Clemson University
TigerPrints

All Theses Theses

8-2022

Efficiency of Homomorphic Encryption Schemes

Kyle Yates
kjyates@clemson.edu

Follow this and additional works at: https://tigerprints.clemson.edu/all_theses

Part of the Algebra Commons, Number Theory Commons, and the Other Applied Mathematics Commons

Recommended Citation
Yates, Kyle, "Efficiency of Homomorphic Encryption Schemes" (2022). *All Theses*. 3868.
https://tigerprints.clemson.edu/all_theses/3868

This Thesis is brought to you for free and open access by the Theses at TigerPrints. It has been accepted for inclusion in All Theses by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.

Efficiency of Homomorphic Encryption Schemes

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
School of Mathematical and Statistical Sciences

---

by
Kyle Jacob Yates
August 2022

Accepted by:
Dr. Shuhong Gao, Committee Chair
Dr. Ryann Cartor
Dr. Felice Manganiello
Dr. Hui Xue

Abstract

In 2009, Craig Gentry introduced the first fully homomorphic encryption scheme using bootstrapping [11]. In the 13 years since, a large amount of research has gone into improving efficiency of homomorphic encryption schemes. This includes implementing leveled homomorphic encryption schemes for practical use, which are schemes that allow for some predetermined amount of additions and multiplications that can be performed on ciphertexts. These leveled schemes have been found to be very efficient in practice. In this thesis, we will discuss the efficiency of various homomorphic encryption schemes. In particular, we will see how to improve sizes of parameter choices in homomorphic encryption schemes with a variety of techniques to include modulus leveling and techniques of error bound control.

Acknowledgments

As with any work of this size, I could not have have completed it without the encouragement and contributions of many others. First, I would like to thank my advisor Dr. Shuhong Gao for overseeing and guiding me through this process of researching and writing my thesis, as well as providing guidance through various other undertakings during my first two years of graduate school. I would also like to thank my other committee members Dr. Ryann Cartor, Dr. Felice Manganiello, and Dr. Hui Xue for their input on this thesis.

I want to thank Marvin Jones for all of the helpful discussions we've had on cryptography. I would also like to thank Yu-Chung Liu and Dr. Yuyuan Ouyang for the meaningful research discussions I had with them. I'd like to thank the undergraduate students I had the pleasure of working with during the 2021 Clemson REU in cryptography, which was my introduction to the topic of homomorphic encryption. These students are Ivan Hu, Basia Klos, Vir Pathak, and Joshua Ding. I'd also like to thank my undergraduate advisor at San Diego State University, Dr. Carmelo Interlando, for first introducing me to the field of cryptography.

Table of Contents

Title Page . . . . **i**

Abstract . . . . **ii**

Acknowledgments . . . . **iii**

List of Figures . . . . **vi**

1 Introduction **1**
- 1.1 Homomorphic Encryption and Motivation 2
- 1.2 Efficiency 3
- 1.3 Overview of Thesis 4

2 Background **5**
- 2.1 Cryptography 5
- 2.2 Lattices 6
- 2.3 Cyclotomic Polynomials and Rings 9
- 2.4 Notation Overview 10

3 Basic Homomorphic Encryption **12**
- 3.1 Learning With Errors 12
- 3.2 Homomorphic Encryption Schemes 13
- 3.3 BFV 14
- 3.4 Computing $c_2's^2$ without knowing s^2 23
- 3.5 BGV 30
- 3.6 CKKS 35
- 3.7 Remarks on Differences of HE Cryptosytems 41

4 Advanced Computing Techniques **42**
- 4.1 Modulus Reduction and Leveling 42
- 4.2 RNS BFV 48
- 4.3 DFT's, NTT's, and FFT's 57
- 4.4 Non-arithmetic Operations 59

5 Error Bounds **60**
- 5.1 BFV Error 60
- 5.2 The Case when $t = 2$ 65
- 5.3 The Case when $t|(q-1)$ 69
- 5.4 CKKS Error 74

Appendices **78**
- A Lattice Reduction Algorithms 79

Bibliography **83**

List of Figures

2.1 A lattice with a good basis . . . 7
2.2 A lattice with a bad basis . . . 7
2.3 Shortest vector problem . . . 8
2.4 Closest vector problem . . . 9

3.1 BFV Encryption Algorithm . . . 15
3.2 BFV Decryption Algorithm . . . 16
3.3 BFV Addition . . . 17
3.4 BFV Initial Multiplication . . . 19
3.5 Flattened Evaluation Key Generation . . . 24
3.6 Relinearization Version 1 . . . 25
3.7 Scaled Evaluation Key Generation . . . 26
3.8 Relinearization Version 2 . . . 27
3.9 BFV Multiplication Version 1 . . . 30
3.10 BFV Multiplication Version 2 . . . 30
3.11 BGV Encryption Algorithm . . . 31
3.12 BGV Decryption Algorithm . . . 32
3.13 BGV Addition . . . 33
3.14 BGV Initial Multiplication . . . 33
3.15 BGV Multiplication Version 1 . . . 34
3.16 BGV Multiplication Version 2 . . . 35
3.17 CKKS Encryption Algorithm . . . 38
3.18 CKKS Decryption Algorithm . . . 38
3.19 CKKS Addition . . . 39
3.20 CKKS Initial Multiplication . . . 39
3.21 CKKS Multiplication Version 1 . . . 40
3.22 CKKS Multiplication Version 2 . . . 40

