a fair comparison. Nevertheless, it is worth pointing out that the* mpfr *library [mpf] (a library* fplll *depends on) operates a floating number as a linked list of blocks of 32 bits (or 64 bits), therefore, it is possible that increasing precisions with respect to the size of the block (the actual size may be smaller than 32 or 64 due to the overhead of storing a floating number) may derive a better performance, since in this case, the GSO will be updated less often.*

3.2 Analysis

3.2.1 Worst-case complexity

Now, we prove that our algorithm uses the same worst-case complexity with L^2.

The reduction part of L^2 algorithm can be seen as our algorithm with a fixed precision of $O(d)$. Therefore, during the reduction part It can never be more costly than L^2. However, our algorithm needs to recompute the GSO for each step, where the GSO is updated partially in L^2. On the worst-case, it can be more costly than L^2 by the cost of computing the GSO.

For each step, the cost of computing GSO is $O(d^2k^2\beta)$. This brings an overall cost of $O(d^5\beta)$, hence it will not affect the worst-case complexity of $O(d^6\beta + d^5\beta^2)$. As a result, the *ap-fplll* uses the same worst-case complexity with L^2.

3.2.2 Average behaviors

We construct the random lattices as in [GM06]. There exist bases of those lattices that are of the following form:

$$\mathbf{B} = \begin{pmatrix} X_1 & 0 & 0 & \dots & 0 \\ X_2 & 1 & 0 & \dots & 0 \\ X_3 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ X_d & 0 & 0 & \dots & 1 \end{pmatrix},$$

where X_1 is a large prime with β bits. X_i-s ($i \neq 1$) are chosen uniformly between 0 and p.

We analyze the average case complexity of our algorithm with the above bases. Since the lattice is a random one, then its minimas λ_i follow Equation 1.

$$\mathbf{B}_k = \begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,k} & 0 & \dots & 0 \\ x_{2,1} & x_{2,2} & \dots & x_{2,k} & 0 & \dots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ x_{k,1} & x_{k,2} & \dots & x_{k,k} & 0 & \dots & 0 \\ X_{k+1} & 0 & \dots & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ X_d & 0 & \dots & 0 & 0 & \dots & 1 \end{pmatrix}$$

For the k-th step ($k > 2$), the basis is shown as above, where $\|\vec{b}_i\| \lesssim 2^{\frac{k-1}{2}} 2^{\frac{\beta}{k-1}}$ for $i < k$ and $\|\vec{b}_k\| \lesssim 2^{\frac{k-2}{2}} 2^{\frac{\beta}{k-2}}$. Hence, the loop invariant for the current step D_k is then bounded by

$$\prod_{i=1}^{k} \|\vec{b}_i\|^{2(k-i+1)} = 2^{k(k-1)^2-1} 2^{2\beta k + \frac{\beta}{(k-1)(k-2)}}.$$

When the k-th step terminates, $\vec{b}_i$ will be reduced to $2^{\frac{k}{2}} 2^{\frac{\beta}{k}}$ for $i \leq k$. Hence, one obtains $O(\beta)$ loop iterations on average cases. We note that this observation is quite natural, since there are $O(d\beta)$ loop iterations in total, hence, on average there are $O(\beta)$ loop iterations for each k.

Let l be the precision to be used in the algorithms. Then for each loop iteration, one needs to perform $O(1 + \frac{\beta}{l})$ floating point reductions, each at a cost of $O(d^2\mathcal{M}(l))$. Since $l = O(k)$, so it will cost $O(d^2\beta \sum_{i=\gamma}^{d}(1 + \frac{\beta}{i})\mathcal{M}(i)))$ that is $\frac{1}{6}c_1 d^5\beta + \frac{1}{2}c_2 d^4\beta^2$ for some constants c_1 and c_2, if we assume $\mathcal{M}(d) = O(d^2)$.

For comparison, we also show the complexity of L^2: $O(d^2\beta \sum_{i=1}^{d}(1+\frac{\beta}{d})\mathcal{M}(d))$ which is $c_1 d^5\beta + c_2 d^4\beta^2$ for the same constants.

It is straightforward to see that our algorithm uses the same bit complexity with L^2. Further, our algorithm wins on both terms. However, the factor $\frac{1}{6}$ on the first term does not make a difference, which is due to the following: firstly, in this case, $\beta < d$ which indicates that for each lazy reduction, it only requires $O(1)$ *fp*-reductions, while our advantage is in fact a faster *fp*-reduction. Hence, our advantage diminishes. Secondly, the cost of recomputing the GSO is also $O(d^5\beta)$ on worst cases as well.

Nevertheless, when $\beta > d$, we anticipate a lot of reductions. In this case, we should be able to accelerate the reduction by a factor between 0 and 50% for L2 (due to the fact that in practice $\mathcal{M}(d) \leq O(d^{1+\varepsilon})$).

As for FP, in practice, it is possible that the early reduction already produces a good basis. It happens a lot when the dimension is small and $\frac{\beta}{d}$ is small. In this case, the adaptive precision will not boost the reduction, since our advantage works on the final procedure, while in the final procedure, the basis is already in a good shape. Nevertheless, when one increases the dimension and/or the $\frac{\beta}{d}$, the adaptive precision will still accelerate the reduction.

3.2.3 Discussion

Our method can be considered as a more aggressive early reduction as in the FP algorithm. Recall that in the implementation of FP, in the early reduction stage, the bases are reduced with some pre-configured precisions for a few rounds, while in our

scheme, the basis are reduced with different precisions for many rounds. It appears that for some input basis this modification will accelerate the reduction. Assuming that for a given basis, there exists a set of best precisions to perform lattice reduction. Then it is safe to assume that both methods somehow overkill the problem. It is quite natural that using our method the computation power is less wasted, since the precisions are closer to the best ones.

However, the downside of our algorithm is that we need to repeatively update the GSO, which could be very costly in practical, while for the early reductions, for certain basis, it is safe to carry our the computation without the GSO. This is the reason that we believe the proposed method might not be the best way to handle the precisions. We believe that the best method, if it ever exist, needs to balance the tradeoff between the computational gain via smaller floating point and the computational loss during the GSO update.

3.3 Implementation

Now we show the implementation results of *ap-fplll*. The tests were conducted with *fplll* library version 4.0 on Xeon E5640 CPUs @ 2.66GHz. The memory was always sufficient since the algorithm only requires a polynomial space. We used the random lattice basis as shown in the last chapter. We set the dimension to 64 and increase it by 32 each time. For each dimension, we set $\beta = 10d$, $20d$, $\cdots$, and generate the bases accordingly. For each dimension/determinant, we tested 10 different bases where the random numbers are generated from different seeds $0 \sim 9$ using the pseudo-random generator of the NTL library [Sho].

We set the index $\gamma = 40$ so that the required precision is strictly greater than 53. Indeed, one can change γ to improve *ap-fplll*. However, to show a fairer comparison, we use the same value for all the tests. The reduction parameter pair is set to $(0.99, 0.51)$ as the default value of *fplll*. This results in a very strongly reduced basis which is in general most useful for cryptanalysis.

We show the implementation results as follows. As one can see from Table 3.1, one can merely observe any difference between two algorithms at the beginning of the tests, although *ap-fplll*-L2 is slightly faster than *fplll*-L2. We believe the reason is that the cost to recompute the GSO is more or less the same as the advantage of using smaller precisions. However, when the dimension grows, the reductions influence the total complexity more importantly compared with the GSO computation, and as a result,

		β							
		$10d$		$20d$		$30d$		$40d$	
		ap-fplll-L2	*fplll*-L2	*ap-fplll*-L2	*fplll*-L2	*ap-fplll*-L2	*fplll*-L2	*ap-fplll*-L2	*fplll*-L2
d	64	5.298	5.742	11.015	12.319	17.719	20.069	23.941	27.212
	96	29.488	30.607	64.643	69.336	104.553	113.513	144.9	158.222
	128	95.445	99.326	221.722	237.136	368.633	409.15	516.012	577.822
	160	234.241	253.711	575.6	636.703	971.459	1149.22	1417.34	1718.84
	192	470.986	522.537	1241.9	1416.41	2278.79	2876.82	3248.4	4184.2
	224	838.059	1003.08	2385.81	2944.07	4386.86	6062.06	6604.18	9238.31
	256	1349	1702.95	4221.74	5452.48	8235.8	11684.3	11942.6	17102.9
	288	2033.29	2561.16	6979.11	9080.97	13481.2	19231.7		
	320	2772.15	3702.22	11007.3	15048.3				
	352	3686.8	5181.06						
	384	4772.02	7175.78						
	416	6087.31	9476.52						
	448	7641.25	12563.9						

Table 3.1: Test Results: *ap-fplll*-L2 vs *fplll*-L2

		β									
		$10d$		$20d$		$30d$		$40d$		$50d$	
		ap-fplll-FP	*fplll*-FP	*ap-fplll*-FP	*fplll*-FP	*ap-fplll*-FP	*fplll*-FP	*ap-fplll*-FP	*fplll*-FP	*ap-fplll*-FP	*fplll*-FP
d	64	1.251	0.968	2.19	1.941	3.302	3.1	4.484	4.275	5.707	5.536
	96	6.195	4.833	12.206	11.488	19.283	18.919	26.769	27.424	34.075	35.028
	128	19.426	16.934	40.471	40.672	66.898	70.512	93.99	102.083	123.823	134.324
	160	48.422	42.133	109.613	113.167	185.293	201.169	268.095	298.629	357.412	418.395
	192	108.432	95.222	264.037	266.864	476.426	533.024	699.073	777.995	982.345	1076.97
	224	220.045	201.6	543.712	626.538	1160.49	1719.1	1922.35	2589.31	2537.65	3705.78
	256	564.462	605.925	1862.08	2829.1	3979.16	5243.41	6013.42	8036.89	8105.54	10185.1
	288	1127.6	1175.76	4557.49	5576.77	8451.64	10784.1	12722.3	16226	16787.2	21073.3
	320	1879.03	1868.19	7826.52	9341.18	15175.8	18896				
	352	2800.87	2896.21	12445.3	14664.5						
	384	4136.41	4123.88								
	416	6223.48	6150.35								
	448	8821.49	8641.27								

Table 3.2: Test Results: *ap-fplll*-FP vs *fplll*-FP

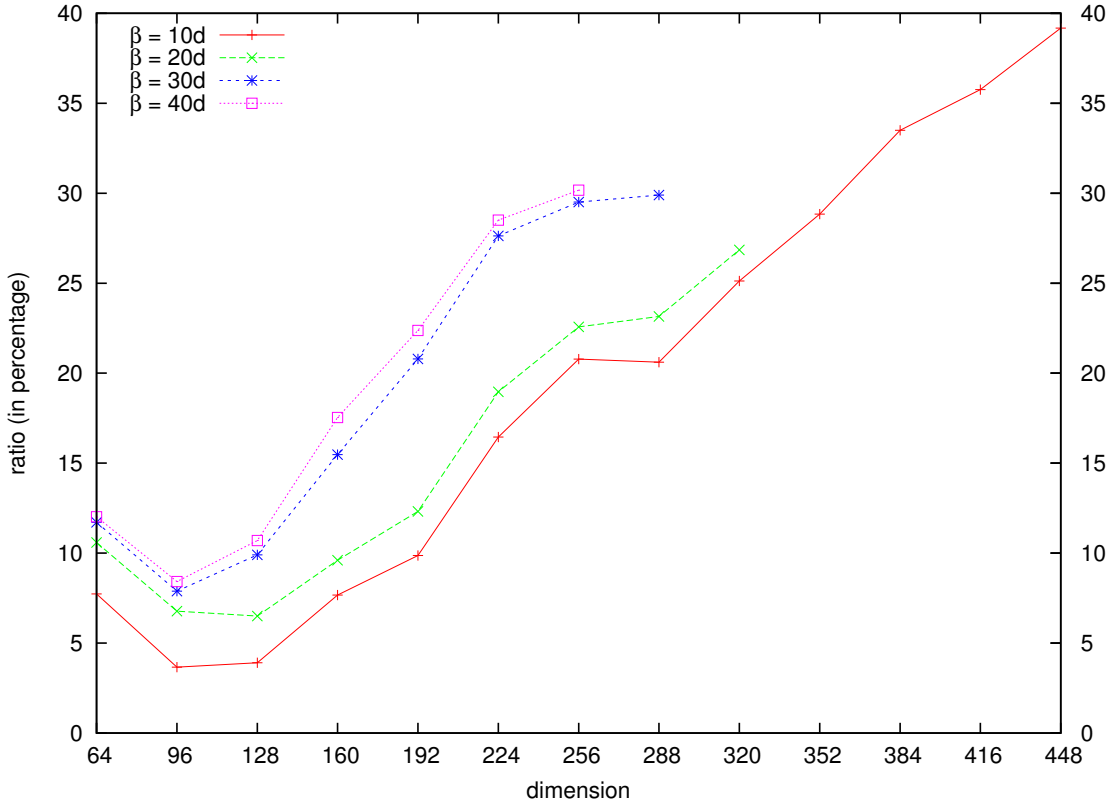

Figure 3.1: Test Results: winning percentage of *ap-fplll* vs *fplll* using L2

the *ap-fplll* starts to be a lot faster. Figure 3.1 illustrated the winning percentage of *ap-fplll*-L2 versus *fplll*-L2. When $d = 64$, we accelerate the reduction by 10%, since it is closer to the starting index $\gamma = 40$. When $d \geq 96$, the influence of γ diminishes, and we start to see the phenomenon where the dimension and/or determinant grow, the advantage increases as well. When the dimension and the determinant are sufficiently large, we can expect an advantage of up to 40%. Overall, our algorithm is always faster in all cases, and we anticipate a boost of over 20% in general.

The results for the FP version are shown in Table 3.2. The results are not as stable as L2 due to the early reductions. As we anticipated, with a small determinant/dimension, i.e., $\beta = 10d$, our algorithm does not accelerate the reduction. The early reduction technique works extremely efficiently in those cases. Nevertheless, the disadvantage is still acceptable considering that even in dimension 448, the disadvantage is less than several minutes.

Meanwhile, for the other cases, when the dimension grows, we start to observe advantages. Further, the advantage rises with the increase of dimension and determinant, just like L2. However, we notice the advantage is not stable. This is because the early reduction affects differently for different dimensions. Overall, as shown in Figure 3.2,

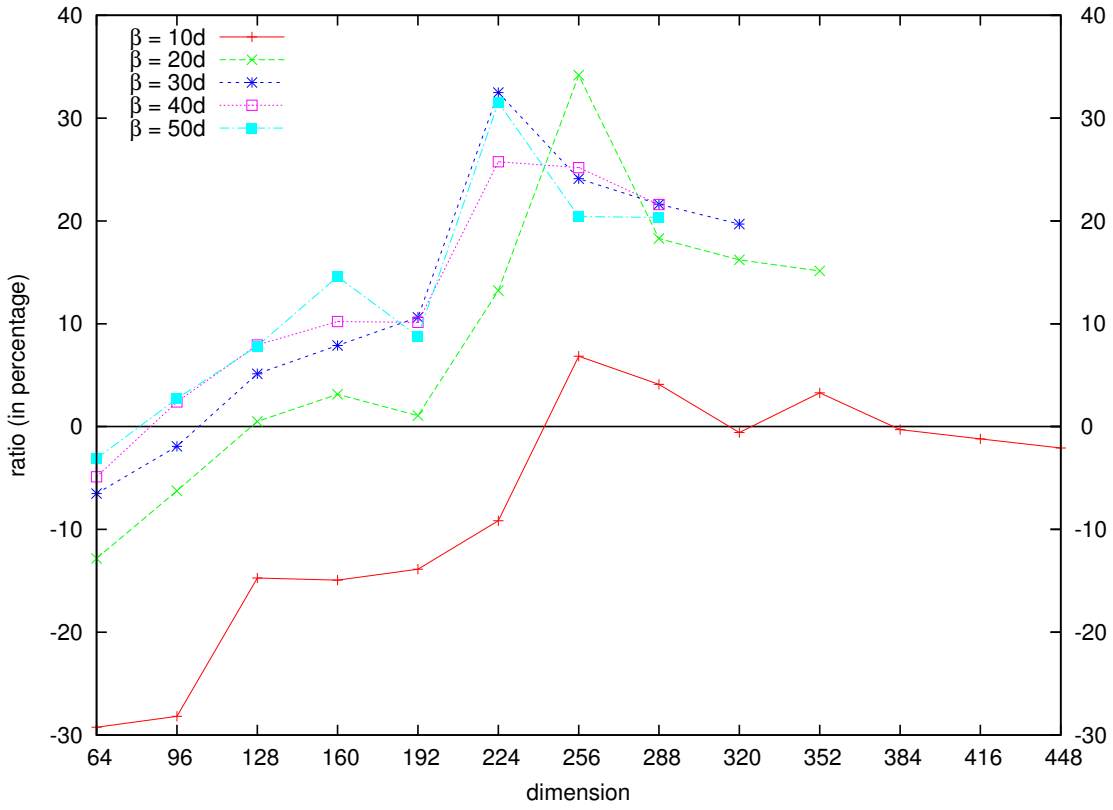

Figure 3.2: Test Results: winning percentage of *ap-fplll* vs *fplll* using FP

as dimension grows, we accelerate the reduction by approximately 20% for $\beta \geq 20d$. In cryptanalysis, one usually needs to deal with lattice with massive dimension and/or determinants, for instance, the Coppersmith-Shamir's technique [CS97] against an NTRU cryptosystem [HPS98], so it is still helpful to use adaptive precisions when $d \geq 128$ and $\beta \geq 20d$.

Chapter 4

Recursive Reduction

In this chapter, we describe a new methodology to adapt any kind of lattice reduction algorithms to deal with the modular knapsack-type basis. As mentioned earlier, the complexity of lattice reduction algorithms to solve those problems is upper-bounded in the function of the lattice dimension d and the maximum number of bits β of the norm of the input basis. In the case of a low density modular knapsack-type basis, the weight of β is mainly from its first column. Therefore, by distributing the weight into multiple columns, it is able to reduce the maximum norm of the input basis. Consequently, the upper bound of the time complexity is reduced.

To show the advantage of our methodology, we incorporate our idea with the floating-point LLL (L^2) algorithm. We bring the complexity from $O(d^{3+\varepsilon}\beta^2 + d^{4+\varepsilon}\beta)$ to $O(d^{2+\varepsilon}\beta^2 + d^{4+\varepsilon}\beta)$ for $\varepsilon < 1$ for the low density knapsack problem, assuming a uniform distribution.

Further, since a principal ideal lattice basis can be seen as a special case of a low density modular knapsack-type basis, we also provide some techniques to deal with a principal ideal lattice basis.

4.1 The Methodology

In this section, we do not propose an algorithm for lattice reduction but rather a methodology applicable to *all* lattice reduction algorithms for the knapsack-type basis with uniform distribution. We assume the following:

Assumption 4.1 *Let $\mathcal{A}$ be an LLL-type reduction algorithm that returns an LLL-reduced basis $\mathbf{B}_{red}$ of a lattice $\mathcal{L}$ of dimension d, where $\mathbf{B}_{red} = (\vec{b}_1, \ldots, \vec{b}_d)$, $0 < \|b_i\| < c_0^d \det(\mathcal{L})^{\frac{1}{d}}$ for certain constant c_0. The running time will be $c_1 d^{a_1}\beta^{b_1} + c_2 d^{a_2}\beta^{b_2}$, where a_1, b_1, a_2 and b_2 are all parameters, c_1 and c_2 are two constants. Without losing generality, assuming $a_1 \geq a_2$, $b_1 \leq b_2$ (if not, then one term will overwhelm the other, and*

hence, making the other term negligible).

We note that this is a formalization of all LLL-type reduction algorithms.

For a knapsack-type basis $\mathbf{B}$ of $\mathcal{L}$, where most of the weight of β is from the first column of the basis matrix $\mathbf{B} = (\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_d)$, it holds that $2^\beta \sim \det(\mathcal{L})$. Moreover, for any sub-lattice $\mathcal{L}_s$ of $\mathcal{L}$ that is spanned by a subset of row vectors $\{\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_d\}$, it is easy to prove that $\det(\mathcal{L}_s) \sim 2^\beta$. In addition, since we assume a uniform distribution, the sub-lattice spanned by the subset of vectors can be seen as a random lattice. Note that the bases of those sub-lattices are knapsack-type bases, so if one needs to ensure the randomness, one is required to add a new vector $\langle X_0, 0, \ldots, 0\rangle$ to the basis and convert it to a modular one. One can verify that this modification will not change the asymptotic complexity. Nevertheless, in practice, it is natural to omit this procedure.

We firstly pre-process the basis, so that the weight is as equally distributed into all columns as possible, and therefore, the maximum norm of the new basis is reduced. Suppose we cut the basis into d/k blocks and each block contains k vectors. Then one applies $\mathcal{A}$ on each block. Since we know that the determinant of each block is $\sim 2^\beta$, this pre-processing gives us a basis with smaller maximum norm $\sim c_0^k 2^{\beta/k}$. Further, since the pre-processed basis and the initial basis span the same lattice, the pre-processing will not affect the quality of reduced basis that a reduction algorithm returns.

Example 4.1 *We show an example of how this methodology works with* 4 *dimensional knapsack-type basis. Let* $\mathbf{B}_A$ *be the basis to process.*

$$\mathbf{B}_A = \left(\begin{array}{ccccc} X_1 & 1 & 0 & 0 & 0 \\ X_2 & 0 & 1 & 0 & 0 \\ \hline X_3 & 0 & 0 & 1 & 0 \\ X_4 & 0 & 0 & 0 & 1 \end{array}\right)$$

Then, we cut $\mathcal{L}$ *into two sub-lattices and pre-process them independently. As a result, we obtain* $\mathbf{B}_B$*.*

$$\mathbf{B}_B = \left(\begin{array}{ccccc} x_{1,1} & x_{1,2} & x_{1,3} & 0 & 0 \\ x_{2,1} & x_{2,2} & x_{2,3} & 0 & 0 \\ \hline x_{3,1} & 0 & 0 & x_{3,4} & x_{3,5} \\ x_{4,1} & 0 & 0 & x_{4,4} & x_{4,5} \end{array}\right)$$

Therefore, $X_i \sim 2^\beta$*, while* $x_{i,j} \lesssim c_0^2 2^{\frac{\beta}{2}}$ *for a classic LLL-type reduction algorithm. Consequently, to reduce* $\mathbf{B}_B$ *is less expensive than* $\mathbf{B}_A$*.*

Now we examine the complexity. The total time complexity of this pre-processing is $c_1 dk^{a_1-1}\beta^{b_1} + c_2 dk^{a_2-1}\beta^{b_2}$. The complexity of the final reduction now becomes $c_1 d^{a_1}(k\log_2(c_0)+\beta/k)^{b_1} + c_2 d^{a_2}(k\log_2(c_0)+\beta/k)^{b_2}$. Therefore, as long as

$$\begin{aligned} c_1 d^{a_1}(k\log_2(c_0)+\beta/k)^{b_1} + c_2 d^{a_2}(k\log_2(c_0)+\beta/k)^{b_2} \\ +c_1 dk^{a_1-1}\beta^{b_1} + c_2 dk^{a_2-1}\beta^{b_2} < c_1 d^{a_1}\beta^{b_1} + c_2 d^{a_2}\beta^{b_2}, \end{aligned} \tag{4.1}$$

conducting the pre-processing will reduce the complexity of whole reduction.

In the case where $k\log_2(c_0)$ is negligible compared with β/k, we obtain:

$$\begin{aligned} c_1 d^{a_1}(\beta/k)^{b_1} + c_2 d^{a_2}(\beta/k)^{b_2} + c_1 dk^{a_1-1}\beta^{b_1} + c_2 dk^{a_2-1}\beta^{b_2} \\ < c_1 d^{a_1}\beta^{b_1} + c_2 d^{a_2}\beta^{b_2}. \end{aligned}$$

Therefore,

$$c_1\left(d^{a_1} - \frac{d^{a_1}}{k^{b_1}} - dk^{a_1-1}\right)\beta^{b_1} + c_2\left(d^{a_2} - \frac{d^{a_2}}{k^{b_2}} - dk^{a_2-1}\right)\beta^{b_2} > 0.$$

Taking L^2 as an example, where $a_1 = 4$, $b_1 = 2$, $a_2 = 5$ and $b_2 = 1$, let $k = d/2$, we obtain $c_1(\frac{7}{8}d^4 - 4d^2)\beta^2 + c_2(\frac{15}{16}d^5 - 2d^4)\beta$ from the left hand side, which is positive for dimension $d > 2$. This indicates that, in theory, when dealing with a knapsack-type basis, one can always achieve a better complexity by cutting the basis into two halves and pre-process them independently. This leads to the recursive reduction in the next chapter.

4.2 The Algorithm

The main idea is to apply our methodology to an input basis recursively, until one arrives at the sub-lattice basis with dimension 2. In doing so, we achieve an upper bounded complexity of $O(d^{a_1-b_1}\beta^{b_1} + d^{a_2-b_2}\beta^{b_2})$. For simplicity, we deal with lattice whose dimension equals to a power of 2, although the same principle is applicable to lattices with arbitrary dimensions.

4.2.1 Algorithm

We now describe our recursive reduction algorithm with LLL-type reduction algorithms. Let $LLL(\cdot)$ be an LLL reduction algorithm that for any lattice basis $\mathbf{B}$, it returns a reduced basis $\mathbf{B}_r$. Algorithm 4 describes our algorithm, where $\mathbf{B}$ is a knapsack-type basis of a d-dimensional lattice, and d is a power of 2.

Since we have proven that, for any dimension of a knapsack-type basis, it is always better to reduce its sub-lattice in advance as long as Equation 4.1 holds, it is straightforward to draw the following conclusion: the best complexity to reduce a knapsack-type basis with LLL-type reduction algorithms occurs when one cuts the basis recursively until one arrives with dimension 2 sub-lattices.

Algorithm 4 Recursive Reduction with LLL algorithm

Input: $\mathbf{B} = (\vec{b}_1, \ldots, \vec{b}_d)$
Output: $\mathbf{B}_r$

```
1: t ← log_2 d        { t is the number of rounds}
2: B_b ← B
3: for i = 1 → t do
4:    n ← 2^i         { n is dimension of the sub-lattice}
5:    k ← d/n         { k is the number of blocks/sub-lattices}
6:    B_r ← EmptyMatrix()
7:    for j = 1 → k do
8:       B_t ← (b_{(j−1)∗n+1}, ..., b_{j∗n})
9:       B_t ← LLL(B_t)
10:      B_r ← VerticalJoin(B_r, B_t)
11:   end for
12:   B_b ← B_r
13: end for
```

In Algorithm 4, the EmptyMatrix() function is to generate an 0 by 0 matrix. The VerticalJoin$(\mathbf{B}_1, \mathbf{B}_2)$ is to adjoin two matrices with the same number of columns vertically.

4.2.2 Complexity

In the following, we prove that the complexity of our algorithm is $O(d^{a_1-b_1}\beta^{b_1} + d^{a_2-b_2}\beta^{b_2})$, assuming that the density ρ of the knapsack is smaller than 1.

For the i-th round, to reduce a single block takes $c_1 2^{ia_1}(\frac{\beta}{2^{i-1}})^{b_1} + c_2 2^{ia_2}(\frac{\beta}{2^{i-1}})^{b_2}$, while

there exist $\frac{d}{2^i}$ such blocks. Hence, the total complexity is as follows:

$$\begin{aligned}
&\sum_{i=1}^{\log_2 d} \left(\frac{d}{2^i}\right) (c_1 2^{ia_1} (\beta/2^{i-1})^{b_1} + c_2 2^{ia_2} (\beta/2^{i-1})^{b_2}) \\
&\quad = d \cdot \sum_{i=1}^{\log_2 d} (c_1 2^{i(a_1-b_1-1)+b_1} \beta^{b_1} + c_2 2^{i(a_2-b_2-1)+b_2} \beta^{b_2}) \\
&\quad = c_1 2^{b_1} d\beta^{b_1} \left(\sum_{i=1}^{\log_2 d} 2^{(a_1-b_1-1)i}\right) + c_2 2^{b_2} d\beta^{b_2} \left(\sum_{i=1}^{\log_2 d} 2^{(a_2-b_2-1)i}\right) \\
&\quad < c_1 2^{b_1} d\beta^{b_1} \left(2^{(\log_2 d+1)(a_1-b_1-1)}\right) + c_2 2^{b_2} d\beta^{b_2} \left(2^{(\log_2 d+1)(a_2-b_2-1)}\right) \\
&\quad < c_1 2^{b_1} d\beta^{b_1} (2d)^{a_1-b_1-1} + c_2 2^{b_2} d\beta^{b_2} (2d)^{a_2-b_2-1} \\
&\quad < c_1 2^{a_1-1} d^{a_1-b_1} \beta^{b_1} + c_2 2^{a_2-1} d^{a_2-b_2} \beta^{b_2}.
\end{aligned}$$

As a result, we obtain a new time complexity $O(d^{a_1-b_1}\beta^{b_1} + d^{a_2-b_2}\beta^{b_2})$.

4.2.3 An example

We describe the application of our method over the classic L^2 algorithm as an example. The L^2 algorithm uses a worst-case complexity of $c_1 d^4\beta^2 + c_2 d^5\beta$ for arbitrary basis. Therefore, applying our recursive methodology, one obtains

$$\begin{aligned}
&\sum_{i=1}^{\log_2 d} \left(\frac{d}{2^i}\right) \left(c_1 2^{4i} \left(\frac{\beta}{2^{i-1}}\right)^2 + c_2 2^{5i} \left(\frac{\beta}{2^{i-1}}\right)\right) \\
&\qquad = \sum_{i=1}^{\log_2 d} (4c_1 d 2^i \beta^2 + 2c_2 d 2^{3i} \beta) \\
&\qquad = 4c_1 d\beta^2 \left(\sum_{i=1}^{\log_2 d} 2^i\right) + 2c_2 d\beta \left(\sum_{i=1}^{\log_2 d} 2^{3i}\right) \\
&\qquad < 4c_1 d\beta^2 (2d) + 2c_2 d\beta 1.15 d^3 \\
&\qquad < 8c_1 d^2\beta^2 + 2.3 c_2 d^4 \beta.
\end{aligned}$$

4.2.4 Discussion

Now we compare our complexity with the original L^2 algorithm. As mentioned earlier, when applying to a knapsack-type basis, the provable worst-case complexity of L^2 becomes $c_1 d^3\beta^2 + c_2 d^4\beta$ rather than $c_1 d^4\beta^2 + c_2 d^5\beta$ as for a random basis. However, it is worth pointing out that in practice, one can achieve a much better result than a worst case, since the weight of most X_i is equally distributed into all the columns.

Heuristically, for the L^2 algorithm, one can expect $\Theta(c_1 d^2 \beta^2)$ when d, β go to infinity and $\beta \gg d$. Nevertheless, this result requires several heuristics. In comparison, our results only requires that the input knapsack-type basis spans a random lattice.

Input a knapsack-type basis, the L^2 algorithm (and almost all other LLL-type reduction algorithms) tries to reduce the first k rows, then the $k+1$ row, $k+2$ row, etc. For a given $k+1$ step, the current basis has the following shape:

$$\mathbf{B}_{knap-L^2} = \begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,k+1} & 0 & 0 & \dots & 0 \\ x_{2,1} & x_{2,2} & \dots & x_{2,k+1} & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ x_{k,1} & x_{k,2} & \dots & x_{k,k+1} & 0 & 0 & \dots & 0 \\ X_{k+1} & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ X_{k+2} & 0 & \dots & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ X_d & 0 & \dots & 0 & 0 & 0 & \dots & 1 \end{pmatrix}$$

L^2 will reduce the first $k+1$ rows during this step. Despite that most of the entries are with small elements ($\|x_{i,j}\| \sim O(2^{\frac{\beta}{k}})$), the worse-case complexity of current step still depends on the last row of current step, i.e., $\langle X_{k+1}, 0, \dots, 0, 1, 0, \dots, 0 \rangle$.

For the recursive reduction, on the final step, the input basis is in the form of:

$$\mathbf{B}_{rec-L^2} = \begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,\frac{d}{2}+1} & 0 & 0 & \dots & 0 \\ x_{2,1} & x_{2,2} & \dots & x_{2,\frac{d}{2}+1} & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ x_{\frac{d}{2},1} & x_{k,2} & \dots & x_{\frac{d}{2},\frac{d}{2}+1} & 0 & 0 & \dots & 0 \\ x_{\frac{d}{2}+1,1} & 0 & \dots & 0 & x_{\frac{d}{2}+1,\frac{d}{2}+2} & x_{\frac{d}{2}+1,\frac{d}{2}+3} & \cdots & x_{\frac{d}{2}+1,d+1} \\ x_{\frac{d}{2}+2,1} & 0 & \dots & 0 & x_{\frac{d}{2}+2,\frac{d}{2}+2} & x_{\frac{d}{2}+2,\frac{d}{2}+3} & \cdots & x_{\frac{d}{2}+2,d+1} \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ x_{d,1} & 0 & \dots & 0 & x_{d,\frac{d}{2}+2} & x_{d,\frac{d}{2}+3} & \cdots & x_{d,d+1} \end{pmatrix}$$

Note that the weight of X_i is equally distributed into $\frac{d}{2}+1$ columns. Hence, the bit length of maximum norm of basis is reduced from β to approximately $d \log_2 c_0 + 2\beta/d$. Therefore, we achieve a better time complexity. In fact, the provable new complexity is of the same level of the heuristic results observed in practice, when $\beta \gg d$.

The cost of previous steps are all negligible, compared with the last step. Thus, if we see those steps as pre-processes, they provide similar functionality with early reductions in L^2. However, our pre-process is in general more cost than early reduction, since in

the early reduction, the algorithm runs with small floating-point precisions (53, for instance), while for us, the precision can be as much as $0.8d$ since half of the basis need to be reduced using proved method. This is the reason that the performance of our algorithm, as we shall see later, lies between the proved method (L2) and the fastest method (L2 with early reduction).

4.3 Extensions

Now we describe a technique when dealing with a principal ideal lattice basis.

Due to the special form of a principal ideal lattice, It is able to reduce the number of reductions in each round to 1, with a cost of $O(d)$ additional vectors for the next round. This technique does not effect the asymptotic complexity, however, in practice, it will accelerate the reduction.

$$\mathbf{B}_I = \begin{pmatrix} \delta & 0 & 0 & \dots & 0 \\ -\alpha \bmod \delta & 1 & 0 & \dots & 0 \\ -\alpha^2 \bmod \delta & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\alpha^{d-1} \bmod \delta & 0 & 0 & \dots & 1 \end{pmatrix}$$

A principal ideal lattice is given in the form of $\{\alpha, \delta, d\}$. Let $X_0 = \delta$, then one obtains $\mathbf{B}_I$ in the above form. From $\mathbf{B}_I$, one constructs a new basis $\mathbf{B}'_I$ as follows.

$$\mathbf{B}'_I = \begin{pmatrix} \delta & 0 & 0 & \dots & 0 & 0 \\ -\alpha & 1 & 0 & \dots & 0 & 0 \\ 0 & -\alpha & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -\alpha & 1 \end{pmatrix}$$

Then, one can obtain a generator matrix of $\mathcal{L}(\mathbf{B}_I)$ by inserting some vectors in $\mathcal{L}$ to $\mathbf{B}'_I$.

Example 4.2 *This example shows how to construct* $\mathbf{G}$ *with* $d = 5$. *The generator*

matrix $\mathbf{G}$ *is given as follows:*

$$\mathbf{G} = \left(\begin{array}{ccccc} \delta & 0 & 0 & 0 & 0 \\ -\alpha & 1 & 0 & 0 & 0 \\ 0 & -\alpha & 1 & 0 & 0 \\ \hline 0 & 0 & \delta & 0 & 0 \\ 0 & 0 & -\alpha & 1 & 0 \\ 0 & 0 & 0 & -\alpha & 1 \end{array}\right)$$

Since vector $\langle 0, 0, \delta, 0, 0\rangle$ *is a valid vector in* $\mathcal{L}(\mathbf{B})$, $\mathbf{B}$ *and* $\mathbf{G}$ *span the same lattice. Applying a lattice reduction algorithm over* $\mathbf{G}$ *will return a matrix with the top row that is a zero vector, while the rest forms a reduced basis of* $\mathcal{L}$.

To reduce $\mathbf{G}$, *we adopt our recursive reduction methodology. We firstly reduce the top half of* $\mathbf{G}$. *Since the second half is identical to the top half, except the position of the elements, we do not need to reduce the second half. Indeed, we use the result of the top half block and then shift all the elements. Finally, we obtain* $\mathbf{G}'$.

$$\mathbf{G}' = \left(\begin{array}{ccccc} x_{1,1} & x_{1,2} & x_{1,3} & 0 & 0 \\ x_{2,1} & x_{2,2} & x_{2,3} & 0 & 0 \\ x_{3,1} & x_{3,2} & x_{3,3} & 0 & 0 \\ \hline 0 & 0 & x_{1,1} & x_{1,2} & x_{1,3} \\ 0 & 0 & x_{2,1} & x_{2,2} & x_{2,3} \\ 0 & 0 & x_{3,1} & x_{3,2} & x_{3,3} \end{array}\right)$$

During our recursive reduction, for round i, instead of doing $d/2^i$ reductions, one needs to perform only one reduction. Finally, one reduces the final matrix $\mathbf{G}$, removes all the zero vectors and starts a new round.

With our technique, the number of vectors grows, and this may increase the complexity of the next round. For the i-th round, the number of vectors grows by $d/2^{i-1}-1$. It will be negligible when $d/2^i \ll d$. For instance, if we adopt this approach between the second last round and the last round, this approach will only increase the number of vectors by 1, while if one uses it prior to the first round, the number of rows will be almost doubled. In practice, one can choose to adopt this technique for each round only when it accelerates the reduction.

We note that the asymptotic complexity remains the same, since generally speaking, the number of vectors remains $O(d)$ as before, while the asymptotic complexity concerns only d and β.

Dimension		4	8	16	32	64	128
L2	Best-case	177s	660s	52.6m	340.8m	36.5h	255.6h
	Avg-case	223.5s	854s	74.3m	445.7m	44.4h	278.7h
FP	Best-case	17s	90s	10.6m	106.3m	22.7h	255.1h
	Avg-case	17.8s	97s	13.8m	153.4m	28.1h	307.9h
Our method	Best-case	26s	258s	23.5m	218.6m	33.8h	194.9h
	Avg-case	37s	213s	33.7m	302.5m	43.7h	271.4h

Table 4.1: Comparison between different algorithms

4.4 Implementation

We implemented our method with L^2. We use L^2 algorithm from *fplll* library [PSC]. The implementation was conducted on DELL Poweredge 1435 Servers powered by AMD Opteron CPU running at 2.3 GHz. For all the test, for each d, β pair, we randomly generated 16 knapsack bases (X_i-s are generated through a pseudo-random generator feed by time seeds using NTL library [Sho]).

The whole implementation can be separated into two parts. In the first part, we compared our technique with the classic L^2 (referred to as L2) and its heuristic variant FP, (classic L^2 with early reduction option and several other optimizations, see [PSC, NS05a]). Generally speaking, as mentioned in the previous chapter, FP tries heuristics first, and if they fail, it will start the L2.

We fixed $\beta \sim 2^{19.5}$ and increased d gradually[2]. Table 4.1 illustrates the result of the first part.

In general, our algorithm runs faster than L2. It is also observed that the FP is the fastest among the three till dimension 64. However, in dimension 128, we believe that heuristics failed and hence became a burden for the FP. Therefore, the FP is a bit slower than L2. Since our method is always faster than L2, it is safe to draw the following conclusion: *our algorithm will be the fastest one among the three after dimension 128.*

In the second part, we aim to prove the average case complexity by showing that our complexity is $O(d^{2+\varepsilon}\beta^2)$ when $\beta \geq d^2$. We increased β and d. We fed the program with β equals to 2^{10}, 2^{14}, 2^{17} and $2^{19.5}$, respectively, and record the reduction time for dimension from 2^2 to 2^7.

In Figure 4.1, the x-axis is $\log_2(d)$, while the y-axis is the logarithm of time in seconds. Therefore, in theory the curve of timing versus dimension is a straight line. Since for our tests other than 2^{10}, $\beta \geq d^2$, the overwhelming complexity should be $O(d^{2+\varepsilon}\beta^2)$. As a consequence, all the lines should be in parallel, and the distance

[2]In Gentry-Halevi's FHE scheme, $\beta = 2^{19.5}$ for small dimension.

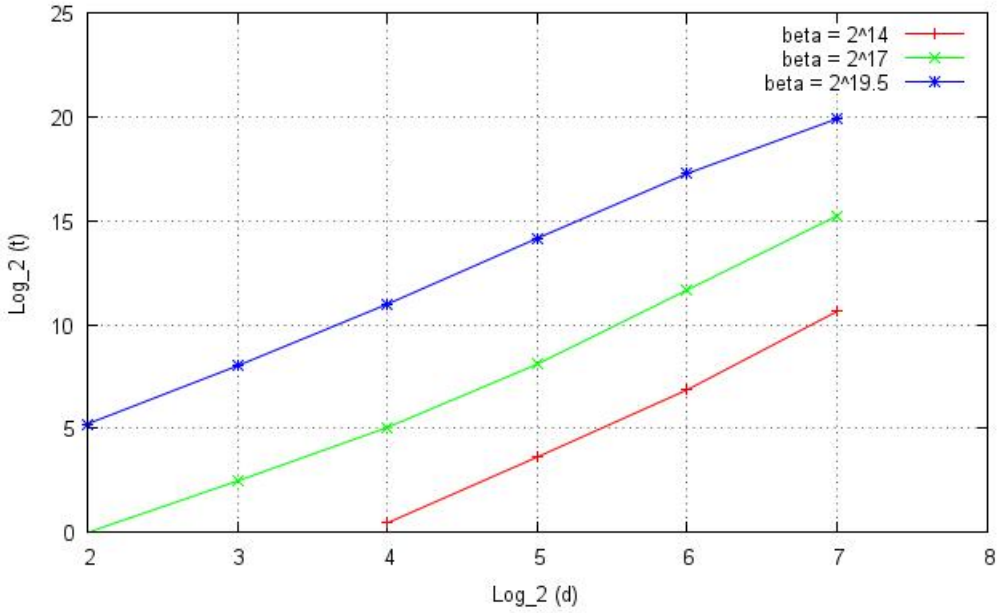

Figure 4.1: Testing results: Time vs Dimension

between each two should be in respect to approximately twice the difference of β. Our test result is consistent with this theory. Moreover, the gradient of the line is the exponential factor on d, hence it should be a real number in 2 and 3. This is also observed in our implementation result.

To sum up, our implementation result shows: *the average-case time complexity of our recursive reduction is correct.*

Chapter 5

LLL for Ideal Lattice

In the previous two sections, we have shown improvements of LLL algorithms over some general bases. In this chapter, we will describe the first variant of LLL algorithm that is dedicated to ideal lattices. The results can be summarized as follows:

- In theory, our algorithm shares the same worst-case bit complexity with the LLL algorithm. But our algorithm is at least as fast as the corresponding LLL algorithm.
- Heuristically, we reduce the complexity from $O((d^3\beta + d^2\beta^2)\mathcal{M}(d))$ (as in L^2) to $O((d^3\beta + d\beta^2)\mathcal{M}(d))$, where $\mathcal{M}(d)$ is the cost of integer multiplication with two d bit integers.
- In practice, our algorithm out-performs all known lattice reduction algorithms in terms of running time.
- In terms of the quality of the output basis, our algorithm produces an LLL reduced basis. Furthermore, in a vast majority of cases, our modification will not affect the output of the corresponding algorithm. With the same input, our iLLL algorithm will output the same result as the LLL algorithm in most tests.
- This leads us to a new result to the Gentry and Halevi fully homomorphic encryption challenge. We solved the toy challenge within 24 days, while we estimate to solve the small challenge in 15.7 years. This result will be presented in the next chapter.

5.1 The Algorithm

We start with an HNF basis of the ideal lattice. We remark that such a basis can be obtained from any basis of the same lattice within polynomial time. The HNF basis

can be obtained straightforwardly if the ideal lattice is a principal one. Nevertheless, the running time of the HNF procedure is negligible compare with the LLL reduction for a given basis.

Ideal lattice maintains this special property: if a vector $\vec{v} \in \mathcal{L}$, then all its rotation vectors over R exist in this lattice as well, where R is the ring. This useful property can accelerate the reduction.

As stated earlier, the LLL algorithm uses a stepping method. At the κ-th step, the first κ vectors in the basis are involved. For a certain step κ, the top $\kappa - 1$ vectors are (δ, η)-reduced. So it will first size-reduce $\vec{b}_\kappa$ with $(\vec{b}'_1, \ldots, \vec{b}'_{\kappa-1})$, where $(\vec{b}'_1, \ldots, \vec{b}'_{\kappa-1})$ denotes the reduced basis of $(\vec{b}_1, \ldots, \vec{b}_{\kappa-1})$, and then perform the LLL reduction on the whole κ vectors.

However, instead of size-reducing the input vector $\vec{b}_\kappa$ whose bit-length is in β, one can use a rotation of a previous vector, providing that this new vector, denoted as $\vec{v}$, together with $\{\vec{b}_i\}$, $1 \leq i \leq d, i \neq \kappa$ also form a basis of $\mathcal{L}$. We name this technique "re-use".

Recall that in the $\kappa - 1$ step, one has already performed a size reduction on $\vec{b}_{\kappa-1}$ with $(\vec{b}'_1, \ldots, \vec{b}'_{\kappa-2})$. Denote $\vec{v}'$ the reduced vector. Then one can simply shift $\vec{v}'$ to the right to obtain $\vec{v}$. $(\vec{b}'_1, \ldots, \vec{b}'_{\kappa-1}, \vec{v})$ also form a basis of $\mathcal{L}(\vec{b}_1, \ldots, \vec{b}_\kappa)$. Moreover, since $\vec{v}$ is already size-reduced to some extent, it will always be shorter than $\vec{b}_\kappa$, and as a result, re-use will always accelerate the reduction.

To use the re-use technique recursively, we obtain iLLL algorithm. In the following, we first show our iLLL algorithm.