4.1 BFV Modulus Reduction Algorithm . . . 43
4.2 BFV Modulus Reduction Encryption . . . 46
4.3 CKKS Rescaling . . . 47
4.4 BFV RNS Conversion Algorithm . . . 50
4.5 BFV RNS Scaling . . . 52
4.6 BFV Complex CRT Scaling . . . 53
4.7 BFV RNS Conversion Algorithm . . . 53
4.8 RNS Flattened Evaluation Key Generation . . . 55

1 Gaussian Lattice Reduction Algorithm . . . 80
2 The LLL Algorithm . . . 81

Chapter 1

Introduction

In the modern world of digital communication, data security is more important than ever. *Cryptography* is the study of secure communication. More specifically, cryptography studies secure communication in the presence of third-party adversaries who may try and steal or compromise sensitive data. The underlying algorithms working to keep data secure are heavily based on mathematical theory. In particular, the areas of abstract algebra, number theory, and probability theory are relevant in designing secure communication tactics. The first chapter of any cryptography textbook will introduce and explain the basic underlying problem and solution of secure data communication through the classic example of Alice and Bob.

Suppose there are two people, Alice and Bob. Alice has a message that she wants to send to Bob, but she does not want anybody other than Bob to read the message. The message that Alice wants to send must travel through an insecure channel, meaning that if she just sends her message plainly, somebody could intercept the message and read it. The question becomes this: how can Alice send a secure message to Bob over an insecure channel?

Instead of just sending the message plainly, Alice will disguise the message through a process known as *encryption*. This disguised message is also known as an *encrypted message*. When Bob receives the encrypted message, he will revert the message back to its original form through a process known as *decryption*, and then read the message. The key to this process is ensuring that Alice disguises the message in such a way that only Bob can decrypt it. This is achieved through a *private key*, which is some information that only Bob and Alice possess that can be used to encrypt and decrypt the message. The encrypted message is constructed in such a way that without the

information of the private key, decrypting the message is computationally infeasible. This way, if the message is intercepted by a third party during transmission, the third party can not read the message. This process of encrypting and decrypting information through a shared private key is known as *symmetric key cryptography.*

With symmetric key cryptography, Alice and Bob both share the same private key. Let's suppose now that Bob wants to receive and read an encrypted message from Alice, but does not want to share the private key with Alice. In addition to the private key that only he has, Bob makes available a *public key.* This public key can only be used in one direction. It can be used to encrypt messages, but can not be used to decrypt messages. Only the private key that Bob possesses can be used to decrypt a message. In this scenario, the public key is made available to the public. This way, anybody can encrypt and send a secure message to Bob, but only Bob can decrypt and read the message. This scenario is known as *public key cryptography.* The collection of algorithms used to encrypt messages, decrypt messages, and generate private and public keys is known as a *cryptosystem.*

1.1 Homomorphic Encryption and Motivation

Suppose now that Alice wants to encrypt multiple messages to send to Bob. Not only that, but Alice also wants computation to be done on these encrypted messages. For instance, let's suppose Alice has two sets of data that she wants to encrypt. She wants to encrypt the two separately, and then add or multiply the encrypted data with each other. This should yield the same result as if she added or multiplied them together first, and then encrypted them. This concept of allowing for mathematical operations on encrypted messages is known as *homomorphic encryption.*

Practically, this specific example does not make much sense. Why would Alice bother trying to find a way to compute on encrypted data, when she can simply perform the computations herself on the plain data, and then encrypt and send that result? In this case, that is the far easier and more efficient route for Alice. There are other scenarios in which homomorphic encryption is useful in a practical sense however. For instance, one application is that of secure cloud computing. If Alice has data for which she wants to carry out computations, but does not have the proper computing capacity, she can send her data to the cloud or some alternate computing center to compute on her data for her. If she uses a homomorphic encryption scheme, she can have the cloud carry out her

computations without any information being leaked. This way, she gets her desired computations on her data, but nobody else gets information about her data.

Another motivation of homomorphic encryption is secure data collection. Consider a scenario in which, for instance, the CDC is conducting a study about patients in a number of various hospitals. In this scenario, the CDC has the private key, whereas the hospitals all possess a public key. The first hospital will encrypt their patient data, and send it to the second hospital. The second hospital, after encrypting their own data, will now apply the necessary operations between their data and the first hospital's data, and then send the result to the third hospital. The third hospital will encrypt and add their data to the mix, and forward the result along, and so on. After the final hospital adds in their data, they will send the pooled encrypted data to the CDC, who will then decrypt the final result with the secret key. In this way, the CDC obtains the finals result of all the combined data, but does not have information on any individual hospital's contribution to the data. Furthermore, the individual hospitals do not have information on the final result of the data, nor do they have any information about any other hospital's data.

1.2 Efficiency

As with any cryptosystem, the difficulty in homomorphic encryption comes with balancing efficiency and security. Cryptosystem parameters need to be large enough to ensure security, but small enough to efficiently encrypt, send, and decrypt data. This is even more relevant in homomorphic encryption, as we not only need to encrypt and decrypt data, but also compute on data. Parameter selection must therefore be carefully considered.

Various techniques have been introduced to speed up computations and algorithms used in homomorphic encryption. These techniques include modulus leveling, residue number systems, and error control. These techniques will be discussed in chapters 4 and 5 of this thesis. Although we are concerned with efficiency in HE, this thesis will focus on techniques used to improve parameter choices rather than discussing actual runtimes of algorithms. The goals of this thesis are as follows:

1. Explain and survey the main schemes currently used in homomorphic encryption.

2. Describe techniques of improving computational efficiency in homomorphic encryption schemes.

3. Analyze and improve worst-case error bounds in homomorphic encryption schemes.

4. Improve parameter choices of homomorphic encryption schemes.

Although we outline all three of the main homomorphic encryption schemes, the focus in our improvements will be on the Brakerski-Fan-Vercauteren scheme.

1.3 Overview of Thesis

The remainder of this thesis will follow the outline given below.

Chapter 2: Background will discuss the necessary mathematical background needed for homomorphic encryption. This will include basic descriptions of private and public key cryptosystems, lattices, and cyclotomic polynomials. This chapter will also give an overview of the notation used in this thesis.

Chapter 3: Basic Homomorphic Encryption will introduce the underlying problem in homomorphic encryption schemes, as well as a basic overview of the three main homomorphic encryption schemes. This includes the Brakerski-Fan-Vercauteren (BFV), Brakerski-Gentry-Vaikuntanathan (BGV), and Cheon-Kim-Kim-Song (CKKS) schemes.

Chapter 4: Advanced Computing Techniques will expand on the schemes presented in Chapter 3 and will introduce more advanced techniques to improve efficiency of homomorphic encryption schemes. This chapter will discuss modulus reduction, RNS variants of systems, and computational techniques. We will also briefly discuss non-arithmetic operations in homomorphic encryption.

Chapter 5: Error bounds will discuss existing work as well as present new analysis on ciphertext noise. In particular, we will discuss error growth during homomorphic operations, improve error bounds, compare our parameters to existing parameters, and discuss accuracy of fixed precision representation in homomorphic encryption.

Chapter 2

Background

2.1 Cryptography

In chapter 1, we outlined the basic concepts in cryptography. Now, we will describe and formalize this mathematically using definitions and notation largely based on [2].

Consider a function $\texttt{Enc} : \mathcal{M} \times \mathcal{K} \times \mathcal{R} \to \mathcal{C}$. For any $m \in \mathcal{M}$ and $\texttt{k} \in \mathcal{K}$, one picks a random $r \in \mathcal{R}$ and computes $\texttt{ct} = \texttt{Enc}(m, \texttt{k}, r) \in \mathcal{C}$. We call $\texttt{Enc}$ an encryption function or encryption algorithm, m the message or plaintext, $\texttt{k}$ the key, and $\texttt{ct}$ the ciphertext. When $|\mathcal{R}| = 1$, then we call this deterministic encryption. When $|\mathcal{R}| > 1$, we call this probabilistic encryption, which can provide for higher security. We also consider a function $\texttt{Dec} : \mathcal{C} \times \mathcal{K} \to \mathcal{M}$. For inputs $\texttt{ct} \in \mathcal{C}$, $\texttt{k} \in \mathcal{K}$, and corresponding output $m \in \mathcal{M}$, we write $\texttt{Dec}(\texttt{ct}, \texttt{k}) = m$. We cal $\texttt{Dec}$ a decryption function or decryption algorithm.

Let $\mathcal{E} = (\texttt{Enc}, \texttt{Dec})$. We call $\mathcal{E}$ a Shannon Cipher if for all keys $\texttt{k} \in \mathcal{K}$ and messages $m \in \mathcal{M}$ we have that

$$\texttt{Dec}(\texttt{Enc}(m, \texttt{k}, r), \texttt{k}) = m$$

The above describes the basic concept of *symmetric key cryptography*, which is also often called private key cryptography. If both Alice and Bob have access to the key $\texttt{k}$, Alice can encrypt a message m using $\texttt{Enc}(m, \texttt{k}, r) = \texttt{ct}$, send $\texttt{ct}$ to Bob, and then Bob can decrypt $\texttt{ct}$ via $\texttt{Dec}(\texttt{ct}, \texttt{k}) = m$ and recover m. The assumption behind this is that given $\texttt{ct}$ but not $\texttt{k}$, one can not recover m. The reason this is called symmetric key cryptography is because both Alice and Bob must use the same

key $\mathtt{k}$.