Algorithm 5 The iLLL Algorithm

Input: The HNF basis $\mathbf{B} = (\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_d)$ of an ideal lattice and reduction parameters (δ, η)

Output: An (δ, η)-reduced basis $\mathbf{B}$.

Compute GSO.
$\kappa \leftarrow 2$, $\kappa_1 \leftarrow 2$.
while $\kappa \leq d$ **do**
  Size reduce $(\mathbf{B}, \kappa, \eta)$;
  if $\kappa = \kappa_1$ **and** $\kappa < d$ **then**
    $\vec{v} \leftarrow$ Right shift $(\vec{b}_\kappa)$;
    if $\|\vec{v}\| < \|\vec{b}_{\kappa+1}\|$ **then**
      $\vec{b}_{\kappa+1} \leftarrow \vec{v}$
    end if
    $\kappa_1 \leftarrow \kappa_1 + 1$;
  end if
  if $\delta\|\vec{b}^*_{\kappa-1}\|^2 \leq \|\vec{b}^*_\kappa\|^2 + \mu^2_{\kappa,\kappa-1}\|\vec{b}^*_{\kappa-1}\|^2$ **then**
    $\kappa \leftarrow \kappa + 1$;
  else
    Exchange $\vec{b}_\kappa$ and $\vec{b}_{\kappa-1}$;
    $\kappa \leftarrow \max(\kappa - 1, 2)$;
    Update GSO;
  end if
end while
return $\mathbf{B}$.

The iLLL algorithm is described in Alg. 5. We note that the only difference between Alg. 5 and the LLL algorithm is the re-use technique. In Alg. 5, κ indicates the current vector that iLLL is working on. κ_1 indicates if the current size-reduced vector should be re-used later.

5.2 Analysis

5.2.1 Correctness

We firstly prove that the algorithm is correct. To start with, for the i-th step, we have the following Lemma:

Lemma 5.1 *Let* $\mathbf{B} = (\vec{b}_1, \ldots, \vec{b}_d)$. *Let* $\mathbf{B}_{i-1} = (\vec{b}'_1, \ldots, \vec{b}'_{i-1})$ *where* $(\vec{b}'_1, \ldots, \vec{b}'_{i-1})$ *forms a* (δ, η)*-reduced basis of* $\mathcal{L}(\vec{b}_1, \ldots, \vec{b}_{i-1})$. *Let* $\vec{v} \in \mathcal{L}$ *where the* i*-th coefficient of* $\vec{v}$ *is* 1. *Then* $\mathbf{B}' = (\vec{b}'_1, \ldots, \vec{b}'_{i-1}, \vec{v}, \vec{b}_{i+1}, \ldots, \vec{b}_d)$ *form a basis of* $\mathcal{L}(\mathbf{B})$.

Proof: Firstly, all row vectors of $\mathbf{B}'$ can be obtained by linear operations of row vectors of $\mathbf{B}$. Meanwhile, all row vectors of $\mathbf{B}'$ are linearly independent. $\vec{b}'_1, \ldots, \vec{b}'_{i-1}, \vec{b}_{i+1}, \ldots, \vec{b}_d$

are linear independent since the top $i-1$ vectors are LLL reduced form $\vec{b}_1, \ldots, \vec{b}_{i-1}$. Also, $\vec{v}$ is independent with $\vec{b}'_1, \ldots, \vec{b}'_{i-1}, \vec{b}_{i+1}, \ldots, \vec{b}_d$ since the i-th element of all row vectors of $\mathbf{B}'$ is 0 except for $\vec{v}$. Thus, lattice $\mathcal{L}(\mathbf{B}')$ is contained in lattice $\mathcal{L}(\mathbf{B})$. Further, since $\mathcal{L}(\mathbf{B}')$ and $\mathcal{L}(\mathbf{B})$ have the same rank and determinant, they are a same lattice. Hence, $\mathbf{B}'$ is a basis of $\mathcal{L}(\mathbf{B})$. □

Then we prove the correctness in Theorem 5.2.

Theorem 5.2 *For an input basis* $\mathbf{B}$*, our iLLL algorithm outputs an LLL-reduced basis of* $\mathcal{L}(B)$*.*

Proof: For the quality of the basis, it is quite straightforward that our algorithm produces an LLL-reduced basis, since it checks Lovász condition at the last iteration. Furthermore, we have shown that our algorithm produces a basis that spans the same lattice as the input basis in Lemma 5.1, since the shifted vector is the only one whose $(\kappa+1)$-th coefficient is non-zero. Hence, our iLLL algorithm outputs an LLL-reduced basis of $\mathcal{L}(B)$. □

5.2.2 Worst-case Complexity

Our algorithm shares the same worst-case complexity with the LLL algorithm, however, we show that our algorithm is in theory always faster than LLL algorithm. Further, heuristically, we obtain a complexity of $O(d^5\beta + d^3\beta^2)$.

Recall that the LLL algorithm uses a stepping method. For the κ-th step ($\kappa > 2$), the basis is of the following form:

$$\mathbf{B}_{\kappa,LLL} = \begin{pmatrix} x_{1,1} & x_{1,2} & \ldots & x_{1,\kappa} & 0 & 0 & \ldots & 0 \\ x_{2,1} & x_{2,2} & \ldots & x_{2,\kappa} & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & 0 & \cdots & \vdots \\ x_{\kappa-1,1} & x_{\kappa-1,2} & \ldots & x_{\kappa-1,\kappa} & 0 & 0 & \ldots & 0 \\ x_{\kappa,1} & x_{\kappa,2} & \ldots & x_{\kappa,\kappa} & 0 & 0 & \ldots & 0 \\ X_{\kappa+1} & 0 & \ldots & 0 & 1 & 0 & \ldots & 0 \\ X_{\kappa+2} & 0 & \ldots & 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ X_d & 0 & \ldots & 0 & 0 & 0 & \ldots & 1 \end{pmatrix}$$

where the top κ vectors are LLL-reduced. Then, for the next step, since the top κ vectors are already reduced, there will be no exchange initially. The LLL will directly size-reduce $\vec{b}_{\kappa+1}$, and then operate on $(\vec{b}_1, \ldots, \vec{b}_{\kappa+1})$.

$$\mathbf{B}_{\kappa,iLLL} = \begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,\kappa} & 0 & 0 & \dots & 0 \\ x_{2,1} & x_{2,2} & \dots & x_{2,\kappa} & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots & \vdots \\ x_{\kappa-1,1} & x_{\kappa-1,2} & \dots & x_{\kappa-1,\kappa} & 0 & 0 & \dots & 0 \\ x_{\kappa,1} & x_{\kappa,2} & \dots & x_{\kappa,\kappa} & 0 & 0 & \dots & 0 \\ 0 & v_1 & \dots & v_{\kappa-1} & 1 & 0 & \dots & 0 \\ X_{\kappa+2} & 0 & \dots & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots & \vdots \\ X_d & 0 & \dots & 0 & 0 & 0 & \dots & 1 \end{pmatrix}$$

For comparison, the only change we have made is to replace $\vec{b}_{\kappa+1}$ with $\vec{v}$ as shown in $\mathbf{B}_{\kappa,iLLL}$. This modification indeed accelerates the size-reduction, since $\vec{v}$ is in general significantly shorter than $\vec{b}_{\kappa+1}$. Moreover, since our algorithm does not work on large coefficients (i.e. $X_{\kappa+1}$), it requires less precision of floating-point to size-reduce.

To sum up, in theory, we proved that the iLLL will always be faster than the LLL algorithm due to the fact that $\vec{v}$ is not longer than $\vec{b}_{\kappa+1}$. However, in worst cases, it is possible that $\|\vec{v}\| \sim \|\vec{b}_{\kappa+1}\|$ if all κ vectors are not well reduced. In this case, we share the same worst-case complexity as LLL algorithm. It is also worth pointing out that it can never be more costly than LLL, since if $\|\vec{v}\| > \|\vec{b}_{\kappa+1}\|$, one simply does not adopt the re-use, and we obtain an exact LLL algorithm.

5.2.3 Heuristic Complexity

Now we analyze the heuristic complexity of our algorithm. Heuristically, correlated matrix coefficients are different, the HNF bases of ideal lattices have the same shape as the bases of random lattices [GM06], for instance, the coefficients are of similar length. Thus, from cryptanalysis point of view, they delivers similar results. If a lattice is random, then its minima λ_i follow Equation 2.1. We assume the same property for our input basis.

For the κ-th step ($\kappa > 2$), the basis our algorithm is shown as in the last section, where $\|\vec{b}_i\| \sim 2^{\frac{\kappa-1}{2}} 2^{\frac{\beta}{\kappa-1}}$ for $i < \kappa$ and $\|\vec{b}_\kappa\| \sim 2^{\frac{\kappa-2}{2}} 2^{\frac{\beta}{\kappa-2}}$. The loop invariant for current step D_κ is then bounded by $\prod_{i=1}^{\kappa} \|\vec{b}_i\|^{2(\kappa-i+1)} = 2^{\kappa(\kappa-1)^2-1} 2^{2\beta\kappa+\frac{\beta}{(\kappa-1)(\kappa-2)}}$. When the κ-th step terminates, $\vec{b}_i$ will be reduced to $2^{\frac{\kappa}{2}} 2^{\frac{\beta}{\kappa}}$ for $i \leq \kappa$. Hence, one obtains $O(\beta)$ loop iterations for each step. We note that this observation is quite natural, since there are $O(d\beta)$ loop iterations in total, hence, there are $O(\beta)$ loop iterations for each κ.

Algorithms	Time Complexity
LLL[LLL82]	$O(d^{5+\varepsilon}\beta^{2+\varepsilon})$
LLL for ideal lattice	$O(d^{4+\varepsilon}\beta^{2+\varepsilon})$
L^2[NS05a]	$O(d^{4+\varepsilon}\beta^2 + d^{5+\varepsilon}\beta)$
L^2 for ideal lattice[NS05a]	$O(d^{3+\varepsilon}\beta^2 + d^{4+\varepsilon}\beta)$
$\tilde{L}^1$[NSV11]	$O(d^{\omega+1+\varepsilon}\beta^{1+\varepsilon} + d^{5+\varepsilon}\beta)$
iLLL (Heuristic)	$O(d^{2+\varepsilon}\beta^2 + d^{4+\varepsilon}\beta)$

Table 5.1: Comparison of time complexity

1. For the κ-th step, the κ-th step terminated in $O(\beta(\kappa^2+\beta)\mathcal{M}(\kappa))$ operations:
   (a) There are maximum $O(\beta)$ loop iterations.
   (b) For each loop iteration, there is maximum $O(1+\frac{\beta}{\kappa(\kappa-1)})$ iterations within the size reduction.
   (c) In each size reduction, there are $O(\kappa^2)$ arithmetic operations.
   (d) The cost of arithmetic operations is determined by integer multiplications with bit length $O(\kappa)$.
2. Assuming a naive integer multiplication, one obtains $O(\sum_{\kappa=2}^{d}(\beta\kappa^4+\beta^2\kappa^2)) = O(d^5\beta + d^3\beta^2)$.

Table 5.1 shows a comparison of time complexity between iLLL and some of LLL-type algorithms using fast multiplications.

5.3 Extensions

The main improvement in our provable algorithm is to replace $\vec{b}_{\kappa+1}$ with a vector from previous reductions through the size reduction algorithm. In fact, any vector in the lattice can be used as the replacement, as long as the last non-zero element of this vector is 1. The remaining issue is to find $\vec{v}$ more efficiently than size-reduce $\vec{b}_{\kappa+1}$ with $(\vec{b}_1, \ldots \vec{b}_\kappa)$. Algorithm 6 describes a probabilistic yet very efficient method to find $\vec{v}$.

This algorithm first checks to see if it can directly use the first vector from the previous step with simple shifting. It requires the last non-zero element of $\vec{b}_1$ to be 1. The successful rate is high when β is small and κ is big.

If it fails, then it checks if one can construct a vector by linear combination of two vectors. This can be done using the extended GCD algorithm. In the re-use technique,

Algorithm 6 Heuristic Re-use

Input: $(\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_\kappa)$ a (δ, η)-reduced basis
Output: $\vec{v}$ that can be used for the $\kappa + 1$ step of iLLL.

if The last non-zero coefficient of $\vec{b}_1 = 1$ **then**
  Shift $\vec{b}_1$ such that the last coefficient is 1.
  $\vec{v} \leftarrow \vec{b}_1$
else
  for $i = 1 \rightarrow \kappa$ **do**
    Shift $\vec{b}_i$ such that the last coefficient is non-zero.
  end for
  $\vec{v} \leftarrow$ zero vector
  for $i = 1 \rightarrow \kappa$ **do**
    for $j = i + 1 \rightarrow \kappa$ **do**
      Find x, y and z such that $x = y\vec{b}_{i,\kappa} + z\vec{b}_{j,\kappa}$
      if $x = 1$ **then**
        $\vec{v}' \leftarrow y\vec{b}_i + z\vec{b}_j$.
        if $\vec{v} \neq 0$ **and** $\|\vec{v}\| \geq \|\vec{v}'\|$ **then**
          $\vec{v} \leftarrow \vec{v}'$
        end if
      end if
    end for
  end for
end if
return $\vec{v}$.

it finds all possible vectors where the last non-zero coefficient is 1, and return the shortest one.

This procedure takes $O(\kappa^2\mathcal{M}(\beta))$ on worst-cases reductions and $O(\kappa^2\mathcal{M}(\beta/\kappa))$ heuristically. We note it is negligible in terms of time complexity, compared with the cost of each iteration. In practice, it can be done in less than 1 second for dimensions as large as 500. Hence, it finds a vector much faster than using size-reduction algorithm. However, we note that if one uses this optimization, when the first vector does not qualify, it is possible that the returned vector will be slightly longer than the one form size-reduction. Hence, it will make the next step a bit more costly. We also note that this technique is heuristic, since sometimes it is possible that one cannot find a suitable vector. Nonetheless, our practical tests show that this method is in general faster than iLLL.

In fact, our algorithm can also deal with bases other than principal ideal bases. For instance, general ideal lattices or Coppersmith-Shamir bases, which are used to attack the NTRU encryption scheme.

It is true that for some ideal lattice HNF bases, the diagonal coefficients do not follow the same form as a principal ideal lattice. The first several coefficients on the diagonal are not 1. One can use our technique when 1 starts to appear. It should appear very soon, since the diagonal coefficients are decreasing rapidly to 1 with the increase in dimension.

$$\mathbf{B}_{CS} = \left(\begin{array}{cccccccc} q & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & q & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \dots & q & 0 & 0 & \dots & 0 \\ \hline h_0 & h_1 & \dots & h_{N-1} & 1 & 0 & \dots & 0 \\ h_{N-1} & h_0 & \dots & h_{N-2} & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ h_1 & h_2 & \dots & h_0 & 0 & 0 & \dots & 1 \end{array}\right)$$

The Coppersmith-Shamir basis are of the above form. It is not a principal ideal lattice basis, however, its bottom part does allow one to apply our re-use technique, since the right non-zero coefficient is 1 and the left part of the basis is a rotated basis.

To this end, we formally define a reusable basis as a basis where our re-use technique can be applied.

Definition 5.1 (i-Reusable Basis) $\mathbf{B} = (\vec{b}_1, \dots, \vec{b}_d)$ *be a basis of* $\mathcal{L}$. *Let* $i < d$ *an*

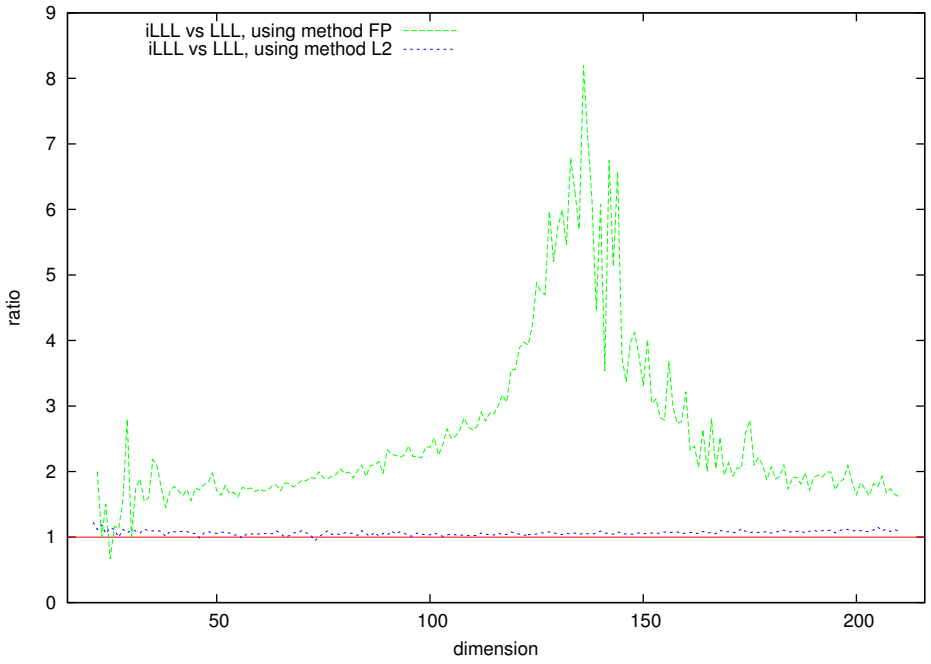

Figure 5.1: Testing results: $\beta = 10d$

integer. **B** *is a reusable basis if* $\forall \kappa > i$*, there exist vectors* $\vec{v}, \vec{v}' \in \mathbb{Z}^n$ *and a permutation matrix* $\vec{P} \in \mathbb{Z}^{n \times n}$ *with regard to the following:*

- $\vec{v} = \vec{v}'\vec{P}$*;*
- $\vec{v}'$ *is a linear combination of* $(\vec{b}_1, \ldots, \vec{b}_{\kappa-1})$
- $\mathcal{L}(\vec{b}_1, \ldots, \vec{b}_{\kappa-1}, \vec{v}) = \mathcal{L}(\vec{b}_1, \ldots, \vec{b}_{\kappa})$.

It is quite straightforward to see that all ideal lattice HNF bases are reusable by simple shifting, while the Coppersmith-Shamir basis is an n-reusable basis with certain permutation. Hence, our iLLL is also applicable to these bases.

5.4 Implementation

In this section, we show some implementation results. The implementation was conducted with MAGMA [BCP97] on Xeon E5640 CPUs @ 2.66GHz. The memory was always sufficient since the algorithm only requires a polynomial space. We first show the average behavior of our algorithm by comparing us with L^2 and FP on bases of random ideal lattice. Subsequently, we apply iLLL to Gentry-Halevi's fully homomorphic

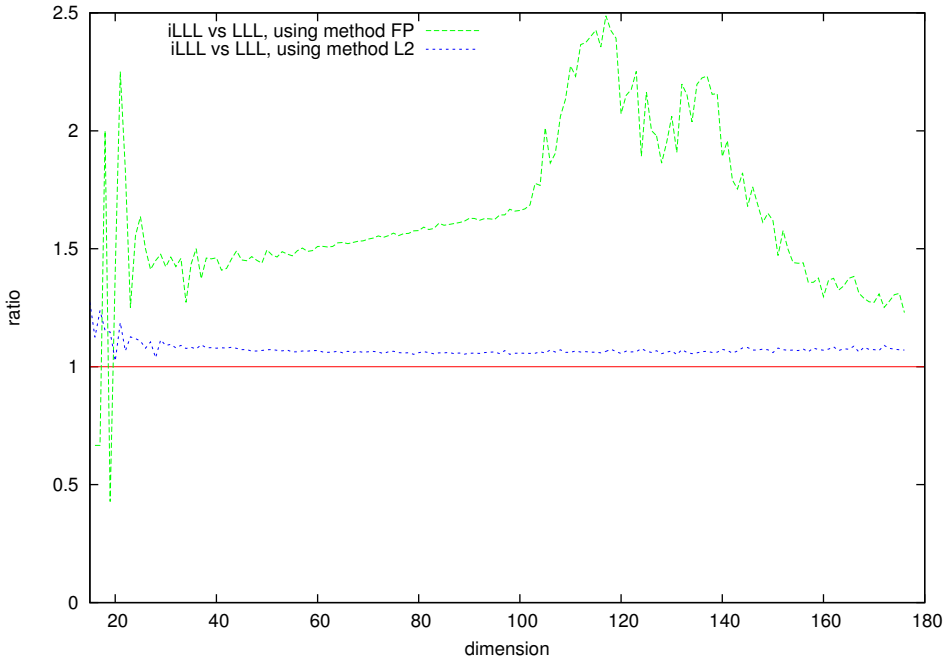

Figure 5.2: Testing results: $\beta = 20d$

encryption challenge, and present the results. Finally we summarize our advantage in practice.

5.4.1 Test Results

We tested our algorithm with bases of random ideal lattice over three scenarios: $\beta \sim 10d$, $\beta \sim 20d$ and $\beta \sim 380d$. The first scenario for $\beta \sim 10d$ is the classical setting for the SVP challenges [svp], and the last one $\beta \sim 380d$ is the requirement for Gentry-Halevi's fully homomorphic encryption scheme. For comparison, we also tested $\beta \sim 20d$ to observe the difference. To ensure the randomness of the ideal lattice, we require α to be a uniform distribution between 1 and γ.

For each dimension of each test, we generated 10 bases with 10 different seeds. Then we present the average time to reduce the bases using both L^2 and FP methods.

Figure 5.1, 5.2 and 5.3 show the advantage for each scenario. The ratio is computed from the running time of LLL divide by the running time of iLLL. As observed, the curves are always above one, which implies that iLLL is always faster than the counterpart. We observe a small advantage for L^2 as we expected, and the advantage is stable for all three scenarios, while it grows with the increase of β/d. With $\beta \sim 380d$, iLLL is 20% faster. The time difference in these cases is due to the re-use technique.

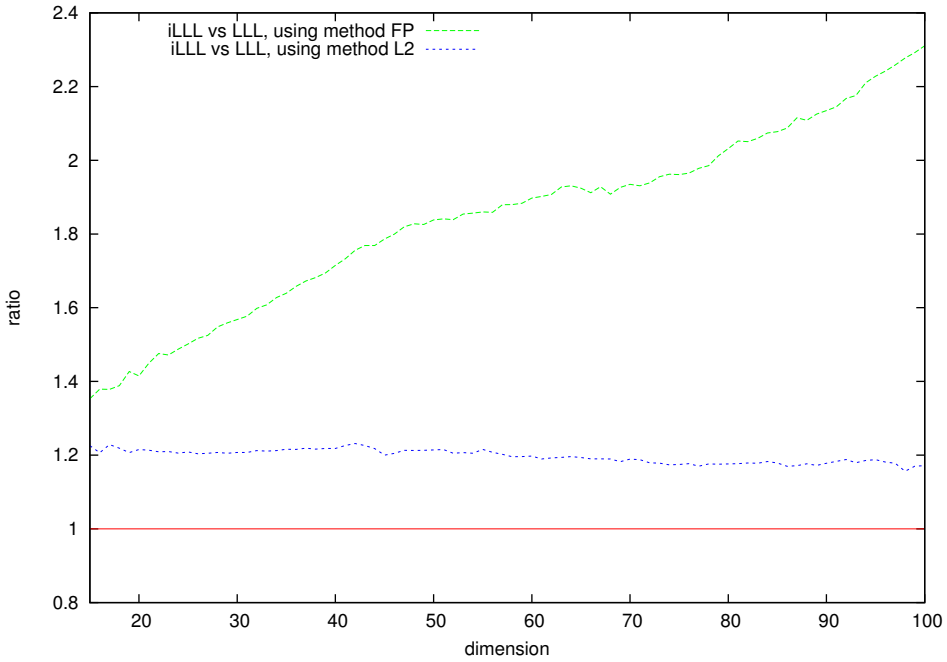

Figure 5.3: Testing results: $\beta = 380d$

In another words, our improvement is due to the unnecessary size reduction in LLL. As the increase of β/d, the length of the reusable vector increases, which results in an increasing advantage.