In addition to symmetric key cryptography, we can also define *public key cryptography.* Consider two functions $\mathtt{Enc}$ and $\mathtt{Dec}$ as described above, a message $m \in \mathcal{M}$, and two keys $\mathtt{sk} \in \mathcal{K}_1$ and $\mathtt{pk} \in \mathcal{K}_2$ such that

$$\mathtt{Dec}(\mathtt{Enc}(m, \mathtt{pk}, r), \mathtt{sk}) = m$$

We call $\mathtt{pk}$ the public key and $\mathtt{sk}$ the secret key. In terms of constructing a public-key cryptosystem, we assume that $\mathtt{pk}$ is public knowledge, while $\mathtt{sk}$ is kept secret and is only known by the party who is picked $\mathtt{sk}$. The key feature here is that anybody can encrypt a message using $\mathtt{pk}$, but only the party possessing $\mathtt{sk}$ can decrypt a message.

The goal of homomorphic encryption is to obtain a homomorphism in the message slot in the encryption function. That is, for two messages m_0 and m_1 in $\mathcal{M}$, a public or private key $\mathtt{k} \in \mathcal{K}$, and random $r_0, r_1 \in \mathcal{R}$ we have that

$$\mathtt{Enc}(m_0, \mathtt{k}, r_0) + \mathtt{Enc}(m_1, \mathtt{k}, r_1) = \mathtt{Enc}(m_0 + m_1, \mathtt{k}, r_2)$$

$$\mathtt{Enc}(m_0, \mathtt{k}, r_0)\mathtt{Enc}(m_1, \mathtt{k}, r_1) = \mathtt{Enc}(m_0 m_1, \mathtt{k}, r_3)$$

for some $r_2, r_3 \in \mathcal{R}$. In designing an reasonably efficient and secure cryptosystem, this is a surprisingly difficult task to achieve. The details of how to obtain a viable homomorphic encryption scheme will be presented in chapter 3.

2.2 Lattices

Research in cryptosystems based on lattices has recently gained much popularity, especially in the homomorphic encryption and post-quantum cryptography communities. This section will discuss lattices, using notation and definitions based on [1]. Let K be the field $\mathbb{Q}$ or $\mathbb{R}$, and V a K-vector space with $\dim(V) = n$. Λ is a lattice if Λ is a discrete additive subgroup of V which spans V. Note that by discrete, we mean that Λ is isomorphic to $\mathbb{Z}^n$ as $\mathbb{Z}$-modules. That is, given $B = (v_1, \ldots, v_n)$ a basis for V, we have that

$$\Lambda = \mathbb{Z}v_1 + \mathbb{Z}v_2 + \cdots + \mathbb{Z}v_n$$

Lattices can also be defined similarly to the above for any field K and ring R, but we will exclude this more general definition. Consider a basis $B = (v_1, v_2)$ of $\mathbb{R}^2$. Such a basis may be nice, as in having properties such as near-orthogonality.

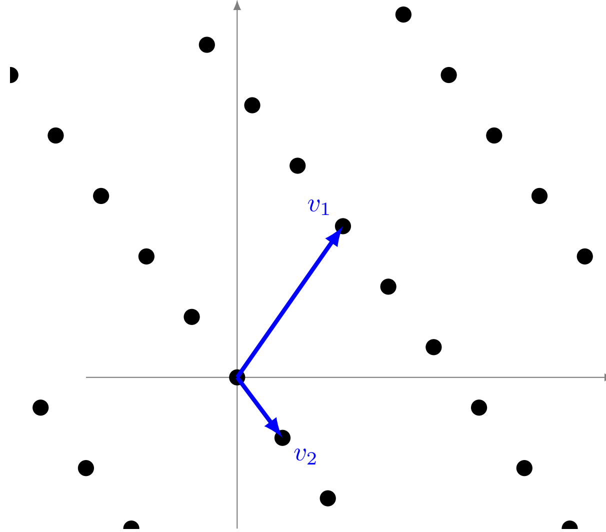

Figure 2.1: A lattice with a good basis

The basis for Λ shown is rather nice, as the basis vectors are short and nearly orthogonal. However, we can easily define a new basis $B' = (v_1', v_2') = (2v_1 + v_2, v_1 + v_1)$ which defines the same lattice. This, unlike the previous basis, does not have near-orthogonality nor short basis vectors.

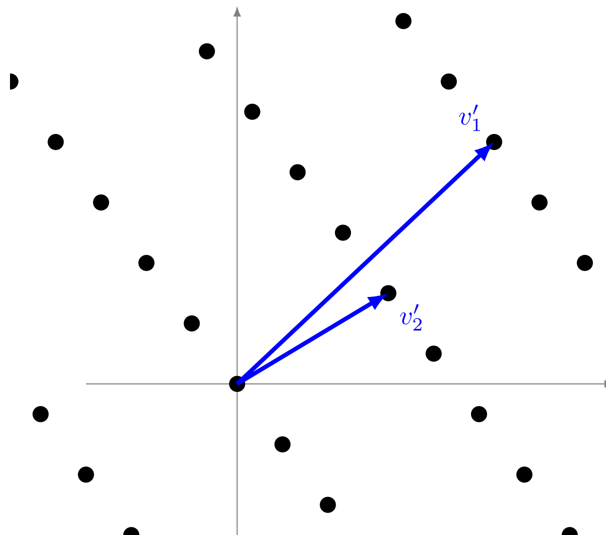

Figure 2.2: A lattice with a bad basis

2.2.1 Hard Lattice Problems

Lattices are an attractive foundation for cryptosystems due to the presumed difficulty of some lattice problems. There are many problems in lattices that are presumed to be difficult in the general case. One such problem is the *shortest vector problem (SVP)*.

Problem 2.2.1 *Given some basis B of a lattice Λ, find the shortest vector in Λ.*

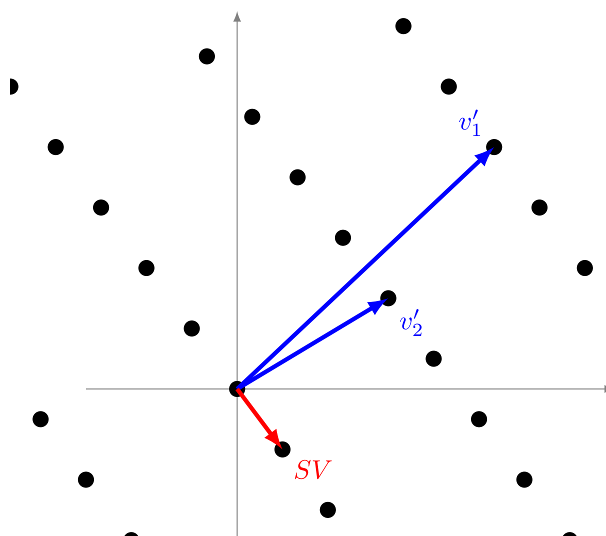

Figure 2.3: Shortest vector problem

An example of the SVP in two dimensions is illustrated in figure 2.3. Given $B = \{v_1', v_2'\}$, we wish to find SV. If the basis B given is a basis with nice properties, this problem becomes easier. In general however, the basis given is not a good basis. A similar problem to the SVP is the *closest vector problem (CVP)*.

Problem 2.2.2 *Given $x \in \mathbb{R}^n$ and a basis B for a lattice Λ, find the closest vector to x in Λ.*

Just as the SVP, solving the CVP is much easier when a good basis is known for Λ. Many cryptosystems rely on these presumably difficult problems to make their protocols secure. This includes the underlying problem to homomomorphic encryption schemes, which was shown to be at least as hard as many worst case lattice problems in [16].

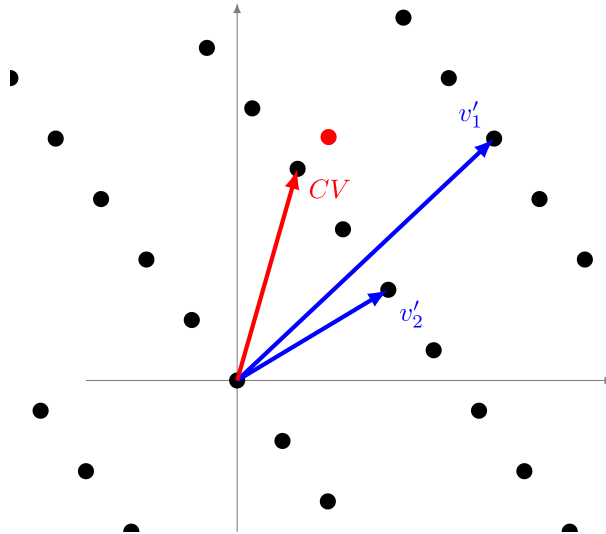

Figure 2.4: Closest vector problem

2.3 Cyclotomic Polynomials and Rings

Integer rings modulo a cyclotomic polynomial are used extensively in homomorphic encryption schemes. We briefly define a cyclotomic polynomial in this section, which we base off [14]. Let $\zeta_m = e^{2\pi i/m}$, which satisfies the equation $x^m - 1 = 0$. Let $\Phi_m(x)$ be the polynomial

$$\Phi_m(x) = \prod_{\gcd(a,m)=1} (x - \zeta_m^a)$$

where $1 \leq a \leq m$. The polynomial $\Phi_m(x)$ is known as the *mth cyclotomic polynomial*. $\Phi_m(x)$ is the unique irreducible polyonomial in $\mathbb{Z}[x]$ which divides $x^m - 1$ but does not divide $x^d - 1$ for any nonnegative integer $d < n$.

In homomorphic encryption, the most popular choice when requiring a cyclotomic polynomial to be used is the mth cyclotomic polynomial where m is a power of two. In this case, the polynomial is given explicitly by $\Phi_m(x) = x^n + 1$ where $m = 2n$. In this thesis, unless otherwise specified, we will always assume m is a power of two. We will omit the m in this case, and just write $\Phi(x)$ to denote a power of two cyclotomic of degree n.