As for the FP method, apart from the starting point and dimension between 100 to 150, iLLL can be approximately twice faster than LLL. When $100 < d < 150$, we enjoy a massive advantage. We do not present the result for $\beta \sim 380d$ in those dimensions, instead we refer the reader to Figure 6.4, which is essentially using the same setting. In some cases, when $d \sim 150$ and $\beta \sim 10d$, iLLL can be as much as 8 times faster than LLL. This is due to the floating-point setting in FP. We shall discuss this in Subsection 6.2.

Part II

New Cryptanalysis on FHE schemes

Chapter 6

On Gentry and Halevi's Challenge

6.1 Results

The algorithms we have presented in the previous chapter are motivated by the Gentry and Halevi's fully homomorphic challenge. Now we present the results of applying those algorithms over the challenge. To solve the challenge, we use techniques in both *ap-fplll* and iLLL. We tested iLLL on the Gentry-Halevi's fully homomorphic encryption challenge for dimension 512 and dimension 2048. For the dimension 512, the lattice basis was obtained from the challenge website [GH]. For the dimension 2048, we generated the basis as per Gentry and Halevi's paper [GH11], since the basis was not available from the website. Figure 6.1 and 6.2 show the results for dimension 512 and dimension 2048, respectively. We also include the test results for our heuristic method, which accelerates the reduction even further.

As observed from Figure 6.1, iLLL is faster than LLL in practice for the small challenges. The classic FP finishes in 32 days. In comparison, with iLLL we are able to finish within 28 days. In addition, our heuristics accelerate a bit further to 24 days. Overall, we are around 33% faster than the classic FP algorithm.

As for the 2048 challenge, within 6 months, we are able to reach dimension 559 and 560 for iLLL and its heuristics, respectively, while LLL reaches dimension 512. Furthermore, as the dimension grows, the gap between the running time of iLLL and LLL grows as well, which indicates the time that iLLL gains is increasing as the dimension grows. Nevertheless, we anticipate to accelerate the reduction by 30%, which is the same case as the toy challenge. However, due to the fact that the size reduction of a single vector becomes less important as the dimension grows, compared with the whole cost of a single step, the actual ratio of advantage is diminishing.

Figure 6.4 shows a comparison of the running time of iLLL and LLL algorithms. The green curve shows the ratio between the total running time of LLL and iLLL up

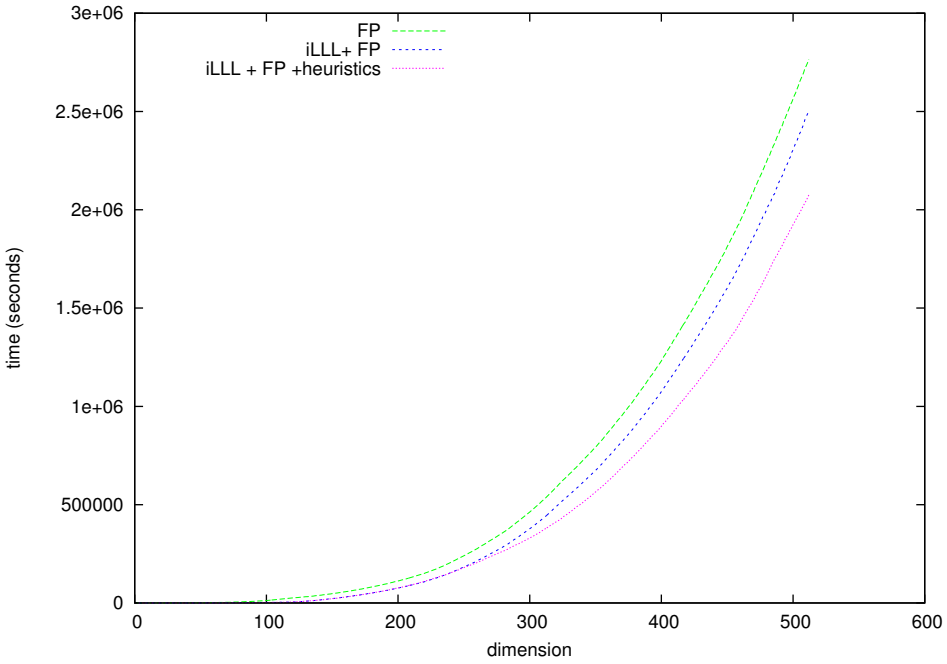

Figure 6.1: Toy challenge

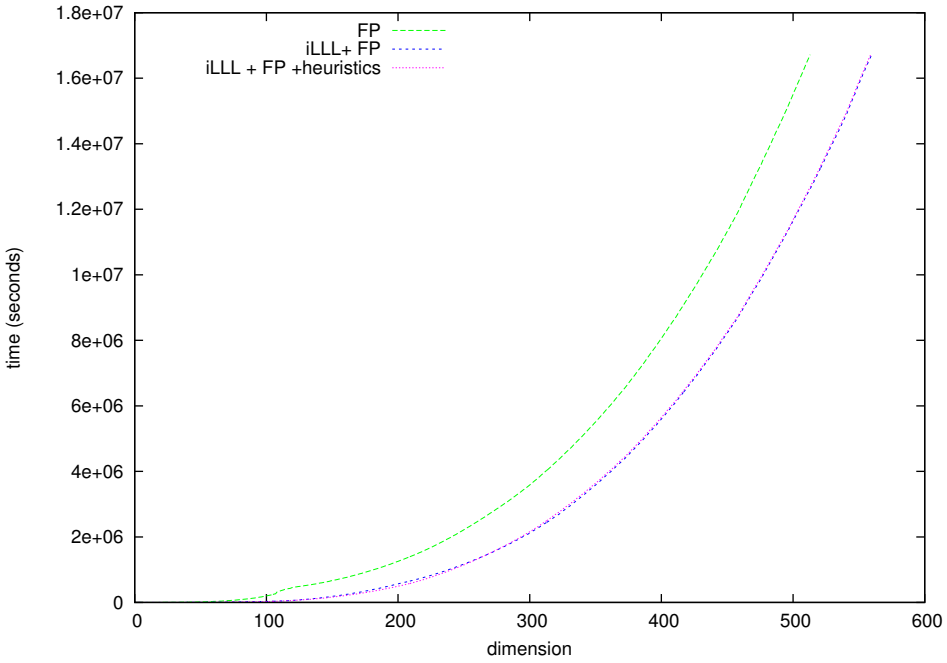

Figure 6.2: Small challenge

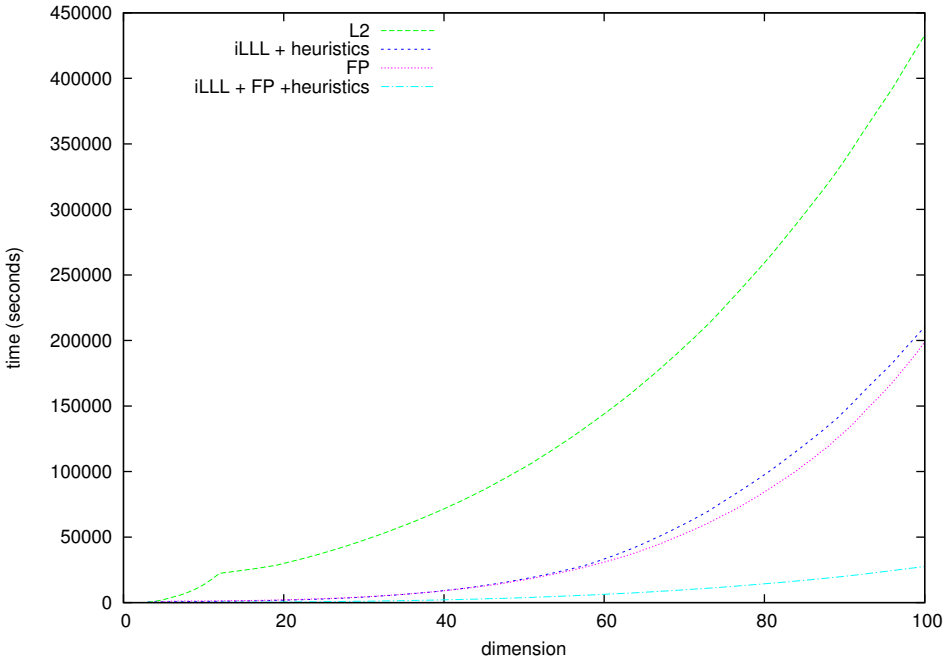

Figure 6.3: Comparison of L2 and FP for Small Challenge

to a certain dimension, while the blue curve shows LLL vs iLLL for each dimension. Since the curves is always higher than 1, it is straightforward to see that iLLL is always faster than LLL. It is also worth pointing out that at dimension around 100, iLLL can be 10 times faster than LLL.

6.2 Analysis

In the previous section, we have seen that the iLLL algorithm is faster than the LLL algorithm in practice. If we look at each step (Figure 6.5 shows one example of this), one can see that in each step, iLLL is faster. This is due to the fact that instead of reducing a vector of the length β at the κ-th step, we can use a vector of length β/κ by simply re-using the previous results. The result follows our theoretical analysis. As stated before, although the time we are able to gain continues to increase, the advantage is actually diminishing as the dimension grows. This is because the cost of size-reducing the large vector is less and less important compare to the cost of reducing the whole basis as the dimension grows.

Interestingly, we enjoy a massive advantage when the dimension is less than 150. A similar phenomenon is also observed in the tests with random lattices. This is

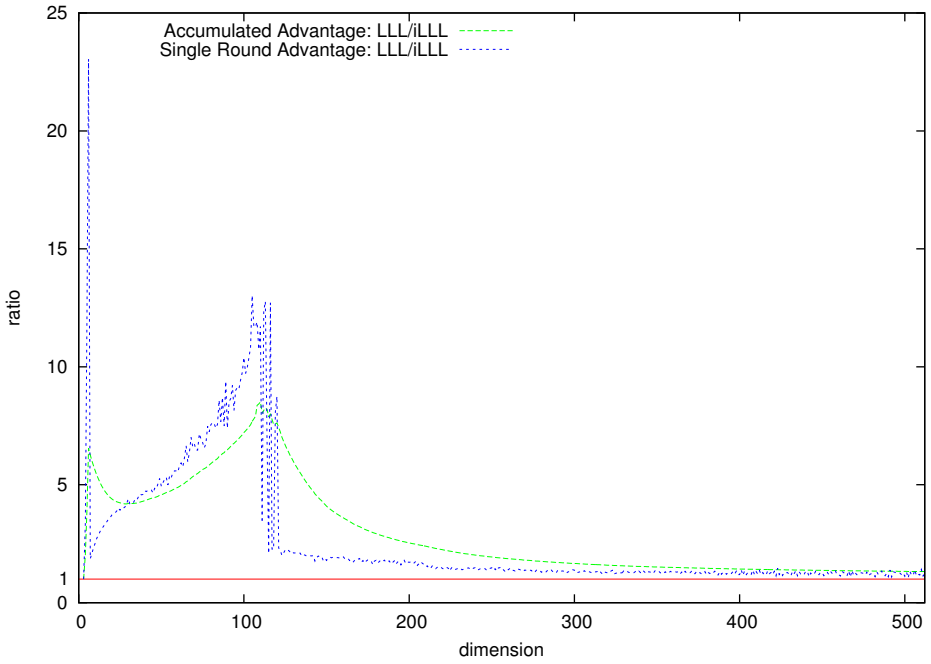

Figure 6.4: Timing results for small challenge: LLL vs iLLL

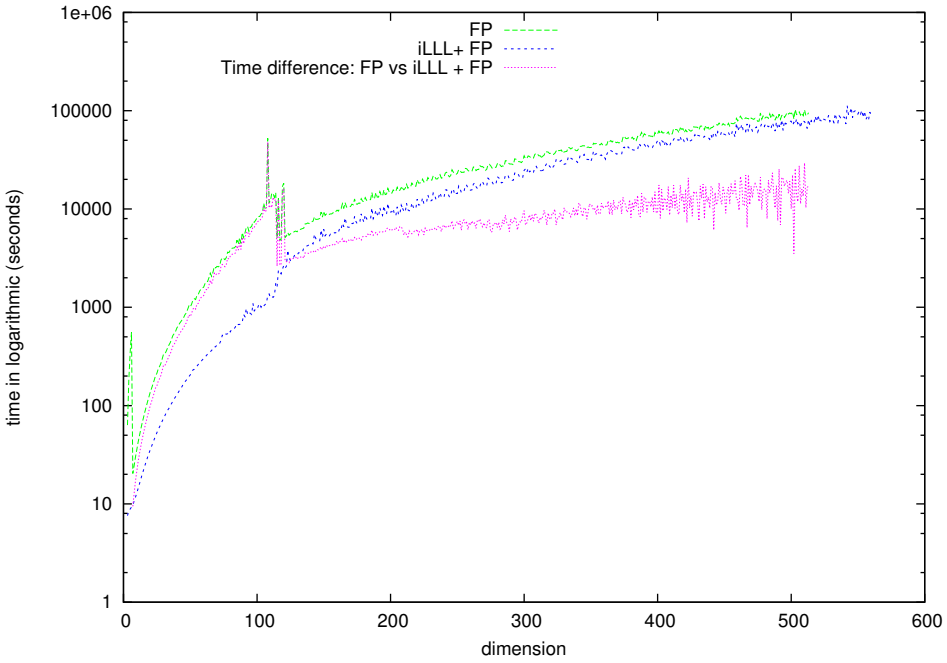

Figure 6.5: Time of each step for dimension 2048

mainly due to the implementation of L^2. As mentioned in $ap-fplll$, L^2 uses a floating point precision ℓ which is in function of d. However, in practice, FP uses several levels of precisions. It starts with the smallest precision, 53, which is with respect to C programming language of type double/int in the program. It then increases the precision to the minimum value of ℓ and the precision for the next stage. This procedure will be continued until it reaches ℓ.

Since the cost relies heavily on the length of floating-points, this optimization guarantees that the reduction is generally performed with the lowest cost.

It must be noted, however, that since the coefficients of the basis for each new step is large, FP will need to use a large precision to provide provable reductions. As a comparison, since we are dealing with small coefficients, when the dimension is smaller, we only require a default precision of 53. This is the reason our algorithm is faster in the beginning.

6.3 New Estimation

In this section, we show our new estimation for the small challenge of Gentry-Halevi's fully homomorphic encryption. We start with predicting L2 as in Figure 6.6. We predict that the curve follows $\frac{d^{2.95}}{65} + \frac{5d^{2.8}}{13}$, which indicates that L2 will finish in $2^{29.6}$ seconds which equals to 25.8 years. We note that this estimation is quite natural, since the previous research [GN08, NS06] has shown that LLL runs in cubic with regard to d when d is big and $\beta \sim O(d^2)$.

Then we look at the time difference between iLLL and LLL as shown by the third curve in Figure 6.5. The result implies that the time we are able to gain with re-use is linear in dimension, when the curve becomes stable after dimension 130. This indicates that the accumulated time contains a quadratic term. Now we predict the running time of iLLL. We use the heuristic method, since in general it is faster. We predict that the curve follows $\frac{d^{2.95}}{65} + \frac{5d^{2.8}}{13} - \frac{77d^2}{5}$. This gives us $2^{29.47}$ seconds, which equals to 23.6 years. To conclude, we summarize our results in Table 6.1.

Remark 6.1 *The previous best prediction [CN11] was using L^2. We note that it is not specified in [CN11] what kind of a platform (CPU, library, etc.) their test was conducted on. Therefore, we performed the same LLL reduction as in [CN11]. We believe the difference between [CN11] and our approach is due to the implementation. This is the reason why we have compared our own LLL/iLLL implementation against the Gentry-Halevi's challenge.*

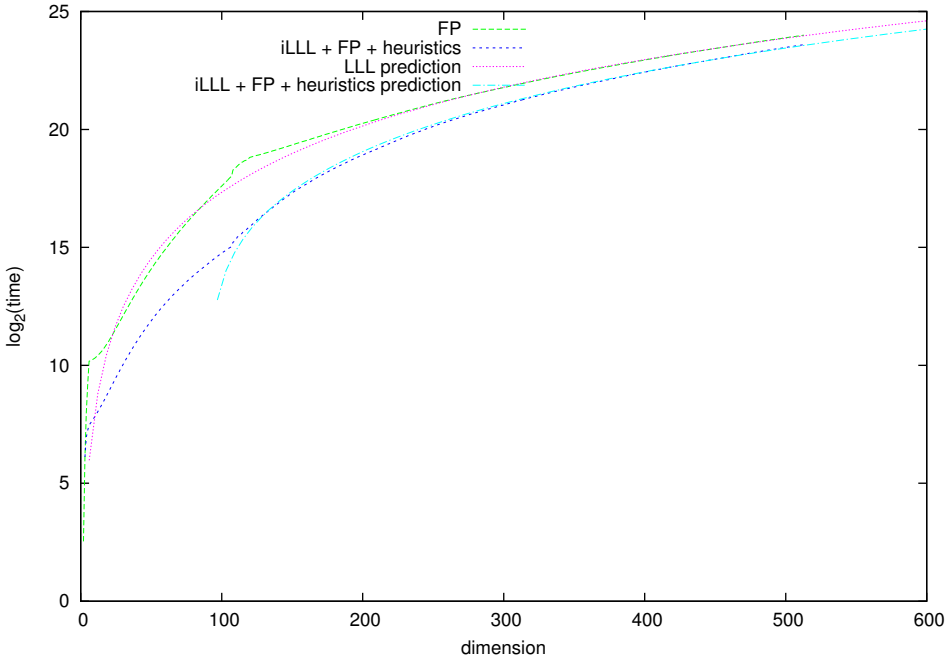

Figure 6.6: Estimated time for dimension 2048

Gentry-Halevi's Challenge	dim 512	dim 2048
The Previous Best Results/Prediction[CN11]	30 days	45 years
LLL implementation @2.66GHz	32 days	25.8 years
iLLL implementation @2.66GHz	24 days	23.6 years
iLLL prediction @4.0GHz	16 days	15.7 years

Table 6.1: Practical Result on Gentry Halevi's Challenge

Chapter 7

CCA-1 Attack against Integer-based SHE schemes

In this chapter we present our CCA-1 attack. We use vDGHV SHE scheme to demonstrate our attack. However, the following attack can be applied to CNMT with a trivial modification. Recall the definition of the IND-CCA security game as follows:

1. The challenger runs KeyGen algorithm and outputs a secret key **sk** and a public key **pk**;
2. The attacker is given two oracles, an encryption oracle and a decryption oracle;
3. The attacker then generates two ciphertexts m_0 and m_1;
4. The challenger generates $c = \text{Encrypt}(m_b, \mathbf{sk})$, where b is uniformly randomly chosen from $\{, 1\}$;
5. (Only for CCA-2) The attacker is given the two oracles again, but it can not query on c;
6. The attacker outputs b'.

The security strength of vDGHV SHE comes from the noise that is added to ciphertexts. If we can somehow reduce the noise in ciphertext, then the scheme will no longer be secure. With the help of a decryption oracle, we can eliminate the noise. Hence, we achieve a CCA-1 attack.

We propose two variants that follow the same idea. The first variant requires several ciphertexts. The main idea is to eliminate the noise and then, to find the GCD of the remaining parts. The second variant requires only *one* ciphertext, and we are able to recover the secret key directly.

We note that our attack recovers the secret key that allows us to decrypt any valid ciphertexts, and does *not* require any access to $\mathcal{O}_D$ at Stage 5 in the CCA attack model.

Thus, our attack falls in the category of CCA-1. However, we also note that essentially, our attack is stronger than CCA-1, because instead of solving only one challenge, we recover the secret key.

7.1 Motivation

As mentioned in the introduction part, the major application of fully homomorphic encryption schemes is to provide a secure outsourced computing for cloud service. Nevertheless, the adoption of data outsourced computation by business has a major obstacle, since the data owner does not want to allow the untrusted cloud provider to have access to the data being outsourced. We highlight some important factors. In fact, one can categorize an outsourced computation into the following models:

1. The user possesses the data and the computation circuit, and the service provider provides the computation power;

   Example 7.1 *a stock share holder buys/sells his/her stocks via the cloud, and then retrieve the receipt from the cloud to obtain his/her updated financial status.*

2. The user possesses the data, and the service provider provides its computational power, while the computation circuit can be made available publicly to both of the entities;

   Example 7.2 *a hospital outsources its patients' information to a research institute for acquiring further analysis from the institute (such as the result of the prostate cancer), as the institute has more computational power compared to the hospital.*

3. The user possesses the data, and the service provider provides its computational power, only the cloud has access to the computation circuit;

   Example 7.3 *a company outsources its financial status to an auditing company, however, the auditing algorithm is auditing the company's private property.*

In all of the above models, the users' data privacy has to be ensured. The difference among them lies on the privacy of the computational circuit.

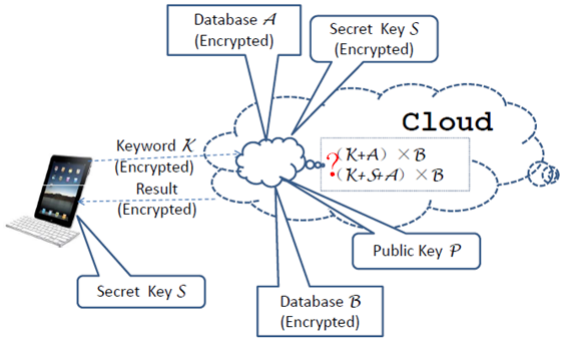

Figure 7.1: Using Fully Homomorphic Encryption in a Cloud Search Scenario

Indeed, a fully homomorphic encryption is a solution for enabling operations on the encrypted data. This feature is useful in the outsourced computation scenario, where one can upload encrypted data to the cloud and enable the cloud to process the data *without* the need for decryption.

We present a practical reaction attack that can be applied to all of the above models, in which every time a user interacts with the cloud, he/she is under the risk of leaking some information. Using this attack, one can construct a probabilistic decryption oracle. Consequently, we argue that for any fully homomorphic encryption schemes, the CCA-1 security is essential.

7.1.1 The Reaction Attack

For any given ciphertext, our attack recovers the message with provability ε.

To use fully homomorphic encryption schemes in outsourced computation scenarios, the users firstly upload their encrypted data to the cloud. Then, they submit their demanded circuits to the cloud, in an on-demand fashion. The demanded circuit consists either of some data and an evaluation function, or merely an evaluation function only. The cloud processes users' data through the requested circuits, and returns the result.