2.4 Notation Overview

Define $\mathbb{Z}_q$ as the ring of centered representatives where $\mathbb{Z}_q := \mathbb{Z} \cap (-q/2, q/2]$. Notice this space is isomorphic to $\mathbb{Z}/q\mathbb{Z}$. When given an integer x, we denote $[x]_q$ as the reduction of x into the interval $\mathbb{Z}_q$ such that q divides $[x]_q - x$. We call $[x]_q$ the centered representative of x. When x is a polynomial, $[x]_q$ means that we apply $[\cdot]_q$ to each coefficient of x. For a vector $v = (v_1, \dots, v_n)$, we let $[v]_q = ([v_1]_q, \dots, [v_n]_q)$. We will use these centered representatives in most computations.

We define R_n as the ring

$$R_n := \mathbb{Z}[x]/(\Phi(x))$$

where $\Phi(x)$ is an m-th cyclotomic polynomial of degree n, a power of two. Namely, $\Phi(x) = x^n + 1$. Similar to R_n, we define $R_{n,q}$ as the ring

$$R_{n,q} := \mathbb{Z}_q[x]/(\Phi(x))$$

For any $c \in \mathbb{R}$, the *infinity norm of* c is defined as $\|c\|_\infty = |c|$. For any polynomial $f(x) = \sum_{i=0}^{k} a_i x^i$ with $a_i \in \mathbb{R}$, the *infinity norm of* $f(x)$ is defined as

$$\|f(x)\|_\infty = \max\{|a_0|, \dots, |a_k|\}$$

therefore using centered representatives, for any $f(x) \in R_{n,q}$ we have $\|f(x)\|_\infty \leq q/2$. We denote $\|f(x)\|_1$ and $\|f(x)\|_2$ norm as the standard 1 and 2 norms respectively. $\|f(x)\|_\infty^{\mathsf{can}}$ will denote the less standard canonical embedding norm, which we discuss fully in chapter 3. If a norm is not specified and is written as $\|f(x)\|$, we will assume it is the infinity norm. We also denote $\mathrm{wt}(f(x))$ as the Hamming weight of $f(x)$. The Hamming weight is the number of nonzero coefficients of $f(x)$, or more formally,

$$\mathrm{wt}(f(x)) = |\{i : a_i \neq 0\}|$$

The Hamming weight can be defined analogously for vectors. The symbols $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ will denote floor and ceiling respectively, whereas $\lfloor \cdot \rceil$ will denote rounding to the nearest integer. When applying this to a polynomial or vector, we mean the rounding of each coefficient.

For a set S and a given probability distribution χ, we let $\chi(S)$ denote that distribution on S. If not specified, we will take χ to be the discrete Gaussian distribution as in [10]. Over $\mathbb{Z}$,

the discrete Gaussian distribution $D_{\mathbb{Z},\sigma}$ assigns a probability proportional to $\exp(-\pi x^2/\sigma^2)$ for each $x \in \mathbb{Z}$. For our cylotomic polynomial $\Phi(x) = x^n + 1$, then $\chi(R_n)$ or $D_{\mathbb{Z},\sigma}(R_n)$ can be simply seen as sampling each coefficient from $D_{\mathbb{Z},\sigma}$. In addition to the discrete Gaussian distribution, we will denote $U(S)$ as a uniform distribution on S.

Chapter 3

Basic Homomorphic Encryption

This chapter will introduce the basic algorithms in homomorphic encryption. By basic algorithms, we mean algorithms necessary to perform encryption, decryption, addition, and multiplication. We will not discuss topics in increasing efficiency in this chapter, as it will be covered extensively in chapters 4 and 5. This chapter will only be concerned with outlining the underlying problem in homomorphic encryption and discussing algorithms needed for performing the basic operations.

3.1 Learning With Errors

Let $\mathbf{s} \in \mathbb{Z}_q^n$ be a secret. We sample an error $e \leftarrow \chi(\mathbb{Z})$ from some desired distribution χ such that $\|e\|_\infty \leq \rho$, where ρ is a desired parameter. Then, we sample a uniform random $\mathbf{a} \in \mathbb{Z}_q^n$ and calculate b via

$$b = \langle \mathbf{a}, \mathbf{s} \rangle + e \mod q$$

The ordered pair $(\mathbf{a}, b) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ is called an *LWE sample*. Given as many LWE samples as desired, the search version of the *Learning With Errors* (Search-LWE) problem is to find $\mathbf{s}$. The *Decision-LWE* problem is given m independent samples $(a_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ where (a_i, b_i) is either an LWE sample or drawn from a uniform random distribution, determine from where (a_i, b_i) was drawn [4]. The LWE problem is assumed to be hard, meaning that the best known algorithms are exponential in relation to the dimension of a lattice [18].

When drawing elements from distributions on R_n and $R_{n,q}$, we can similarly define the *Ring Learning With Errors* (RLWE) problem. Choose $s \in R_{n,q}$ and sample an error $e \leftarrow \chi(R_n)$ such that $\|e\|_\infty \leq \rho$. Sample $a \leftarrow U(R_{n,q})$ and compute $b \in R_{n,q}$ via

$$b \equiv as + e \mod (\Phi(x), q)$$

The ordered pair $(a, b) \in R_{n,q}^2$ is called an *RLWE sample*. The Decision Ring Learning With Errors problems can be defined in an analogous way to the LWE problem.