Ideally, all the data, including the results, are encrypted, and hence, a malicious cloud provider cannot gain information from the users, *i.e.*, let ε_1 be the possibility of $m = 1$, then for any ciphertext, $\mid \varepsilon_1 - 1/2 \mid$ is negligible from the cloud provider's point of view.

Nevertheless, we notice that the attacker can modify the encrypted circuits/results

by adding some random ciphertext c that encrypts a message m. Because homomorphism is enabled, modifying the demanded circuits/results will affect the plaintext eventually. To be more precise, if the added random ciphertext encrypts an 0, the returned result remains the same; while if the added random ciphertext encrypts a 1, the returned result is modified. By observing the users' reactions, the service provider can increase or decrease ε_1 accordingly, and eventually recover m.

Generally speaking, the cloud provider can compare users' reaction with their former reaction, if the users are acting "unexpectedly", then the cloud can expect $m = 1$. For completeness, we list some (*but not* all) possible reactions that can be defined as "unexpected" behaviors.

- The users set up a new task much sooner than usual, after they acquire the result sent by the cloud;
- The circuit of a new task is identical from a former one;
- The number of tasks is significantly higher than average - this occurs when the cloud provider feeds same faulty information for a certain period.

We note that these users' reactions are very natural and practical. To anticipate the reactions is even easier, when the users use a certain software, instead of expecting the results themselves, to communicate with the cloud. We argue that this is very common in practice as nobody will conduct this process manually.

However, the success of our attack relies highly on the actions performed by the users after receiving valid or error results. Hence, if the users act completely randomly, then our attack will be unsuccessful. Nevertheless, we argue that the latter usually will not happen in practice, as it is the users' interest to acquire the results that they would like to obtain.

We use several examples to demonstrate our attack.

Example 7.4 (Classic Search) *Suppose we have a cloud search engine (see Figure 7.1), which looks up keywords from database A, and outputs the corresponding results in database B. Database A consists of names of stocks, while database B shows corresponding price for each stock.*

Table 7.1 shows the databases in plaintext for our example.

Let K, A and B denote the binary form of the keyword, database A and database B, respectively. Let k_i, a_i and b_i be the i-th digit of K, A and B. Also, let $\bigotimes_1^n c_i$ be $c_1 \otimes c_2 \otimes \cdots \otimes c_n$. Then, a basic search algorithm is defined in Algorithm 7.

Entry	A	B
1	AAPL	335
2	GOOG	494
3	MSFT	027
4	SPRD	013
5	NDAQ	024
...	...	...

Table 7.1: Databases in plaintext

Then we show how to incorporate fully homomorphic encryption in this example. Without losing generality, we use vDGHV scheme to demonstrate our example.

Example 7.5 (Homomorphic Search) *To distinguish from a traditional search engine, the databases of the cloud search engine are all encrypted, and the search circuit is a homomorphic circuit. Note that Table 7.1 is no longer exactly the databases stored in the cloud. The cloud maintains multiple copies of the databases for different users, each copy is encrypted under different users' FHE secret key. Then, the database in the cloud consists of* $\text{ENC}(a_i)$ *and* $\text{ENC}(b_i)$*. Following the previous example, we achieve a fully homomorphic searching algorithm in Algorithm 8 using the fully homomorphic encryption scheme over integers.*

Algorithm 7 Basic Search (K, A, B)

1: $l_a \leftarrow$ *LEN_OF_WORD_A* $\quad \{l_a = 32\}$
2: $l_b \leftarrow$ *LEN_OF_WORD_B* $\quad \{l_b = 24\}$
3: ***for*** $j = 0 \rightarrow$*END_OF_ENTRY*-1 ***do***
4: ***for*** $i = 1 \rightarrow l_b$ ***do***
5: $r_i \leftarrow b_{i+jl_b} \otimes (\bigotimes_{t=1}^{l_a}(a_{t+jl_a} \oplus k_{t+jl_a})) \oplus r_i$
6: ***end for***
7: ***end for***

As shown in Table 7.2, suppose we want to look for the price of GOOG, with the basic search algorithm, we obtain $52, 57, 52$ *in ASCII code, which is* 494*. While with the homomorphic search algorithm, we obtain the third column of Table 7.2. The cloud cannot decrypt* $\text{ENC}(52)/\text{ENC}(57)/\text{ENC}(52)$*, hence, the users' privacy is guaranteed. The user holds the secret key, therefore, he/she is the only one who knows the searching results, while the cloud cannot even distinguish the difference between the two* $\text{ENC}(52)$*.*

Now we show how our reaction attack works in this example.

Algorithm 8 Homomorphically Search $(\text{Enc}(k_i), \text{Enc}(a_i), \text{Enc}(b_i))$

1: $l_a \leftarrow$ *LEN_OF_WORD_A* $\{l_a = 24\}$
2: $l_b \leftarrow$ *LEN_OF_WORD_B* $\{l_b = 32\}$
3: ***for*** $j = 0 \rightarrow$*END_OF_ENTRY*-1 ***do***
4: ***for*** $i = 1 \rightarrow l_a$ ***do***
5: $r_i \leftarrow \text{Enc}(b_{i+jl_b}) \times \prod_{t=1}^{l_a}(\text{Enc}(a_{t+jl_a}) + \text{Enc}(k_{t+jl_a})) + r_i$
6: ***end for***
7: ***end for***

Database A	Basic Search	Homomorphic Search	Homo Search + Faulty Info
AAPL	0,0,0	Enc(0),Enc(0),Enc(0)	Enc(0),Enc(0),Enc(0)
GOOG	52,57,52	Enc(52),Enc(57),Enc(52)	Enc(0),Enc(0),Enc(0)
MSFT	52,57,52	Enc(52),Enc(57),Enc(52)	Enc(0),Enc(0),Enc(0)
SPRD	52,57,52	Enc(52),Enc(57),Enc(52)	Enc(0),Enc(0),Enc(0)
NDAQ	52,57,52	Enc(52),Enc(57),Enc(52)	Enc(0),Enc(0),Enc(0)
...	...	...	...
result	494	Enc("494")	error

Table 7.2: Searching Results in ASC II

Example 7.6 (Reaction Attack) *Let c be a ciphertext that the cloud would like to decrypt. The cloud adds c to the keyword, and there are two possible consequences.*

- *if* $\text{Dec}(c) = 0$*, the algorithm will search for* $\text{Enc}(GOOG)$ *as before, and therefore, no error will occur, and the user will most likely do nothing.*
- *if* $\text{Dec}(c) = 1$*, instead of searching for* $\text{Enc}(GOOG)$*, the input of the algorithm is actually* $\text{Enc}(GOOH)$*. Therefore, no match will be found. It is reasonable to believe that the user will start to execute another search, in which case the cloud increases* ε_1*.*

As we have stated, a malicious cloud can also modify the circuit/result accordingly. However, we notice that in the above example, modifying the result merely helps our attack, as if the cloud induces an $\text{Enc}(1)$*, the user will receive* 495*, which will be recognized as a valid result.*

7.1.2 Impact on CCA Security

It is well known that for any public key encryption schemes, the CPA security is essential. Meanwhile, constructing a CCA-2 secure fully homomorphic encryption scheme

is impossible, since homomorphic operations on ciphertexts are enabled (and also it is due to the "malleability" of the ciphertext). Moreover, unfortunately, fully homomorphic encryption schemes that follow Gentry's framework cannot be CCA-1 secure due to the bootstrapping technique. In fact, if a somewhat homomorphic encryption scheme is CCA-1 secure is questionable. Indeed, we have seen two CCA-1 attacks in the previous chapter.

We note that the consequence of such an attack might be severe. A CCA-1 attack is an attack model that assumes there exists a decryption oracle. People may argue that this is just merely an attack model, since in practice, having such an oracle (or an honest user who is helping the attacker during the learning phase) is impractical. Furthermore, constructing such an oracle in general is not really feasible. Nevertheless, with our message attack, it is possible to construct a probabilistic decryption oracle in practice. Consequently, if for a certain fully homomorphic encryption scheme, a CCA-1 attack is successful with a non-negligible advantage of ϕ using a hypothetical deterministic oracle, then one can achieve this attack with an advantage of $\phi\varepsilon^n$ using our message attack, where n is the number of ciphertext required for the CCA-1 attack. Hence, the CCA-1 attack becomes really practical.

To sum up, we argue that in practice, if a fully homomorphic encryption scheme is not CCA-1 secure, then it alone cannot deliver secured outsourced computation.

7.2 The CCA-1 Attack

Essentially, in vDGHV SHE scheme, the public keys (x_i-s) can be treated as ciphertexts encrypting 0-s with smaller noise. Therefore, the following algorithms can be applied on public keys with less cost. However, in order to strictly follow the definition of CCA-1 attack, we apply our algorithms on real ciphertexts.

We note that for any correct ciphertext c_i, the following holds that

$$c_i = m_i + 2r'_i + g'_i p$$

for certain $r'_i \in (-p/4, p/4]$ and integer g'_i, since if $|2r'| > p/2$ decryption error will be induced. For convenience, denote $\alpha' = \beta - 1$. Using the recommended parameter configuration, we have $\alpha' = \lambda^2 - 1$, where λ is the security parameter.

Now we show our attack against vDGHV SHE scheme. Suppose we have k ciphertexts $c_1, c_2, ..., c_k$ of encrypted 0-s, i.e.: $c_i = g'_i p + 2r'_i$. It holds that $\text{DECRYPT}(c_i, sk) = 0$. The length of r'_i-s is no greater than α'.

Let $\mathcal{O}_D(c)$ be the decryption oracle that returns $\text{DECRYPT}(c, sk)$. The following pieces of pseudo-code describe two variants of our attack.

Algorithm 9 NOISEELI(c)

$l_p \leftarrow 2^{\beta} - 2^{\alpha'+1}$
$r_p \leftarrow 2^{\beta+1} + 2^{\alpha'+1}$
while $r_p - l_p > 2$ **do**
  $s \leftarrow \lfloor (l_p + r_p)/4 \rceil \times 2$
  if $\mathcal{O}_D(c + s) = 0$ **then**
    $l_p \leftarrow s$
  end if
  if $\mathcal{O}_D(c + s) = 1$ **then**
    $r_p \leftarrow s$
  end if
end while
if $\mathcal{O}_D(c + s + 1) = 1$ **then**
  $s \leftarrow s + 1$
end if
return $c' \leftarrow c + s$

Algorithm 9: NOISEELI is to help to eliminate the noise in ciphertext. Instead of generating a ciphertext with no noise, this algorithm generates a ciphertext with a fixed noise.

Algorithm 10: CCA-GCD describes the first variant of our attack, while Algorithm 11 CCA-P describes the second variant of our attack.

Algorithm 10 CCA-GCD($c_0, c_1, ..., c_k$)

$c'_0 \leftarrow \text{NOISEELI}(c_0)$
$c'_1 \leftarrow \text{NOISEELI}(c_1)$
$c'_2 \leftarrow \text{NOISEELI}(c_2)$
$p' \leftarrow \gcd(c'_2 - c'_1, c'_1 - c'_0)$
$t \leftarrow 3$
while $p' \geq 2^{\beta+1}$ **and** $t \leq k$ **do**
  $c'_t \leftarrow \text{NOISEELI}(c_t)$
  $p' \leftarrow \gcd(c'_t - c'_{t-1}, p')$
  $t \leftarrow t + 1$
end while
return p'

The first algorithm inputs a ciphertext $c = 2r' + g'p$ with any noise r', it outputs a new ciphertexts $c' = g'p + \lfloor p/2 \rfloor$. The new ciphertext contains a constant noise of $\lfloor p/2 \rfloor$.

For any ciphertext $c = 2r + gp$, the DECRYPT algorithm will always output 0, as long as $|2r| \leq \lfloor p/2 \rfloor$, and output 1 if $2r > \lfloor p/2 \rfloor$. Denote $l = \lfloor p/2 \rfloor - 2r$. l represents the threshold, such that $\mathcal{O}_D(c + l) = 0$ and $\mathcal{O}_D(c + l + 2) = 1$. Also, we know that $l \in (2^\beta - 2^{\alpha'+1}, 2^{\beta+1} + 2^{\alpha'+1})$. Therefore, we set l_p and r_p to be the lower and upper bound of l. Then we start a while loop to narrow the bound as in Algorithm 9.

Algorithm 11 CCA-P(c)

1: $a \leftarrow$ NOISEELI(c)
2: $b \leftarrow$ NOISEELI($-c$)
3: **return** $a + b + 1$

7.3 Analysis

7.3.1 Correctness

In this section we prove the correctness of our attack.

For any *even* integer $s \in (2^\beta - 2^{\alpha'+1}, 2^{\beta+1} + 2^{\alpha'+1})$, decrypting $c + s$ has two possible consequences:

- If $\mathcal{O}_D(c + s) = 1$, we know that the threshold l for this ciphertext is smaller than s, then we move the upper bound r_p to s;
- One the other hand, if $\mathcal{O}_D(c + s) = 0$, we know that the threshold l for this ciphertext is greater than s, then we move the lower bound l_p to s;

By the end of the loop, we have $s = l_p = r_p - 2$. Also, it holds that $\mathcal{O}_D(c + s) = 0$ and $\mathcal{O}_D(c + s + 2) = 1$. If $\lfloor p/2 \rfloor$ is an even integer, then $s = \lfloor p/2 \rfloor$, and $\mathcal{O}_D(c + s + 1) = 1$. By contrast, if $\lfloor p/2 \rfloor$ is odd, then $s = \lfloor p/2 \rfloor - 1$, and $\mathcal{O}_D(c + s + 1) = 0$. In this case, we increase s by 1.

Hence, s is the threshold l we were looking for, and $c + s = gp + \lfloor p/2 \rfloor$. Therefore, we successfully generate a fixed noise ciphertext in $\log(2^\beta + 2^{\alpha'+2}) + 1$ queries.

The second algorithm is more straightforward. Given $k + 1$ outputs of Algorithm 1, we obtain k linear independent noise free ciphertexts. By running a classic GCD algorithm we obtain p'. It holds that either $p = p'$ or $p|p'$.

To have $p = p'$ it requires $\gcd(g_2 - g_1, g_3 - g_2, ..., g_k - g_{k-1}) = 1$. The probability of k random integers from $\mathbb{Z}$ to be coprime is $1/\zeta(k)$, where $\zeta(x)$ is the Riemann Zeta

function $\sum_{i=1}^{\infty} \frac{1}{i^x}$ (see [Nym75] for more details), while the probability of having k numbers randomly chosen form $(0, 2^{\lambda^5})$[1] to be coprime is greater than $1/\zeta(k)$.

In practice, 4 random integers have $1/\zeta(4) > 92\%$ probability of being coprime, while 7 random integers have $1/\zeta(7) > 99\%$ probability of being coprime. Our test (see section 3.5) confirmed this result, where on average cases, 3 random integers are coprime.

For the last algorithm, we generate $a = \text{NoiseEli}(c)$ and $b = \text{NoiseEli}(-c)$. It holds that:

$$a = gp + \lfloor p/2 \rfloor, \qquad b = -gp + \lfloor p/2 \rfloor.$$

Therefore, we obtain $2 \times \lfloor p/2 \rfloor$ from $a + b$. Because p is an odd integer, we recover the secret key by $p = a + b + 1$.

7.3.2 Complexity

We examine the efficiency of our last two algorithms. For original ciphertexts (no homomorphic operations have been evaluated on them), it requires $\log(2^{\beta} + 2^{\alpha'+2}) + 1 < \beta + 3$ queries to find the fixed noise ciphertext. CCA-GCD algorithm requires a minimum 3 fixed noise ciphertexts. Therefore, in best cases we recover the secret key in $3(\beta + 3)$ queries. As $\beta = \lambda^2$, Algorithm 10 recovers the secret key in $O(\lambda^2)$ operations.

Algorithm 11 also works on $O(\lambda^2)$ but with better performance. To be more precise, it uses one ciphertext only, therefore to eliminate the noise requires at most $2(\beta + 3)$ queries.

It is true that Algorithm 11 is more efficient that Algorithm 10. The reason that we propose Algorithm 10 is that we observe Algorithm 11 will fail if we modify the decryption circuit. For instance, if $c \bmod p$ returns an integer within $[0, p)$ instead of $(-p/2, p/2]$, then Algorithm 11 will be unsuccessful. However, in this case Algorithm 10 is still valid.

7.3.3 An Example

In this section, we give an example of our CCA-1 attack. In our example, the security parameter λ is 2. Therefore, the noise r'_i and the multiplier g'_i are bounded by 2^2 and

[1]The integer-based FHE requires each elements to be λ^5 bits in order to stop a lattice attack[vDGHV10]

	r'_i	p	g'_i	c'_i
c_1	-3	19	343759059	6531422115
c_2	1	19	230194545	4373696357
c_3	2	19	276209466	5247979858

Table 7.3: Three sample ciphertexts.

2^{28}, respectively. The secret key p is an odd integer between 2^4 and 2^5. The ciphertext is in form of $c'_i = 2r'_i + g'_i p$. Table 7.3 lists three ciphtertexts.

The results below indicate that we retrieve the secret key successfully. Table 7.4 shows that the noise is successfully eliminated within maximum 5 queries for each ciphertext. We recover s_i for each ciphertext such that $c_i + s_i = g_i p + (p-1)/2$.

Table 7.5 and 7.6 show how to extract p from c'_i in two ways. Both of the two examples uses NoiseEli to find constant noise ciphertext. As displayed in the tables, in GCD-CCA we recover $3p$ instead of p, and we did not further proceed the algorithm, since it is merely an example. This example requires access to the oracle for 14 times for the GCD-CCA variant and 10 times for CCA-p.

NoiseEli(c_1)				
l_p	r_p	s_1	$\mathcal{O}_D(c_1 + s_1)$	action
0	24	12	0	$l_p \leftarrow 12$
12	24	18	1	$r_p \leftarrow 18$
12	18	14	0	$l_p \leftarrow 14$
14	18	16	1	$r_p \leftarrow 16$
14	16	14		end of loop
$\mathcal{O}_D(c_1 + s_1 + 1) = 1 \Longrightarrow s_1 = 15$				

NoiseEli(c_2)				
l_p	r_p	s_2	$\mathcal{O}_D(c_2 + s_2)$	action
0	24	12	1	$r_p \leftarrow 12$
0	12	6	0	$l_p \leftarrow 6$
6	12	8	1	$r_p \leftarrow 8$
6	8	6		end of loop
$\mathcal{O}_D(c_2 + s_2 + 1) = 1 \Longrightarrow s_2 = 7$				

NoiseEli(c_3)				
l_p	r_p	s_3	$\mathcal{O}_D(c_3 + s_3)$	action
0	24	12	1	$r_p \leftarrow 12$
0	12	6	1	$r_p \leftarrow 6$
0	6	2	0	$l_p \leftarrow 2$
2	6	4	0	$l_p \leftarrow 4$
4	6	4		end of loop
$\mathcal{O}_D(c_3 + s_3 + 1) = 1 \Longrightarrow s_3 = 5$				

Table 7.4: Eliminate the noise of three ciphertexts.

CCA-GCD(c_1,c_2)	
$c_1' = c_1 + 15 - c_2 - 7$	$c_2' = c_2 + 7 - c_3 - 5$
$p' = \gcd(c_1', c_2') = 57$	

Table 7.5: Find p with CCA-GCD.

7.4 Extensions

7.4.1 Comparisons

We note that the LMSV attack is different from the attack described in this chapter. The LMSV attack uses the decryption oracle to find the integer s such that $[w \times s/d] = 1/2$, and eventually recover the secret key, while our attack aims to manipulate the noise in the ciphertexts. By recovering the noise, our attack will recover the secret key of vDGHV variant.

A proof of such a difference is that our attack can be adapted to recover the noise for LMSV SHE scheme as well. However, this does not help us to recover the secret key or break the CCA-1 security. Another evidence is that using LMSV SHE's solution (i.e. generating $\perp$ for invalid ciphertext) *will not* stop our attack either (see section 7.4.2).

We also note that our method cannot be adapted to attack SHE schemes that use ideal lattices. Recall the SHE scheme in the Gentry scheme. Essentially, one needs

$$\vec{c} \leftarrow \vec{m} + \vec{r} \times \mathcal{I} + \vec{g} \times \mathcal{J}$$

where $\vec{r}$ and $\vec{g}$ are randomly chosen. $\mathcal{I}$ and $\mathcal{J}$ are two ideal lattices that are co-prime. More specifically, the encryption algorithm is

$$\vec{c} = \vec{m} + \vec{r} \cdot \mathbf{B}_{\mathcal{I}} + \vec{g} \cdot \mathbf{B}_{\mathcal{J}}^{\mathbf{pk}}$$

Therefore, even we can somehow eliminate $\vec{r}$ through our attack, we still need to solve such a problem: given as many $\vec{g} \cdot \mathbf{B}_{\mathcal{J}}^{\mathbf{pk}}$, find $\mathbf{B}_{\mathcal{J}}^{\mathbf{sk}}$. This is a GGH type cryptosystem [GGH97]. As a result, we cannot recover $\mathbf{B}_{\mathcal{J}}^{\mathbf{pk}}$ directly using our technique.

Example 7.7 *Take the* LMSV *scheme for instance. for a ciphertext* $c = 2r(a) + m \pmod{d}$*, our attack recovers* $r(x)$*. To recover the* i*-th coefficient of* $r(x)$*, one passes* $c + 2 \times r' \times a^i$ *to the decryption oracle, where* r' *is a random integer picked by the attacker. By observing if the oracle returns* m *or* $\perp$*, one increases or decrease* r' *accordingly. Eventually, the attacker obtains* $r' + r_i = T$*. As a result, the attacker recovers one coefficient of* $r(x)$*. By doing this repetitively, one recovers the entire* $r(x)$*.*

However, this attack is not successful. To recover the secret key one still needs to solve the following problem: given an ideal lattice in the form of a, d*, find a good basis of this lattice.*

7.4.2 Possible Solutions

In this section we consider the existing proposed solution to make vDGHV SHE scheme CCA-1 secure. We note that our attack is successful, since we are able to eliminate the noise. Therefore, if there exist some techniques to disturb the noise elimination, our attack will fail.

Loftus et al. [LMSV11] showed a solution to combat their own CCA-1 attack against Gentry-Halevi SHE scheme/Smart-Vercauteren SHE scheme (which we refer to as the LMSV SHE scheme). However, the solution in LMSV SHE scheme is *not applicable* in our case. Their possible solution is to generate some error $\perp$ (or even some random 0-s or 1-s), when the decryption oracle detects that the noise r is very close to $\pm(p-1)/2$. The decryption algorithm sets a bound T, such that when $(p-1)/2 - |r| < T$, it will not proceed decryption. However, essentially it will still leak some information. We can modify our attack to find T and consequently find a fixed noise ciphertext.

We modify our attack as follows: for each round, we query to the oracle multiple times. If the feedbacks are consistent (meaning that the attacker is not confused by random 0-s and 1-s) and not $\perp$, we proceed to the next round. Otherwise, we recover a fixed noise ciphertext with a noise level of T. Hence, our attack will still be successful even after the "patch" suggested by Loftus et al. [LMSV11].

7.5 Implementation

In this section, we show the result of our implementation of our attack with different λ. This implementation is based on the NTL library [Sho].

The implementation was conducted in a 2.66 GHz CPU. The memory was always sufficient, as it merely required more than 600 Mbs. We started from $\lambda = 2$, and increased λ continuously until it reached 32. For each λ, we fed the program with 100 different seeds, and recorded the average time to find the secret key p, as well as average number of ciphertexts required for Algorithm 2.

The average number of ciphertexts required for Algorithm 2 for different choice of λ is quite stable. Approximately 3.8 ciphertexts are required to recover the secret key p. This implies that on average case the number of integers required to have them being coprime is 2.8.

Figure 7.2 shows the timing results of our implementation. The x axis shows the choice of λ, while the y axis indicates the average time (in seconds) for each attack.

NoiseEli(c_1)				
l_p	r_p	s_1	$\mathcal{O}_D(c_1 + s_1)$	action
0	24	12	0	$l_p \leftarrow 12$
12	24	18	1	$r_p \leftarrow 18$
12	18	14	0	$l_p \leftarrow 14$
14	18	16	1	$r_p \leftarrow 16$
14	16	14		end of loop
$\mathcal{O}_D(c_1 + s_1 + 1) = 1 \Longrightarrow s_1 = 15$				

NoiseEli($-c_1$)				
l_p	r_p	s_{-1}	$\mathcal{O}_D(c_1 + s_{-1})$	action
0	24	12	1	$r_p \leftarrow 12$
0	12	6	1	$r_p \leftarrow 6$
0	6	2	0	$l_p \leftarrow 2$
2	6	4	1	$r_p \leftarrow 4$
2	4	2		end of loop
$\mathcal{O}_D(-c_1 + s_{-1} + 1) = 1 \Longrightarrow s_{-1} = 3$				

CCA-p(c_1)	
$a \leftarrow c_1 + 15$	$b \leftarrow -c_1 + 3$
$p \leftarrow (a + b) + 1 = 19$	

Table 7.6: Find p with CCA-p.

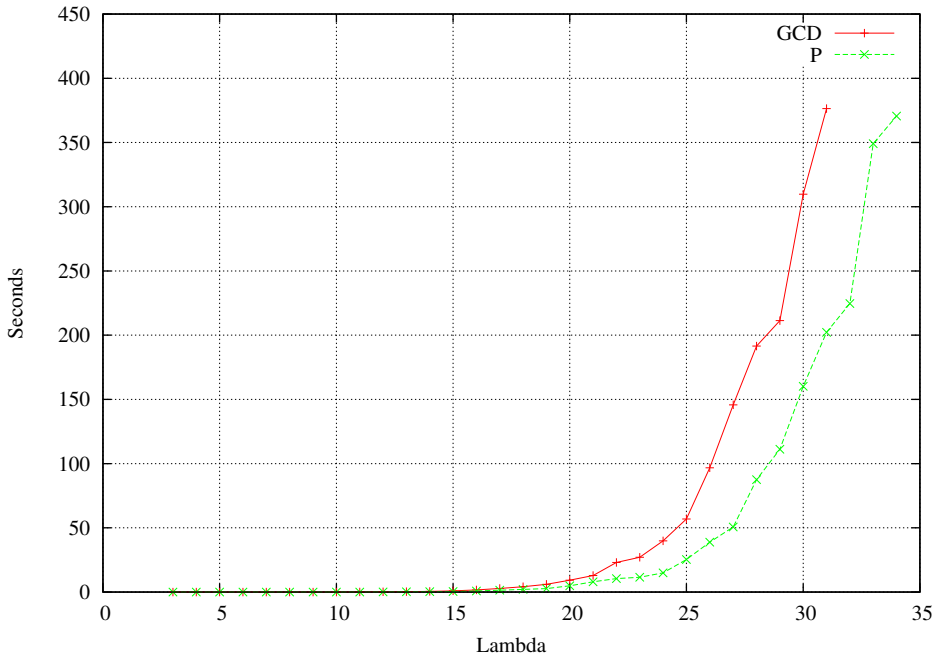

Figure 7.2: Average time for recovering p.

Statistically, attack CCA-P uses approximately 1.9 times more time in comparison to attack CCA-GCD (this is due to the number of ciphertexts required to be noise-eliminated), and this is consistent with our result.

Part III

A New Fully Homomorphic Encryption Scheme

Chapter 8

The Hidden Lattice

In this chapter, we introduce the notion of hidden lattice. Based on a hidden lattice, we present new problem, namely *the bounded distance decoding over hidden ideal lattice* We show that the bounded distance decoding problem over hidden ideal lattice (BDDH problem) is harder than both the BDD over ideal lattice and AGCD problems, which, as mentioned earlier, are the two out of three main problems that have been used to design fully homomorphic encryption schemes do date.

More specifically, we show that the BDDH problem over dimension n is equivalent to the BDD problem over dimension $O(n^{\xi})$, where $\xi \in [1, 2]$. Furthermore, we conjectured that the proposed scheme is still secure even when $\xi = 2$.

8.1 The Hidden Lattice Theory

We formally define the new problems related to the hidden lattice.

Definition 8.1 (Hidden (Ideal) Lattice) *Let $\alpha \in \mathbb{R}^+$ be a positive real, $\vec{v}_i \in \mathbb{Z}^n$ be τ integer vectors such that there exists a unique (ideal) lattice $\mathcal{L}$ and some unique vectors $\vec{w}_i \in \mathcal{L}$ respecting $\forall 1 \leq i \leq \tau$, $dist(\vec{v}_i, \vec{w}_i) \leq \alpha$. Then $\mathcal{L}$ is called an α-Hidden (Ideal) Lattice hidden under $\{\vec{v}_i\}$.*

For a hidden lattice, it is quite straightforward to see that one of the core problems is to recover the lattice, given that there exist such a lattice, which we shall define as follows:

Definition 8.2 (Hidden (Ideal) Lattice Problem) *Let $\alpha \in \mathbb{R}^+$ be a positive real, $\vec{v}_i \in \mathbb{Z}^n$ be τ integer vectors such that there exists an α-Hidden (Ideal) Lattice $\mathcal{L}$ hidden under $\{\vec{v}_i\}$. The α-Hidden (Ideal) Lattice Problem, denoted by α-$HLP_{n,\tau}$ (α-$HILP_{n,\tau}$, resp.), is to find $\mathcal{L}$, given $\{\vec{v}_i\}$.*

Informally, in the HLP/HILP the existence and the uniqueness of the lattice is guaranteed by several vectors close to the lattice, and one is asked to find the such a lattice given those vectors.

Definition 8.3 (BDDP over Hidden (Ideal) Lattice) *Let $\alpha, \beta \in \mathbb{R}^+$ be some positive reals. Let $\vec{v}_i \in \mathbb{Z}^n$ be τ integer vectors such that there exists an α-Hidden (Ideal) Lattice, $\mathcal{L}$, hidden under $\{\vec{v}_i\}$. Let $\vec{u} \in \mathbb{Z}^n$ be an integer vector such that there exists a unique $\vec{w} \in \mathcal{L}$ respecting $dist(\vec{u}, \vec{w}) \leq \beta$. Then the Bounded Distance Decoding problem over Hidden(ideal) lattice, denoted by α, β-$BDDH_{n,\tau}$ (α, β-$BDDHi_{n,\tau}$, resp.), is to find $\vec{w}$, given $\{\vec{v}_i\}$ and $\vec{u}$.*

Definition 8.4 (Dec BDDP over Hidden (Ideal) Lattice) *Let $\alpha, \beta \in \mathbb{R}^+$ be some positive reals. Let $\vec{v}_i \in \mathbb{Z}^n$ be τ integer vectors such that there exists an α-Hidden (Ideal) Lattice, $\mathcal{L}$, hidden under $\{\vec{v}_i\}$. Let $\vec{u} \in \mathbb{Z}^n$ be an integer vector. Then the Decisional Bounded Distance Decoding problem over Hidden (ideal) lattice, denoted by Dec α, β-$BDDH_{n,\tau}$ (Dec α, β-$BDDHi_{n,\tau}$, resp.), is to decide if there exists a unique $\vec{w} \in \mathcal{L}$ such that $dist(\vec{u}, \vec{w}) \leq \beta$ or not, given $\{\vec{v}_i\}$ and $\vec{u}$.*

8.2 Reductions from Existing Problems

In this section, we provide reductions of our new problems from some existing problems. The relations among the problems is described in Figure 8.1, where an arrow from problem A to problem B means that A is no easy to solve than B. We omit the proof of the reductions from problems over general lattice to problems over ideal lattice (i.e. HILP to HLP), since if an algorithm can solve the problem in any lattice, it can solve the problem in an ideal lattice.

Here, we do not provide an average case/worst case equivalence. We note that this is not surprising, considering that both of the previous works that we generalized (due to Gentry and Halevi's scheme [GH11] and van Dijk et.al's scheme [vDGHV10]) also do not provide such a proof. Nevertheless, Gentry also provided an average/worst case equivalent for BDD over ideal lattice [Gen10b], but this work was not adopted in the subsequent work in Gentry and Halevi's scheme due to its impracticality. Further, both of the schemes [GH11, vDGHV10] also rely on another problem (SSSP), in which the proof for the average case/worst case equivalent has never been investigated yet, to the best of our knowledge.

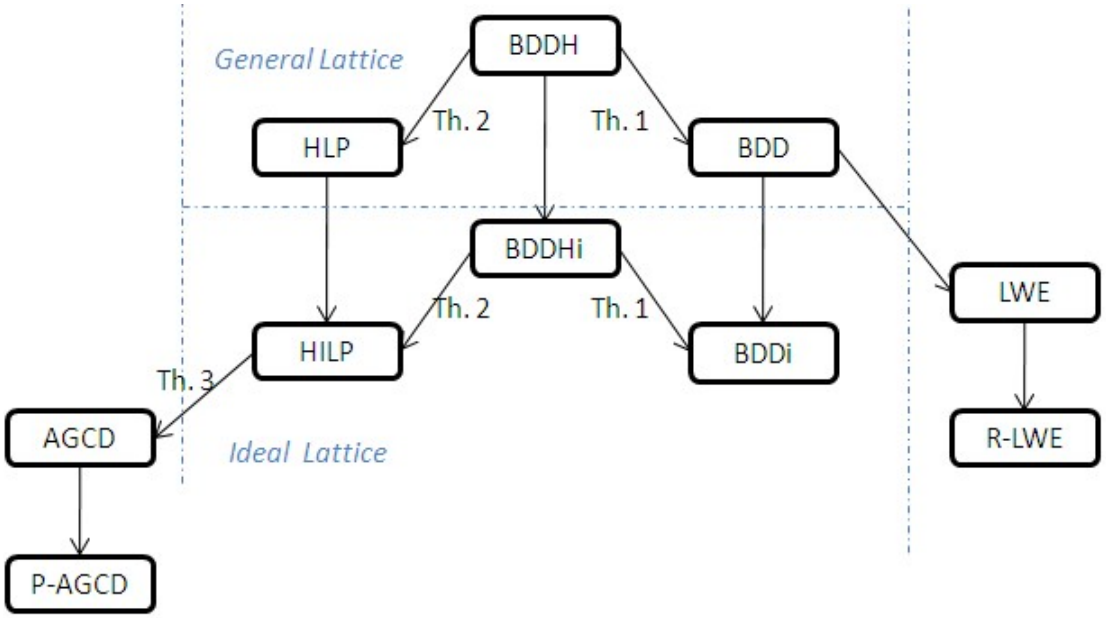

Figure 8.1: Relations among problems.

Theorem 8.1 *If an algorithm $\mathcal{A}$ solves α, β-BDDH$_{n,\tau}$ (α, β-BDDHi$_{n,\tau}$ resp.) with an advantage of ε, then there exists an algorithm $\mathcal{B}$ that solves the γ-BDD$_n$ (γ-BDDi$_n$ resp.) with an advantage of at least ε. The running time of $\mathcal{B}$ is polynomial in the running time of $\mathcal{A}$.*

Proof: Let $\{\vec{v}_i\}, \vec{u}$ be the input of a γ-BDD$_n$ (γ-BDDi$_n$ resp.) problem, where $\{\vec{v}_i\}$ is a basis of $\mathcal{L}$. Set $\tau \leftarrow \#\vec{v}_i$, $\alpha \leftarrow 0$ and $\beta \leftarrow \gamma$. Call $\mathcal{A}$ with $\{\vec{v}_i\}$, $\vec{u}$. Since $\text{dist}(\vec{v}_i, \mathcal{L}) \leq \alpha$, and $\text{dist}(\vec{u}, \mathcal{L}) \leq \gamma$, $\{\vec{v}_1\}$ and $\vec{u}$ is in the correct form of the input of $\mathcal{A}$. Therefore, $\mathcal{A}$ returns the unique $\vec{w} \in \mathcal{L}$ such that $dist(\vec{u}, \vec{w}) \leq \beta = \gamma$. Return $\vec{w}$. The above theorem is also correct for the decisional version of the problems. We omit the proof, which can be adapted in a similar fashion as above.

Theorem 8.2 *If an algorithm $\mathcal{A}$ solves α, β-BDDH$_{n,\tau}$ (α, β-BDDHi$_{n,\tau}$, resp.) with an advantage of ε, then there exists an algorithm $\mathcal{B}$ that solves the α-HLP$_{n,\tau}$ (α-HILP$_{n,\tau}$, resp.) with an advantage of at least ε. The running time of $\mathcal{B}$ is polynomial in the running time of $\mathcal{A}$.*

Proof: Let $\{\vec{v}_i\}$ be the input of an α-HLP$_{n,\tau}$ (α-HILP$_{n,\tau}$, resp.). Set $\beta \leftarrow \alpha$, $\tau \leftarrow \#\vec{v}_i$. For $1 \leq i \leq \tau$, set $\vec{u} \leftarrow \vec{v}_i$ and call $\mathcal{A}$ with $\{\vec{v}_i\}$, $\vec{u}$ to get the unique $\vec{w}_i \in \mathcal{L}$ where $\mathcal{L}$ is the α-Hidden (Ideal) Lattice hidden under $\{\vec{v}_i\}$, such that $dist(\vec{u}, \vec{w}_i) \leq \beta = \alpha$. Reconstruct the lattice $\mathcal{L}$ from the generating vectors $\{\vec{w}_i\}$. Return $\mathcal{L}$.

Theorem 8.3 *If an algorithm $\mathcal{A}$ solves α-HILP$_{n,\tau}$ with an advantage of ε, then there exists an algorithm $\mathcal{B}$ that solves the γ-AGCD$_\tau$ with an advantage of at least ε. The running time of $\mathcal{B}$ is polynomial in the running time of $\mathcal{A}$.*

Proof: Let c_i be the input of a γ-AGCD$_\tau$ problem. Set $n \leftarrow 1$, $\alpha \leftarrow \gamma$ and $\vec{v}_i \leftarrow \langle c_i \rangle$. Call $\mathcal{A}$ with $\{\vec{v}_i\}$ to get the unique $\mathcal{L}$ such that $\text{dist}(v_i, \mathcal{L}) \leq \alpha$. The basis of $\mathcal{L}$ is equal to the vector $\langle p \rangle$ such that $c_i = q_i p + r_i$ with $|r_i| \leq \alpha$. Return p.

Chapter 9

A Homomorphic Encryption from Hidden Lattice

Now we describe our homomorphic encryption scheme from hidden lattice. As one shall see, we aim to build our scheme on a harder problem than the existing problems. As we shall see in the next chapter, a harder problem enables us to construct an FHE scheme with smaller parameters, which will leads to better performance.

9.1 The Somewhat Homomorphic Encryption Scheme

The general idea of our work is to give some vectors close to the lattice, instead of giving the lattice directly, for enabling encryption. Only the secret key holder knows the lattice, and hence, can perform the correct decryption. The ciphertexts are vectors close to the lattice with a bounded distance, therefore we do not lose the property of homomorphism of the ciphertext. Indeed, Figure 9.1 illustrates a comparison between classic lattice-based cryptography and hidden lattice-based cryptography. One shall see that less information with regard to the lattice is released, which, in return, makes the scheme more secure.

The construction below uses the following parameters. chapter 10.4 provides the concrete values for the parameters.

- ρ: the norm of the random noise vector;
- η: the bit length of the norm of generating polynomial;
- γ: the bit length of the norm of the random multiplier vector;
- τ: the number of vectors in the public key;
- ζ: the norm of noise used in encryption;
- n: the dimension of the hidden lattice.

Our somewhat homomorphic encryption scheme uses following four algorithms:

KeyGen(λ)

- Set parameters $\rho, \eta, \gamma, \tau, \zeta, n$ as in chapter 10.4 with respect to λ, n is a power of 2;
- Pick an irreducible polynomial of degree n, $f(x) = x^n + 1$ (see Remark 2);
- Pick a vector $\vec{v}$ randomly in $\{\vec{u} \in \mathbb{Z}^n, 2^{\eta-1} < \|\vec{u}\| < 2^{\eta}, \sum_{i=0}^{n-1} u_i \bmod 2 = 1\}$;
- Generate the rotation matrix $\mathbf{V} \leftarrow \text{Rot}(\vec{v}, f)$, i.e., when $f(x) = x^n + 1$,

$$\text{Rot}(\vec{v}, f) = \begin{vmatrix} v_0 & v_1 & v_2 & \dots & v_{n-1} \\ -v_{n-1} & v_0 & v_1 & \dots & v_{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -v_1 & -v_2 & -v_3 & \dots & v_0 \end{vmatrix}.$$

- $d \leftarrow |\det(\mathbf{V})|$ is the determinant of $\mathbf{V}$ (see Remark 3);
- Pick $\tau - 1$ vectors $\vec{g}_i$ randomly in $\{\vec{u} \in \mathbb{Z}^n, 2^{\gamma-1} < \|\vec{u}\| < 2^{\gamma}\}$ and another vector $\vec{g}_\tau$ randomly in $\{\vec{u} \in \mathbb{Z}^n, \|\vec{u}\| < 2^{\gamma}, \sum_{j=0}^{n-1} u_j \bmod 2 = 1\}$;
- Pick $\tau - 1$ vectors $\vec{r}_i$ randomly in $\{\vec{u} \in \{-1, 0, 1\}^n, \|\vec{u}\| \leq \rho\}$ and another vector $\vec{r}_\tau$ randomly in $\{\vec{u} \in \{-1, 0, 1\}^n, \|\vec{u}\| \leq \rho, \sum_{j=0}^{n-1} u_j \bmod 2 = 1\}$;
- Compute τ vectors $\vec{\pi}_i \leftarrow \vec{g}_i \times \vec{v} + \vec{r}_i$ for $1 \leq i \leq \tau$;
- Find the integer polynomial $w(x)$, such that $w(x) \times v(x) = d \bmod f(x)$ (see Remark 3), denote $\mathbf{W} \leftarrow \text{Rot}(\vec{w}, f)$;
- Output $sk \leftarrow \{d, \vec{w}\}$ and $pk \leftarrow \{\vec{\pi}_i\}$.

Remark 9.1 *In [SV10], Smart and Vercauteren showed that one can use any irreducible polynomial for the ideal lattice to build FHE cryptosystems, however, Gentry*

and Halevi [GH11] restricted it to x^n+1, where n is a power of 2, for enabling a faster operation. Recent result in [SS11] show that n being a power of 2 is not essential, even if that leads to x^n+1 being not irreducible. In our case, we adopt the setting from Gentry and Halevi's scheme in [GH11] to have fast operations.

Remark 9.2 *The complexity of finding $w(x)$ depends on d. If d is prime, one can execute XGCD(v, f) to find w, where XGCD is the extended GCD algorithm [SV10]. However, this is only feasible on a small dimension. In a large dimension, having d to be prime is costly. One needs to use Gentry and Halevi's technique, where d needs to be* odd.

ENCRYPT(m, pk)

- Pick $\tau+1$ integer vectors $\{\vec{s}_1, \ldots, \vec{s}_\tau, \vec{s}_{\tau+1}\}$ satisfying:
  - $\sum_{j=1}^{n} s_{i,j} \bmod 2 = 0$, $1 \leq i \leq \tau - 1$
  - $\sum_{j=1}^{n} s_{\tau,j} \bmod 2 = m$, $\sum_{j=1}^{n} s_{\tau+1,j} \bmod 2 = 0$;
  - Denote $\vec{s} \leftarrow \langle \vec{s}_1, \ldots, \vec{s}_\tau, \vec{s}_{\tau+1} \rangle$, $\|\vec{s}\| \leq \zeta$.
- Output $\vec{\psi} \leftarrow \sum_{i=1}^{\tau} \vec{s}_i \times \vec{\pi}_i + \vec{s}_{\tau+1}$

DECRYPT($\vec{\psi}$, sk)

- $\vec{\psi}' \leftarrow \lfloor \vec{\psi} \times \vec{w}/d \rceil$;
- Return $m \leftarrow \psi'(1) \bmod 2$.

EVALUATE $(\psi_1, \ldots, \psi_t, C, pk)$

- For each addition or multiplication gate in C, call ADD or MULT algorithm;
- Return the output of C.

ADD$(\vec{\psi}_1, \vec{\psi}_2)$

- Return $\psi \leftarrow \vec{\psi}_1 + \vec{\psi}_2$.

MULT$(\vec{\psi}_1, \vec{\psi}_2)$

- Return $\vec{\psi} \leftarrow \vec{\psi}_1 \times \vec{\psi}_2$.

9.2 Analysis

Below we review two definitions provided by Gentry's that will be used in our proof.

Definition 9.1 (r_{Enc}) *r_{Enc} represents the maximum possible distance between a ciphertext $\vec{\psi}$ generated by* ENCRYPT *algorithm and the hidden lattice $\mathcal{L}$.*

Definition 9.2 (r_{Dec}) *r_{Dec} represents the decryption radius: the minimum distance such that any $\vec{\psi}$ (generated by* ENCRYPT *or* EVALUATE*) can be decrypted correctly, if dist$(\vec{\psi}, \mathcal{L}) \leq r_{Dec}$.*

These definitions are also applicable to r_{pk}, i.e., the maximum distance between a public key $\vec{\pi}_i$ and the hidden lattice. As per definition, we have $r_{pk} = \rho$. Our noise of a ciphertext comes from the production of $\vec{s}$ and $\vec{r}_i$. Hence, we have $r_{Enc} \leq \theta\rho\zeta$.[1] Meanwhile, the result of [GH11] shows that $r_{Dec} \sim 2^{\eta}$. So we prove the following under the assumption that $\theta\rho\zeta \ll 2^{\eta}$.

We firstly show the correctness of the DECRYPT algorithm. Essentially, any ciphertext $\vec{\psi}$ is a vector close to $\mathcal{L}(\mathbf{V})$, and can be decrypted correctly as long as $r_{Enc} < r_{Dec}$. Without losing generality, we assume $\vec{\psi} = \vec{a} + \vec{b}$, for certain $\vec{a} \in \mathbb{Z}^n$, $\vec{b} \in \mathcal{L}$, $\|\vec{a}\| \leq \theta\rho\zeta$, where $\vec{a} = \sum_{i=1}^{\tau} \vec{r}_i \times \vec{s}_i + \vec{s}_{\tau+1}$. We firstly prove $\vec{\psi}' = \vec{a} \pmod 2$ as follows: Because $\vec{b} \in \mathcal{L}$, hence, $\vec{a} = \vec{\psi} \bmod \mathbf{V} = \vec{\psi} - \lfloor \vec{\psi} \cdot \mathbf{V}^{-1} \rceil \cdot \mathbf{V}$. Since $\mathbf{V}^{-1} = \mathbf{W}/d$, and $\mathcal{L}(\mathbf{V})$

[1] This value is indeed an upper bound. The actual r_{Enc} is expected to be much smaller. We provide more details in chapter 10.4.