Nearly all homomorphic encryption schemes use the RLWE problem as opposed to the LWE problem, as it will allow us to encryption and compute on much longer messages encoded in polynomials rather than just one bit at a time. The ciphertexts of homomorphic encryption schemes will all essentially take the form of and RLWE sample.

3.2 Homomorphic Encryption Schemes

Variants of the LWE and RLWE problems have been proposed for use in homomorphic encryption. The main variants found are the BFV [10], BGV [3], and CKKS [6] systems, named after their authors Brakerski-Fan-Vercauteren, Brakerski-Gentry-Vaikuntanathan, and Cheon-Kim-Kim-Song respectively. A number of open-source cross platform libraries provide free to use implementations of homormohpic encryption schemes based on RLWE problems. The libraries include SEAL developed by Microsoft, HElib developed by IBM, and PALISADE by various developers. While the implemented schemes vary between homomorphic encryption libraries, most support schemes for both exact integer computation (BFV or BGV) and approximate real number computation (CKKS). Residue number system (RNS) implementation is also available on some libraries, which takes advantage of breaking down moduli with the Chinese Remainder Theorem (CRT) for significant speedup in algorithms.

In this section, we will introduce the BFV, BGV, and CKKS schemes, along with the basic encryption, decryption, addition, and multiplication operations for each. RNS variants, as well as other efficient computing techniques, will be discussed in chapter 4.

3.3 BFV

Our first scheme will be the BFV scheme originally presented in [10]. The format of a BFV ciphertext generally makes for a more difficult analysis of error, which we will discuss in chapter 5. By error, we mean an error polynomial introduced to the system such as we did in the RLWE problem.

3.3.1 BFV Key Generation, Encryption, and Decryption

Choose a secret $s \in R_{n,3}$. In other words, choose $s \in R_{n,q}$ with coefficients in $\{-1, 0, 1\}$. The secret key is $\texttt{sk} = s \in R_{n,3}$. Often times in homomorphic encryption papers, the secret key will be an ordered pair consisting of s and 1 to allow some equations to be written with as an inner product. For our purposes however, we will only need the secret key as s. Sample a uniform random $a \leftarrow U(R_{n,q})$ and a random error polynomial $e \leftarrow \chi(R_n)$ such that $\|e\|_\infty \leq \rho$. In BFV, we define the public key $\texttt{pk} = (a, b) \in R_{n,q}^2$ where b is computed by

$$b \equiv -(as + e) \mod (\Phi(x), q)$$

It is easy to see that $\texttt{pk}$ is simply an RLWE sample. Using this public key, we can now define the public key encryption algorithm. We sample $e_0', e_0'' \leftarrow \chi(R_n)$ such that $\left\|e_0'\right\|_\infty, \left\|e_0''\right\|_\infty \leq \rho$, and let $D \in \mathbb{Z}$ be large. The user generates a random polynomial $u \in R_{n,q}$ with coefficients in $\{-1, 0, 1\}$ of a specified weight, which they keep secret. For a message $m_0 \in R_{n,t}$, we compute $(a_0, b_0) \in R_{n,q}^2$ as the ciphertext for m_0 where

$$a_0 \equiv au + e_0' \mod (\Phi(x), q)$$
$$b_0 \equiv bu + Dm_0 + e_0'' \mod (\Phi(x), q)$$

The encryption algorithm for BFV is given formally in algorithm 3.1.

After encryption, we obtain the ciphertext $(a_0, b_0) \in R_{n,q}^2$. We can group together the error terms so that

$$b_0 + a_0 s \equiv Dm_0 + e_0 \mod (\Phi(x), q) \tag{3.1}$$

	$\texttt{BFV.Encrypt}(m_0, \texttt{pk})$
Input:	$m_0 \in R_{n,t}$ message,
	$\texttt{pk} = (a, b) \in R_{n,q}^2$ public key,
Output:	$\texttt{ct}_0 = (a_0, b_0) \in R_{n,q}^2$ BFV ciphertext.
Step 1.	Generate a random $u \in R_{n,3}$.
Step 2.	Sample $e_0', e_0'' \leftarrow \chi(R_n)$ such that $\left\|e_0'\right\|_\infty, \left\|e_0''\right\|_\infty \leq \rho$.
Step 3.	Compute $a_0 \in R_{n,q}$ with $a_0 \equiv au + e_0' \mod (\Phi(x), q)$ and $b_0 \in R_{n,q}$ with $b_0 \equiv bu + Dm_0 + e_0'' \mod (\Phi(x), q)$.
Step 4.	Return $\texttt{ct}_0 = (a_0, b_0)$.

Algorithm 3.1: BFV Encryption Algorithm

for $e_0 = eu + e_0' - se_0''$. It is easy to see that $\|e_0\|_\infty \leq \|eu\|_\infty + \left\|e_0'\right\|_\infty + \left\|se_0''\right\|_\infty \leq 2\rho\delta_R + \rho$, where δ_R is the expansion factor of R_n given by

$$\delta_R = \max\left\{\frac{\|x \cdot y\|_\infty}{\|x\|_\infty \cdot \|y\|_\infty} : x, y \in R_n\right\}$$

The error bound on a freshly encrypted ciphertext is given more formally via the following lemma.