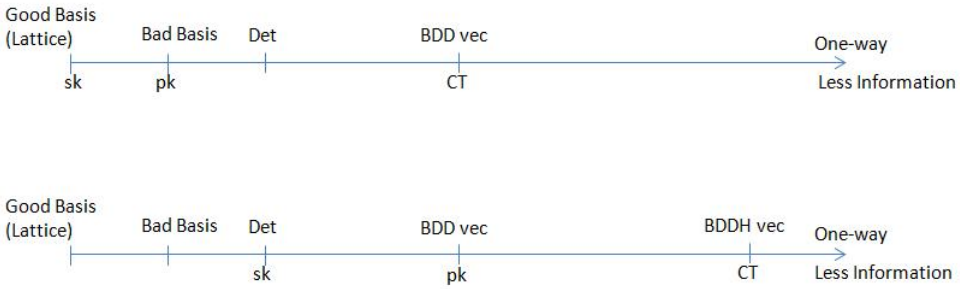

Figure 9.1: Why BDDH is better?

and $\mathcal{L}(\langle 2 \rangle)$ are coprime, we obtain $\vec{a} \bmod 2 = \lfloor \vec{\psi} \cdot \mathbf{W}/d \rceil \bmod 2 = \lfloor \vec{\psi} \times \vec{w}/d \rceil \bmod 2$. Therefore, $\vec{\psi}' \bmod 2 = \sum_{i=1}^{\tau} \vec{r}_i \times \vec{s}_i + \vec{s}_{\tau+1} \bmod 2$. Then we show $\psi'(1) \bmod 2 = m$ as follows: $\psi'(1) \bmod 2 = \sum_{i=1}^{\tau} r_i(1)s_i(1) + s_{\tau+1}(1) \bmod 2 = r_\tau(1)s_\tau(1) = m \pmod 2$. Therefore, the DECRYPT algorithm is correct.

Then we prove the correctness of MULT. We omit the proof of ADD. Assume $\psi_1 = \vec{a}_1 + \vec{b}_1$ and $\psi_1 = \vec{a}_2 + \vec{b}_2$ for certain $\vec{a}_1, \vec{a}_2 \in \mathbb{Z}^n$, $\vec{b}_1, \vec{b}_2 \in \mathcal{L}$, $\|\vec{a}_1\|, \|\vec{a}_2\| \leq \theta\rho\zeta$, where $a_j(x) = \sum_{i=1}^{\tau} r_{j,i}(x)s_{j,i}(x) + s_{j,\tau+1}(x) \bmod f(x)$. Hence, $\psi(x) \leftarrow \psi_1(x) \times \psi_2(x) \bmod f(x) = a_1(x)a_2(x) + a_1(x)b_2(x) + a_2(x)b_1(x) + b_1(x)b_2(x) \bmod f(x)$. Since the vector form of $\psi(x) - a_1(x)a_2(x)$ is in $\mathcal{L}$, as long as $\|a_1(x)a_2(x)\| < r_{Dec}$, decrypting $\vec{\psi}$ will return $a_1(1)a_2(1) = m_1 \times m_2$. Hence, MULT is correct.

9.2.1 Comparisons

We start with a comparison between BDDH based encryptions and BDD based encryptions. As one shall see from Figure 9.1, good basis, bad basis, etc. are all notions associated with a lattice. Further, from a left notion, one is able to compute all the notions to the right, but not vise versa. The more it is to the right side, the less information it contains about the lattice.

For the classic BDD based encryption schemes, one usually uses a good basis as the secret key, a bad basis as the public key and BDD vectors as the ciphertexts. In comparison, in our construction, we use the determinant of the lattice as the secret key, the BDD vectors as the public key and the BDDH vectors as the ciphertext. And it can be considered that less information is given away from the lattice in our scheme, which in theory makes our problem no easier to solve than the counter part.

Now we compare the SHE schemes. We note that our SHE scheme is a generalization of the integer based scheme [vDGHV10] as well as the ideal lattice based scheme [Gen09a].

By setting $\tau = 1$ and let $\vec{r}_i = 0$, we obtain the ideal lattice based scheme. In this case, the lattice is not hidden anymore because the noise is zero. The public key of our scheme is a vector of the lattice, and the rotation of the vector forms a bad basis of the lattice, which will be used in the lattice based scheme. The rest of the construction follows the ideal lattice based scheme.

To obtain the integer based scheme, we set $n = 1$. Then the hidden lattice is a lattice with dimension 1. Its determinant is the secret key p used in the integer based scheme, while the public keys are τ number of integers $g_i p + r_i$.

We note that the LWE based schemes are still by far the most efficient ones. As mentioned earlier, it allows one to change the determinant of the lattice (modulus switching), and therefore, the growth of the noise is much slower than the lattice based scheme. Hence, essentially this technique boosts the system exponentially. Unfortunately, this technique cannot be directly applied to the lattice-based scheme. Therefore we omit the comparison with the LWE-based FHE schemes.

9.2.2 Semantic Security

The following theorem proves the semantic security[2] of our algorithm.

Theorem 9.1 *If an algorithm $\mathcal{A}$ breaks the semantic security with advantage ε, then there exists an algorithm $\mathcal{B}$ that solves the Dec α, β-BDDHi$_{n,\tau}$ with advantage of $\frac{\varepsilon}{8}$. The running time of $\mathcal{B}$ is polynomial in the running time of $\mathcal{A}$.*

Proof: Fix parameters $\rho, \eta, \gamma, \tau, \zeta, n$ as in KeyGen. Set $\alpha \leftarrow \rho$, $\beta \leftarrow \theta\zeta\rho$. Let $\{\vec{v}_i\}$ and $\vec{u}$ a decisional α, β-BDDHi$_{n,\tau}$ problem. Assume $\vec{v}_i = \vec{g}_i\mathbf{B} + \vec{r}_i$, where $\mathbf{B}$ is a basis of $\mathcal{L}$, $\|\vec{r}_i\| \leq \alpha$, $\vec{g}_i, \vec{r}_i \in \mathbb{Z}^n$. Call algorithm $\mathcal{A}$ with $m_b, b \in \{0, 1\}$ and $\{\vec{v}_i\}$. For $\vec{v}_\tau$, it has one chance in four that $\sum_{j=0}^{n} r_{i,j} \bmod 2 = 1$ and $\sum_{j=0}^{n} g_{i,j} \bmod 2 = 1$. We do not have parity requirement for $\sum_{j=0}^{n} r_{i,j} \bmod 2$ and $\sum_{j=0}^{n} g_{i,j} \bmod 2$ when $i \neq \tau$. Therefore, the input vectors have at least $\frac{1}{4}$ probability of being in the correct form of the public key if $\det(\{\vec{v}_i\})$ is odd. As a result, the overall probability is $\frac{1}{8}$.

Algorithm $\mathcal{A}$ then returns m_b^*. Algorithm $\mathcal{B}$ outputs $m_b^* + m_b + 1$. If the vectors are in the correct form of the public keys (meaning that for $\vec{v}_\tau$, we require $\sum_{j=0}^{n} r_{\tau,j} = 1$ and $\sum_{j=0}^{n} g_{\tau,j} = 1$), $m_b^* = m_b$ with a probability of $\frac{1}{2} + \varepsilon$. Meanwhile, if $\{\vec{v}_i\}$ is not in the correct form, $\mathcal{A}$ will return a random bits. Which implies the probability of $m_b^* = m_b$ is 1/2. Overall, $\mathcal{B}$ has an probability of $\frac{1}{8}(\frac{1}{2} + \varepsilon) + \frac{7}{8} \cdot \frac{1}{2} = \frac{1}{2} + \frac{\varepsilon}{8}$ to obtain correct result. Hence, the overall advantage is $\frac{\varepsilon}{8}$.

[2] As in Gentry's work [Gen09a], the definition is without the reference to the Evaluate algorithm.

9.2.3 Attacks

Practical Public Key Attack

In this section, we present the best known attack, which is an adaptation of the attack proposed in [vDGHV10] to solve the AGCD of k integers. We simply generalize the attack from a lattice dimension equal to 1 to any n. We note that the attack in [vDGHV10] was already a generalization of the AGCD_k attack in [HG01] from 2 integers to any number of integers.

$$\mathbf{B} = \begin{vmatrix} Id(\theta\rho) & Rot(\vec{\pi}_2) & Rot(\vec{\pi}_3) & \dots & Rot(\vec{\pi}_k) \\ 0 & Rot(\vec{\pi}_1) & 0 & \dots & 0 \\ 0 & 0 & Rot(\vec{\pi}_1) & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & Rot(\vec{\pi}_1) \end{vmatrix}$$

Let $\mathcal{L}(\mathbf{B})$ be a lattice with a basis matrix with $k \leq \tau$ public keys as above. $Id(a) = a \cdot \mathbf{I}_n$ where $\mathbf{I}_n$ is an $n \times n$ identity matrix. For $\vec{\pi}_1 = \vec{r}_1 + \vec{g}_1\mathbf{V}$ and any of $\vec{\pi}_i = \vec{r}_i + \vec{g}_i\mathbf{V}$ where $2 \leq i \leq k$, we have $\vec{\pi}_1\vec{g}_i - \vec{\pi}_i\vec{g}_1 = \vec{r}_1\vec{g}_i - \vec{r}_i\vec{g}_1$. Therefore, the lattice $\mathcal{L}(\mathbf{B})$ contains a vector $\vec{u}$ such that $\vec{u} = \langle \theta\rho\vec{g}_1, \vec{r}_2\vec{g}_1 - \vec{r}_1\vec{g}_2, \vec{r}_3\vec{g}_1 - \vec{r}_1\vec{g}_3, \dots, \vec{r}_k\vec{g}_1 - \vec{r}_1\vec{g}_k \rangle$. Finding such a vector breaks the public key security, since one can recover $\vec{r}_1$ form $\vec{g}_1$ and $\vec{\pi}_1$.

In our analysis, the capability of lattice reduction algorithms is expressed as a function of two minimas, i.e. $\frac{\lambda_2}{\lambda_1}$. For this attack to work, we need $\lambda_1 = \|\vec{u}\|$. it has been shown that the best lattice reduction algorithm cannot find $\vec{u}$ if $\frac{\lambda_2}{\lambda_1} < c^{nk}$ for some constant c. That is

$$\lambda_2(\mathcal{L}) < c^{nk}\|\vec{u}\|. \tag{9.1}$$

A recent work in [CN11] shows that the smallest c that a reduction algorithm is reachable is 1.009.[3]

Additionally, accordingly to Equation 2.1, we have

$$\lambda_2(\mathcal{L}) \leq \sqrt{nk}^{\frac{nk}{nk-1}} \left(\frac{\det(\mathcal{L})}{\|\vec{u}\|}\right)^{\frac{1}{nk-1}}. \tag{9.2}$$

Combining Inequations 9.1 and 9.2 , we obtain that $\vec{u}$ should not be found using a lattice reduction if

[3] In [CN11], the authors also provided an enumeration technique, that allows one to obtain $c = 1.009$ with 2^{35} operations, however, the enumeration technique only works for small dimensions, while in our case the dimension is approximately τn. Hence, enumeration technique is not applicable in our scenario.

$$\sqrt{nk}^{\frac{nk}{nk-1}} \det(\mathbf{B})^{\frac{1}{nk-1}} < c^{nk} \|\vec{u}\|^{\frac{nk}{nk-1}}.$$

Therefore, we need to guarantee that

$$\det(\mathbf{B}) < c^{nk(nk-1)} \|\vec{u}\|^{nk}. \tag{9.3}$$

We know that $\det(\mathbf{B}) = \rho^n \det(Rot(\vec{\pi}_1))^{k-1}$. Using the Hadamard upper bound [GvL96], we obtain $\det(Rot(\vec{\pi}_1)) \leq \|\vec{\pi}_1\|^n$, therefore, we have $\det(\mathbf{B}) \leq \rho^n \|\vec{\pi}_1\|^{n(k-1)}$, which is

$$\det(\mathbf{B}) \leq \rho^n (\theta \|\vec{g}_1\| \|\vec{v}\|)^{n(k-1)}. \tag{9.4}$$

Meanwhile, we have $\|\vec{u}\| > \theta\rho\|\vec{g}_1\|$, and therefore we can expect the attack to be successful if

$$\rho^n (\theta \|\vec{g}_1\| \|\vec{v}\|)^{n(k-1)} \geq c^{nk(nk-1)} (\theta\rho\|\vec{g}_1\|)^{nk}.$$

To relax the condition a bit further, the attack will be successful if $\rho^n (\theta \|\vec{g}_1\| \|\vec{v}\|)^{n(k-1)} \geq c^{n^2k^2} (\theta\rho\|\vec{g}_1\|)^{nk}$. As a result, we have the following equation:

$$\eta(k-1) \geq \log_2 \theta + nk^2 \log_2 c + \gamma + (k-1)\log_2 \rho, \tag{9.5}$$

which is

$$\log_2 c \leq -\frac{\gamma + \log_2 \theta + (k-1)(\log_2 \rho - \eta)}{nk^2}.$$

To allow the best advantage of the attacker (where the attacker can use c as great as possible), we need to maximize the right hand side of the inequation. Denote A the right hand side of the inequation, and let $\kappa = \frac{1}{k}$, then

$$A = -\frac{\gamma + \log_2 \theta + \log_2 \rho - \eta}{n} \kappa^2 - \frac{\log_2 \rho - \eta}{n} \kappa$$

Since the coefficient of κ^2 term is negative (because $\gamma > \eta$), the maximum value of A is achieved when

$$\kappa = \frac{\eta - \log_2 \rho}{2(\gamma + \log_2 \theta + \log_2 \rho - \eta)},$$

which is

$$k = \frac{2(\gamma + \log_2 \theta + \log_2 \rho - \eta)}{\eta - \log_2 \rho}.$$

Considering that $\log_2 \rho$ is neglible compared with η, hence we know that the best attack occurs when $k \sim O(\frac{\gamma}{\eta})$.

Remark 9.3 *The best attack occurs when $O(\frac{\gamma}{\eta})$ public keys are used. This is also the case with the AGCD attack against the integer based fully homomorphic encryption scheme.*

To sum up, we need Equation 3 to be false for all k between 2 and τ. Moreover, in our parameter settings, we have $\frac{\gamma}{\eta} > \tau$. Hence, to allow the greatest advantage to the attacker, he/she should use all public keys in this attack. Therefore, taking $\gamma \sim O(n\eta)$ and $\tau \sim O(n)$, we conjecture that the best attack requires a lattice dimension that is quadratic with n.

Best Known Message Attack

The best known message attack come from the idea of attacking a GGH type cryptosystem in [Ngu99], i.e., converting a BDD problem into finding a shortest non-zero vector in a certain lattice.

Let $\vec{u} \leftarrow \langle 1, \vec{s}_1, \ldots, \vec{s}_\tau, \vec{s}_{\tau+1} \rangle$.We know that $\vec{u}$ is a shortest non-zero vector in the lattice whose basis is shown as follows:

$$\mathbf{B} = \begin{vmatrix} 1 & 0 & 0 & 0 & \ldots & 0 & \vec{\psi} \\ 0 & I_n & 0 & 0 & \ldots & 0 & \textsc{Rot}(\vec{\pi}_1) \\ 0 & 0 & I_n & 0 & \ldots & 0 & \textsc{Rot}(\vec{\pi}_2) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & I_n & \textsc{Rot}(\vec{\pi}_\tau) \end{vmatrix}$$

Therefore, using an equivalent estimation as in the best known public key attack, we estimate that no lattice reduction will be able to find $\vec{u}$ from the lattice if the following equation holds:

$$\log_2 \theta + \gamma + \eta < \tau(n\tau - 1) \log_2 c + \tau \log_2 \zeta. \tag{9.6}$$

It is worth pointing out that for the message attack, the attacker is obliged to put all the public key in the lattice, which implies that the dimension of the attacking lattice is strictly $n\tau + 1$, which is slightly different from the public key attack.

Birthday Paradox on Public Keys

For two public keys $\vec{\pi}_i$ and $\vec{\pi}_j$, one can guess the corresponding noise $\vec{r}_i$ and $\vec{r}_j$, and construct two lattices $\mathcal{L}_1 \leftarrow \mathcal{L}(\textsc{Rot}(\vec{\pi}_i - \vec{r}_i))$ and $\mathcal{L}_2 \leftarrow \mathcal{L}(\textsc{Rot}(\vec{\pi}_j - \vec{r}_j))$. A collision will be found when $\mathcal{L}_1 = \mathcal{L}_2$, which implies $\mathcal{L}_1 = \mathcal{L}_2$ is the hidden lattice. To stop the birthday paradox attack, it requires that the number of possible $\vec{r}_i$ in a single public key is greater than $2^{\lambda/2}$.

Brute Force on Ciphertext

The ciphertext is protected by the noise $\vec{s}$. To attack the ciphertext, one guesses $\vec{s}$. Therefore, our scheme requires that the number of possible $\vec{s}$ to be greater than 2^λ.

AGCD Attack There is also another AGCD attack in [CNT12], which is a generalization of the partial AGCD attack presented in [CN12]. For two integers $c_1 = g_1 p + r_1$ and $c_2 = g_2 p + r_2$, the attack is to find the greatest common divisor of $\prod_{i=0}^{2^\gamma - 1}(c_1 - i)$ and $\prod_{i=0}^{2^\gamma - 1}(c_2 - i)$, where γ is the maximum bit-length of r_i. Then, one can recover p from this divisor. This attack runs in $O(2^\gamma)$ time. We note that this attack is not directly applicable to our scheme, since we are dealing with vectors instead of integers. The best adaption of this attack is to replace c_i with $\det(\text{ROT}(\pi_i))$, and guess r_i with all possible noise. Nevertheless, this attack will be no better than a birthday paradox attack, due to the extra cost of computing the product of the determinant.

9.2.4 Security Conjecture

In section 8.2, we have shown that a $\text{BDDH}_{n,\tau}$ is harder than a BDD_n problem. In the previous part of this chapter we also show that if there exists a $\text{BDD}_{O(n\tau)}$ solver, then one is able to solve $\text{BDDH}_{n,\tau}$. Generally speaking, we have the following relations between these problem, assuming $\tau \geq n$,

$$BDD_n \geq BDDH_n \geq BDD_{O(n^2)},$$

where $A \geq B$ means B is harder than A.

Thus, we know that BDDH_n is equivalent to a BDD_{n^ξ} where $1 \leq \xi \leq 2$, and our construction is based on BDD_{n^ξ}.

Conjecture 9.1 *If a $BDD_{O(n^2)}$ problem is secure, then $BDDH_{n,\tau}$ is also secure, if $\tau \geq n$.*

In chapter 10.4 we will present two parameter settings. We note that the first parameter setting does not rely on this security conjecture. However, using this conjecture will further improve the performance of the scheme as in our second parameter setting.

Chapter 10

A Bootstrappable Scheme

In this section, we show how to bootstrap our scheme. We firstly squash the decryption algorithm to obtain a low degree decryption polynomial. Then we show that our cryptosystem is able to evaluate this polynomial homomorphically.

10.1 the Squashed scheme

In order to bootstrap our scheme, we adopt the squashing technique used in Gentry's scheme.

Essentially, we need to evaluate $\vec{\psi} \times Rot(\vec{w})$. The multiplication circuit in the decryption algorithm is squashed into several additions. The squashed scheme takes as follows:

KeyGen*(pk, sk)

- Generate a vector $\vec{w}^* = \langle w_1^*, \ldots, w_n^* \rangle$, where $w_i^* \leftarrow \lceil 2^{\eta^*} \times w_i/d \rfloor$, and $\eta^* = \eta + \gamma + 1$ is an integer.
- For each coefficient w_i^* of vector $\vec{w}^* = \langle w_1^*, \ldots, w_n^* \rangle$, generate an l dimentional integer vector $\vec{y}_i = \langle y_{i,1}, \ldots, y_{i,l} \rangle$ and a binary vector $\vec{z}_i = \langle z_{i,1}, \ldots, z_{i,l} \rangle$, such that $w_i^* = \sum_{k=1}^{l} y_{k,i} z_{k,i}$, and the hamming weight of $\vec{z}_i$ is t.
- Repeat the last step for all coefficients of $\vec{w}^*$;
- Output $pk^* \leftarrow \{\vec{y}_k\}_{k=0}^{n-1}$ and $sk^* \leftarrow \{\vec{z}_k\}_{k=0}^{n-1}$.

Remark 10.1 *In our parameter setting, we use $t = 1$, which indicates that one of coefficients of $\vec{y}_i = \langle y_{i,1}, \ldots, y_{i,l} \rangle$ is w_i^*. To do so, one can randomly assign w_i^* to one of the coefficient of $\vec{y}_i$ and fill in the rest of the coefficients with approximately same length.*

SQUASH($\vec{\psi}, pk^*$)

- Given a ciphertext $\vec{\psi} = \langle \psi_0, \ldots, \psi_{n-1} \rangle$, for each coefficient, generate $\vec{x}_i = \psi_i \vec{y}_i / 2^{\eta^*}$ for $0 \leq i \leq n-1$, and keep ω precisions behind the decimal point;
- Output $\{\vec{x}_i\}_{i=1}^{n}$.

DECRYPT*($\{\vec{x}_i\}, sk^*$)

- $\psi_i^* \leftarrow [\vec{x}_i \vec{z}_i]$;
- $\vec{\psi}^* \leftarrow \langle \psi_1^*, \ldots, \psi_n^* \rangle$;
- $\vec{\psi}' \leftarrow$ row sum of the matrix $Rot(\vec{\psi}^*)$
- Output $m \leftarrow \psi'(1) \bmod 2$.

10.2 Analysis

10.2.1 Correctness

Our proof takes two stages. We firstly prove that the decryption is correct under the assumption that the roundoff will not introduce errors. Under this assumption, we have $\psi_i^* = \vec{x}_i \vec{z}_i = \psi_i \vec{y}_i \vec{z}_i / 2^{\eta^*} = \psi_i w_i^* / 2^{\eta^*} = \psi_i w_i / d$. Hence, the decryption is correct.

Now we show the requirement of our assumption. For the i-th coefficient, let $\Delta \leftarrow w_i^* - 2^{\eta^*} \times w_i / d$, then $w_i^* = \Delta + 2^{\eta^*} w_i / d$. Therefore $\psi_i^* = [\psi_i \times w_i^* / 2^{\eta^*}] = [\psi_i \times \Delta / 2^{\eta^*} + \psi_i \times w_i / d]$. The first term needs to be smaller than $1/2$. Since $\Delta < 1/2$ by definition, setting $2^{\eta^*} > \psi_i$ for all $i < n$ will guarantee there is no error in the decryption. As a result, our decryption algorithm is correct when $\eta^* \geq \eta + \gamma + 1$.

10.2.2 Security

On a high level, for each coefficient of the secret key w_i^*, $0 \leq i \leq n-1$, we squash it into a subset with l elements. To recover the secret key, an attacker is required to recover all w_i'-s. This is mainly due to the hidden ideal lattice we used. Since the attacker does not know the lattice, it is incapable of verifying if the w_i^* it recovered is correct. The only method that can be conducted is to recover all w_i^*-s and then to use them to decrypt a certain ciphertext and check the decryption correctness. As a

Number of Additions ($t \times n$)	Precisions of Floating Point (ω)	Degree of Decryption Polynomial (q)	Number of Monomials (m)
$2 \sim 3$	2	3	9
$4 \sim 7$	3	7	5145
$8 \sim 15$	4	15	$\sim 2^{34}$
$16 \sim 31$	5	31	$\sim 2^{75}$
$32 \sim 63$	6	63	$\sim 2^{176}$
...	...	...	...
$512 \sim 1023$	10	1023	$\sim 2^{3180}$

Table 10.1: Relations between the # Additions and the Decryption Polynomial

result, the security of our squashed scheme is $f(t,l)^n$, where $f(t,l)$ is the number of operations that required to solve a t,l-SSP, which equals to $\binom{l}{t}$ for a small set.

Conjecture 10.1 *The best attack to solve n joint t,l-SSP instance is via a brute force attack. The complexity is $\binom{l}{t}^n$.*

In fact, as stated earlier, this is another main advantage of hiding the lattice other than using smaller dimensions. In contrast, in Gentry and Halevi's implementation, one is required to provide security for each SSP. If one have recovered one coefficient, it is able to recover the whole secret key. This resulted into a big set of at least 1024 element for each w_i^* that has been adopted.

As we shall see in the next section, since we have increased the security by an exponential factor of n, we are able to use exponentially smaller sets (in terms of number of elements). For instance, we squash w_i^* into a set of only 6 elements, with only one of them being w_i^*. The attacker will have to decide which one out of the six is picked. As a result we obtain a security of 6^n.

However, for each coefficient of $\vec{\psi}'$, we will need to perform $t \times n$ additions, compared with only t additions in Gentry and Halevi's scheme. In this case, since the decryption is additions of $t \times n$ floating points, we need $\log_2(t \times n) + 1$ digits precision after the decimal point, as shown in Table 10.1. As a result, the degree of decryption polynomial, denoted by q, is increased.

10.3 Bootstrapability

The bootstrapability of the squashed scheme depends on the degree of the binary form of the decryption polynomial. The result of [GH11] shows that to evaluate m monomials with a degree q polynomial homomorphically, the noise of the resulting ciphertext is

around $\sqrt{m}(r_{Enc})^q$. Since we have previously shown that $r_{Enc} \leq \theta\rho\zeta$, now we evaluate the number of monomials.

Since our decryption algorithm has essentially the same structure with Gentry and Halevi's scheme, the following result of the squashed the decryption follows their observation. The number of degree-ℓ monomials in the squashed decryption algorithm is

$$m = \prod_{i=0}^{\lfloor \log_2 \ell \rfloor} \binom{\ell}{2^i}.$$

For instance, if $\ell = 31$, then $m = \binom{31}{1}\binom{31}{2}\binom{31}{4}\binom{31}{8}\binom{31}{16} \sim 2^{75}$.

Then the squashed decryption requires a multiplication between x_i and z_i. However, it is not necessary to encrypt x_i. Therefore, this multiplication does not increase the number of monomials. Finally, since we need to support a product of two homomorphically-decrypted bits, our scheme must support polynomials with m degree-ℓ monomials. As a result, our scheme is expected to be able to handle a homomorphic decryption plus one more multiplication/addition if Equation 10.1 holds.

$$2^\eta \geq \sqrt{m}(\theta\rho\zeta)^\ell. \tag{10.1}$$

10.4 Parameters

Now, we provide two sets of parameters. In the first set we have $\xi = 1$. We do not rely on the security conjecture. In this case, we work on hidden lattices which dimension are quite large, and as a result, we only need a constant number of public keys. Essentially it is quite close to Gentry-Halevi's scheme. The major difference is that our lattice is hidden, so we do not rely on the SSP. As a result, we have a smaller circuit depth, which results in smaller parameters.

For the second set, we rely on our conjecture. We work on hidden lattices with small dimensions, and the number of public keys is approximately the same as the dimension. In the configuration, we use $c = 1.007$, which implies that no efficient reduction algorithm should be able to find vectors smaller than $c^n \det^{\frac{1}{n}}$ with sufficiently large n. Nevertheless, one should adapt c with the development of new reduction algorithms.

10.4.1 Parameter Setting 1

From subsection 8.2 it is straightforward to see that the BDDH problem is harder than the corresponding BDD problem over the lattices with the same dimension for the same parameters. Therefore, we propose a set of parameters of our FHE scheme where even to solve the BDD problem is hard.

Similar to [GH11], we do not provide an asymptotic complexity. Table 10.2 highlights the parameters for security levels 2^{80}. Since our parameters are bounded by several requirements, and most parameters are interconnected, we tested all the possible secure combinations to achieve the smallest public key size.

We use $\lambda = 80$ as an example. It requires a lattice with dimension 1024, with 2 public keys. To stop the birthday paradox attack, we allow 5 coefficients of each noise for each π_i to be -1, or 1, while the rest are 0. Hence, the maximum norm of the noise is $\sqrt{5}$. This allows $\binom{1024}{5} > 2^{\lambda/2}$ possibilities. As for the security of the encryption noise $\vec{s}$, $\vec{s}$ maintains $\tau + 1$ blocks. Each block has n coefficients. Further, the parity of each block (i.e., the sum of all bits) follows the requirement of the ENCRYPT algorithm. We set maximum 5 coefficients to be 1 or -1 (4 coefficients if the encrypted message is 0), with the rest 0-s. As a result, there are maximum 5 blocks with non-zero entries besides the τ-th block. The total possibility is at least $\binom{\tau+1}{5}(\binom{n}{2}2^2)^5 > 2^{80}$. Hence, we are secure against the brute force attack. We also use a $1, 2$-SSP setting to achieve a 2^{1024} security level of the squashed secret keys. This setting gives us a decryption polynomial of degree 3 as we stated eailer.

Now we look into more details of r_{Enc}. Recall that $\psi(x) = \sum_{i=1}^{\tau} s_i(x)\pi_i(x) + s_{\tau+1}(x) \bmod f(x)$. Therefore, the maximum noise of each ψ_i is 5, since $\vec{s}$ contains only 5 non-zero coefficients. As a result, the worst case $r_{Enc} = \sqrt{1024}\sqrt{5}\sqrt{5}$, although in most cases, it will be much smaller.

To bootstrap, we need $2^{\eta} \geq \sqrt{m}(r_{Enc})^{1023}$, where $m = \binom{1023}{1}\binom{1023}{2}\cdots\binom{1023}{512} \sim 2^{3180}$. Therefore, the setting $\eta = 9080$ will allow us to bootstrap. Having set all parameters as above, we choose the smallest γ allowed according to Equation 3 and 4. As a result, we obtain parameters in the second row of Table 10.2. For the completeness, we also list the parameters for security level $\lambda = 128$ and 160.

10.4.2 Parameter Setting 2

As for the conjectured security, we assume that the best known attack works on a lattice which dimension is quadratic with the hidden lattice. Therefore, we build our

	ρ	η	γ	τ	ζ	n	t	l	Conjecture
$\lambda = 80$	$\sqrt{5}$	9080	1280000	310	$\sqrt{5}$	1023	1	2	N
$\lambda = 80$	$\sqrt{10}$	222	18255	111	$\sqrt{11}$	31	1	6	Y
$\lambda = 128$	8	595	88411	301	$\sqrt{17}$	63	1	5	Y
$\lambda = 160$	8	604	91127	307	$\sqrt{21}$	63	1	6	Y

Table 10.2: Parameters

hidden lattice which dimension is square root of the dimension where a normal lattice problem is secure. However, due to the fact that the dimension is smaller than the proved scheme, the noise in each vector has to be increased to deliver the same security.

	GH Scheme $\lambda = 72$	Our Parameter Setting 2 $\lambda = 80$
Lattice dimension	2048	31
Lattice determinant	$2^{780,000}$	$2^{573,000}$
Set size	1024	6
Subset size	15	1
Public key size (Mbits)	552	173.5
Ciphertext size (kbits)	780	573

Table 10.3: Comparisons with Gentry and Halevi's implementation

We also use $\lambda = 80$ as an example. The parameters for security level $\lambda = 128$ and 160 are listed in Table 10.2. For $\lambda = 80$ it requires a lattice with dimension 31, with 57 public keys. To stop the birthday paradox attack, we allow the coefficient of each noise for each π_i to be $-1, 0$, or 1, so the maximum norm of the noise is $\sqrt{32}$. This allows $3^{32} > 2^{\lambda/2}$ possibilities. Further, $\vec{s}$ maintains $\tau + 1$ blocks. Each block has 32 coefficients. We set maximum 11 coefficients to be 1 or -1 (10 coefficients if the encrypted message is 0), with the rest 0-s. As a result, there are maximum 5 blocks with non-zero entries besides the τ-th block. The total possibility is at least $\binom{\tau+1}{5}(\binom{n}{2}2^2)^5 > 2^{80}$. Hence, we are secure against the brute force attack. We also use a $1, 6$-SSP setting to achieve a $6^{32} \sim 2^{82}$ security level of the squashed secret keys.

As for the r_{Enc}, the maximum noise of each ψ_i is 11, since $\vec{s}$ contains only 11 non-zero coefficients. As a result, the worst case $r_{Enc} = 11\sqrt{32}$. To bootstrap, it requires $2^\eta \geq \sqrt{2^{75}}(11\sqrt{31})^{31}$, which gives us $\eta = 222$. Finally, we choose the smallest γ allowed according to Equation 3 and 4. As a result, we obtain parameters in the third row of

Table 10.2.

10.4.3 Comparisons

Here we focus on the performance of our second parameter setting. We omit the comparison with parameter setting 1, as this setting is highly inefficient as stated earlier.

For our conjectured cryptosystem, our SHE scheme uses a ciphertext size of $(222 + 18255) \times 31 \sim 573$ kbits. The SHE also uses a public key space of 111×573 kbits ~ 63.6 Mbits. The squashed scheme uses a public key of size $6 \times 31 \times (222 + 18255)$ bits ~ 3.4 Mbits. Finally, one needs to encrypt the squashed secret key, which further adds another $6 \times 31 \times 573$ kbits ~ 106.5 Mbits. As a result, our whole system uses a public key size of $63.6 + 3.4 + 106.5 \sim 173.5$ Mbits.

To compare with the van Dijk et al.'s scheme, it is almost straightforward to see that we are more efficient, since their scheme works with approx $2^{31.6}$ integers, each of the length $2^{31.6}$ bits, to obtain a security level of $\lambda = 80$. Therefore we mainly compare our scheme with Gentry and Halevi's implementation.

To compare with Gentry and Halevi's implementation, our first parameter setting uses similar parameters with their SHE scheme. Nevertheless, due to the fact that the lattice is hidden, we achieve a better bootstappability by removing the necessity of relying on the SSP problem. While for our conjectured version of the scheme, we improve the efficiency by both the reducing the dimension and removing the SSP. To complete this subsection, we list the parameters used in Gentry and Halevi's implementation with $\lambda = 72$ (see [GH11]). It is easy to see that ours is more efficient than Gentry and Halevi's scheme in terms of space. The running time of the system is mainly influenced by the size of the ciphertext and the squashed decryption polynomial (both degree and number of monomials). We also note that those parameters in our scheme are smaller than Gentry and Halevi's system, therefore, it is straightforward to see that the running time of our scheme is better.

Part IV

Conclusion and Future Work

We briefly reiterate our main contributions.

- **An adaptive precision floating-point LLL algorithm**.
  In chapter 3, we proposed an improvement of LLL algorithm can be used for all purposes of lattice reduction. It accelerates the procedure by 20% on average, and its asymptotic complexity remains the same with classic algorithms.

- **A recursive reduction algorithm for knapsack-type lattice basis**.
  In chapter 4, we present a method of recursive reduction. It can be applied over a knapsack-type basis, and it successfully improves the asymptotic complexity from $O(d^{3+\varepsilon}\beta^2 + d^{4+\varepsilon}\beta)$ to $O(d^{2+\varepsilon}\beta^2 + d^{4+\varepsilon}\beta)$.

- **An dedicated LLL algorithm for ideal lattices**.
  In chapter 5 we present the iLLL algorithm. It can be applied over ideal lattice basis and bases with similar structures. Combining this technique with ap-fplll, we predict to solve the Gentry Halevi's fully homomorphic encryption challenge in 15.7 years. The details of the results are presented in chapter 6.

- **A CCA-1 attack against fully homomorphic encryption schemes using integers**.
  In chapter 7, we show a CCA-1 attack against fully homomorphic encryption scheme with integers. We also present a reaction attack that can be used to construct a decryption oracle when a fully homomorphic encryption scheme is used in outsourced computing.

- **A new fully homomorphic encryption scheme using hidden lattice**.
  In Chapter III we presented a new fully homomorphic encryption scheme using hidden lattice. We based our scheme on a hidden lattice problem, which unifies the problems that the ideal lattice based and integer based schemes are relying on. It also delivers the best performance among the three under a security conjecture. Hence, it is the best alternative to the learning with error based fully homomorphic encryption schemes.

In this thesis, we have reviewed the state-of-art fully homomorphic encryption schemes and their cryptographic primitives. Although the results presented in this thesis have several new techniques towards the fully homomorphic encryptions, it could be further developed in a number of ways:

- **A better precision strategy for $\mathbf{L}^2$ algorithm**.
  As we have stated earlier, the update strategy for floating points in this thesis might not be the best solution. Further research needs to be done to confirm if it is the best, or to find the best solution.

- **Cryptanalysis NTRUencryption scheme with iLLL-type algorithm**
  To cryptanalysis an NTRUencryption scheme one usually starts with reducing a Coppersmith-Shamir basis. The proposed iLLL algorithm can be applied directly over this kind of bases to accelerate reduction. However, in terms of cryptanalysis, the LLL type algorithms are not strong enough, and one usually need to do a BKZ reduction. Thus, it would be interesting to see if the rotation method in the iLLL algorithm can be applied over other lattice reduction algorithms and potentially produces better results.

- **A better reduction between hidden lattice problems and classic lattice problems**.
  In this thesis, we have shown that a BDDH problem over a dimension n^{ε} lattice is equivalent to a BDD problem over a dimension n normal lattice for $\varepsilon \geq 1$. The BDDH problem enables our fully homomorphic encryption scheme, and the parameters of our scheme rely on the hardness of this problem. So if there exists a better reduction between the problems, (for instance, $\varepsilon \geq 1.5$) then we shall be able to use more loose parameters which will make our scheme more efficient.

Bibliography

[BBD08] Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen. *Post Quantum Cryptography*. Springer Publishing Company, Incorporated, 2008.

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. the user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997.

[BDPR98] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In *CRYPTO*, pages 26–45, 1998.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, pages 97–106, 2011.

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from Ring-LWE and security for key dependent messages. In *CRYPTO*, pages 505–524, 2011.

[CJL+92] Matthijs J. Coster, Antoine Joux, Brian A. LaMacchia, Andrew M. Odlyzko, Claus-Peter Schnorr, and Jacques Stern. Improved low-density subset sum algorithms. *Computational Complexity*, 2:111–128, 1992.

[CMNT11] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In *CRYPTO*, pages 487–504, 2011.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In *ASIACRYPT*, pages 1–20, 2011.

[CN12] Yuanmi Chen and Phong Q. Nguyen. Faster algorithms for approximate common divisors: Breaking fully-homomorphic-encryption challenges over the integers. In *EUROCRYPT*, pages 502–519, 2012.

[CNT12] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 446–464, 2012.

[Cop96a] Don Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In *EUROCRYPT*, pages 178–189, 1996.

[Cop96b] Don Coppersmith. Finding a small root of a univariate modular equation. In *EUROCRYPT*, pages 155–165, 1996.

[CS97] Don Coppersmith and Adi Shamir. Lattice attacks on ntru. In *EUROCRYPT*, pages 52–61, 1997.

[Gen09a] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[Gen09b] Craig Gentry. *A Fully Homomorphic Encyrption Scheme*. PhD thesis, Stanford University, September 2009.

[Gen10a] Craig Gentry. Computing arbitrary functions of encrypted data. *Commun. ACM*, 53(3):97–105, 2010.

[Gen10b] Craig Gentry. Toward basing fully homomorphic encryption on worst-case hardness. In *CRYPTO*, pages 116–137, 2010.

[GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In *CRYPTO*, pages 112–131, 1997.

[GH] Craig Gentry and Shai Halevi. Public Challenges for Fully-Homomorphic Encryption. online. `http://researcher.watson.ibm.com/researcher/view_project.php?id=1548`.

[GH11] Craig Gentry and Shai Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In *EUROCRYPT*, pages 129–148, 2011.

[GHPS12] Craig Gentry, Shai Halevi, Chris Peikert, and Nigel P. Smart. Ring switching in bgv-style homomorphic encryption. In *SCN*, pages 19–37, 2012.

[GHS12a] Craig Gentry, Shai Halevi, and Nigel P. Smart. Better bootstrapping in fully homomorphic encryption. In *Public Key Cryptography*, pages 1–16, 2012.

[GHS12b] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *EUROCRYPT*, pages 465–482, 2012.

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. i-Hop Homomorphic Encryption and Rerandomizable Yao Circuits. In *CRYPTO*, pages 155–172, 2010.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[GM06] Daniel Goldstein and Andrew Mayer. On the equidistribution of hecke points. In *Forum Mathematicum.*, volume 15, pages 165–189, 2006.

[GN08] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In *Proceedings of the theory and applications of cryptographic techniques 27th annual international conference on Advances in cryptology*, EUROCRYPT'08, pages 31–51, Berlin, Heidelberg, 2008. Springer-Verlag.

[GvL96] Gene H. Golub and Charles F. van Loan. *Matrix computations (3. ed.)*. Johns Hopkins University Press, 1996.

[HG01] Nick Howgrave-Graham. Approximate integer common divisors. In *CaLC*, pages 51–66, 2001.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. Ntru: A ring-based public key cryptosystem. In *ANTS*, pages 267–288, 1998.

[HPS11a] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Algorithms for the shortest and closest lattice vector problems. In *IWCC*, pages 159–190, 2011.

[HPS11b] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Terminating bkz. *IACR Cryptology ePrint Archive*, 2011:198, 2011.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):pp. 203–209, 1987.

[KS01] Henrik Koy and Claus-Peter Schnorr. Segment lll-reduction of lattice bases. In *CaLC*, pages 67–80, 2001.

[Lai01] Ming Kin Lai. Knapsack cryptosystems: The past and the future. 2001.

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen, Springer-Verlag*, 261:513–534, 1982.

[LM08] Vadim Lyubashevsky and Daniele Micciancio. Asymptotically efficient lattice-based digital signatures. In *TCC*, pages 37–54, 2008.

[LM09] Vadim Lyubashevsky and Daniele Micciancio. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In *CRYPTO*, pages 577–594, 2009.

[LMSV11] J. Loftus, A. May, N.P. Smart, and F. Vercauteren. On CCA-Secure fully homomorphic encryption. In *Selected Areas in Cryptography*, 2011.

[LNV11] Kristin Lauter, Michael Naehrig, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *ccsw*, 2011.

[LO85] J. C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1):229–246, 1985.

[Lov86] L. Lovász. *An Algorithmic Theory of Numbers, Graphs and Convexity*, volume 50 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM Publications, 1986.

[MG02] D. Micciancio and S. Goldwasser. *Complexity of Lattice Problems, A Cryptographic Perspective*. Kluwer Academic Publishers, 2002.

[Mic07] Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *Computational Complexity*, 16(4):365–411, 2007.

[Min96] H. Minkowski. *Geometrie der Zahlen*. B. G. Teubner, Leipzig, 1896.

[mpf] mpfr library. online. `http://www.mpfr.org/`.

[MSV09] I. Morel, D. Stehle, and G. Villard. H-lll: using householder inside lll. In *ISSAC*, pages 271–278, 2009.

[MW00] Daniele Micciancio and Bogdan Warinschi. A linear space algorithm for computing the hermite normal form. *Electronic Colloquium on Computational Complexity (ECCC)*, 7(74), 2000.

[Ngu99] Phong Q. Nguyen. Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from crypto '97. In *CRYPTO*, pages 288–304, 1999.

[Ngu11] Phong Q. Nguyen. Breaking fully-homomorphic-encryption challenges. In *CANS*, pages 13–14, 2011.

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

[NS05a] P. Q. Nguyen and D. Stehle. Floating-point LLL revisited. In *Advances in Cryptology - Eurocrypt 2005, Lecture Notes in Computer Science 3494, Springer-Verlag*, pages 215–233, 2005.

[NS05b] Phong Q. Nguyen and Jacques Stern. Adapting density attacks to low-weight knapsacks. In *ASIACRYPT*, pages 41–58, 2005.

[NS06] P. Q. Nguyen and D. Stehle. LLL on the average. In *7th International Symposium on Algorithmic Number Theory (ANTS 2006)*, pages 238–256, 2006.

[NSV11] Andrew Novocin, Damien Stehlé, and Gilles Villard. An lll-reduction algorithm with quasi-linear time complexity: extended abstract. In *STOC*, pages 403–412, 2011.

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

[Nym75] J. E. Nymann. On the probability that k positive integers are relatively prime ii. *Journal of Number Theory*, 7(4):406–412, 1975.

[PSC] Xavier Pujol, Damien Stehle, and David Cade. fplll library. online. `http://perso.ens-lyon.fr/xavier.pujol/fplll/`.

[RAD78] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–177. Academic Press, 1978.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.

[Reg10] Oded Regev. Learning with errors over rings. In *ANTS*, page 3, 2010.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[Sch88] Claus-Peter Schnorr. A more efficient algorithm for lattice basis reduction. *J. Algorithms*, 9(1):47–62, 1988.

[Sch06] Claus-Peter Schnorr. Fast lll-type lattice reduction. *Inf. Comput.*, 204(1):1–25, 2006.

[SE94] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994.

[Sho] Victor Shoup. NTL - A Library for Doing Number Theory. online. `http://www.shoup.net/ntl/index.html`.

[Sho94] Peter W. Shor. Polynominal time algorithms for discrete logarithms and factoring on a quantum computer. In *ANTS*, page 289, 1994.

[SS10] Damien Stehlé and Ron Steinfeld. Faster fully homomorphic encryption. In *ASIACRYPT*, pages 377–394, 2010.

[SS11] Peter Scholl and Nigel P. Smart. Improved key generation for gentry's fully homomorphic encryption scheme. In *IMA Int. Conf.*, pages 10–22, 2011.

[Ste10] Damien Stehlé. Floating-point lll: Theoretical and practical aspects. In Phong Q. Nguyen and Brigitte Vallée, editors, *The LLL Algorithm*, Information Security and Cryptography, pages 179–213. Springer, 2010.

[SV10] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography*, pages 420–443, 2010.

[svp] SVP CHALLENGE. online. `http://www.latticechallenge.org/svp-challenge/index.php`.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43, 2010.

[vHN10] Mark van Hoeij and Andrew Novocin. Gradual sub-lattice reduction and a new complexity for factoring polynomials. *CoRR*, abs/1002.0739, 2010.