Lemma 3.3.1 *Let $\texttt{ct}_0 = (a_0, b_0)$ be the ouput of algorithm 3.1, where $b_0 + a_0 s \equiv Dm_0 + e_0$. Then, $\|e_0\|_\infty \leq 2\rho\delta_R + \rho$.*

Proof. Since $e_0 = eu + e_0' - se_0''$, we have that

$$\begin{aligned}
\|e_0\|_\infty &= \left\|eu + e_0' - se_0''\right\|_\infty \\
&\leq \|eu\|_\infty + \left\|e_0'\right\|_\infty + \left\|se_0''\right\|_\infty \\
&\leq \delta_R \|e\|_\infty \|u\|_\infty + \left\|e_0'\right\|_\infty + \delta_R \|s\|_\infty \left\|e_0''\right\|_\infty \\
&\leq 2\delta_R\rho + \rho
\end{aligned}$$

as desired. □

When referring to a BFV ciphertext, we will always consider it in the format as in equation 3.1 with a single error term. We do provide a basic bound for this encryption error term, but a much more thorough analysis of all errors will be presented in chapter 5. Given a BFV ciphertext, we can decrypt it using the following algorithm.

	$\texttt{BFV.Decrypt}(\texttt{ct}_0, \texttt{sk})$
Input:	$\texttt{ct}_0 = (a_0, b_0) \in R_{n,q}^2$ BFV ciphertext,
	$\texttt{sk} = s \in R_{n,3}$ secret key.
Output:	$m_0 \in R_{n,t}$ message.
Step 1.	Compute $m_0 = \lfloor \frac{b_0 + a_0 s \mod (\Phi(x), q)}{D} \rceil$.
Step 2.	Return m_0.

Algorithm 3.2: BFV Decryption Algorithm

In step 1 of algorithm 3.2, note that the $\mod (\Phi(x), q)$ in the numerator means we reduce $b_0 + a_0 s$ modulo $(\Phi(x), q)$ before dividing by D and rounding. It is important to note that the original message m_0 is only recovered by the decryption algorithm if certain conditions are met on the error bound of e_0. More precisely, we require that $\|e_0\|_\infty < D/2$. Notice,

$$\left\lfloor \frac{b_0 + a_0 s \mod (\Phi(x), q)}{D} \right\rceil = \left\lfloor \frac{e_0}{D} + m_0 \right\rceil = \left\lfloor \frac{e_0}{D} \right\rceil + m_0$$

The final equality above follows from the fact that m_0 is not a fraction. As $\left\|e_0/D\right\|_\infty < 1/2$, e_0/D will round downwards to 0 when applying the above rounding, and we will retrieve the original message m_0 from the decryption algorithm. We therefore require that $\|e_0\|_\infty < D/2$ for a guarantee of correctness in decryption. D is chosen to be large to guarantee this bound, and often with nice properties. For instance, one might choose D such that $D \approx q/t$. In classic BFV, D is chosen so that $D = \lfloor \frac{q}{t} \rfloor$. In the case of classic BFV, decryption can also be given by computing

$$m_0 = \left\lfloor \frac{t(b_0 + a_0 s \mod (\Phi(x), q))}{q} \right\rceil \mod t$$

which will also result in a correct decryption if $\|e_0\|_\infty < D/2$. The special case we use is letting $D = (q-1)/t$, where $t | q-1$. It is worth noting that with these decryptions, we are assuming that $Dm_0 + e_0 \in R_{n,q}$. That is, the coefficients of $Dm_0 + e_0$ are already fully reduced modulo q. If this

is not the case, we can decrypt similarly by computing

$$\begin{aligned}\left\lfloor \frac{t(b_0 + a_0 s) \mod \Phi(x)}{q} \right\rceil \mod t &\equiv \left\lfloor \frac{t(Dm_0 + e_0 + qr)}{q} \right\rceil \mod t \\ &\equiv \left\lfloor \frac{t((\frac{q-1}{t})m_0 + e_0 + qr)}{q} \right\rceil \mod t \\ &\equiv m_0 + \left\lfloor \frac{t}{q}(e_0 - \frac{1}{t}m_0) \right\rceil + tr \mod t \\ &\equiv m_0 \mod t\end{aligned}$$

so long as $\left\| \left\lfloor \frac{t}{q}(e_0 - \frac{1}{t}m_0) \right\rceil \right\|_\infty = 0$.

3.3.2 BFV Addition

Although encryption and decryption is standard in any cryptosystem, the key feature of homomorphic encryption schemes is the ability to apply operations directly on ciphertexts. Addition and multiplication are the standard operations which are covered in this chapter. More advanced non-arithmetic operations will be touched on later in this text in chapter 4. With arithmetic operations, the main concern is error growth. As we will see, arithmetic operations increase the size of error terms in a ciphertext, thus creating potential for incorrect decryption of a message if the necessary decryption bound is not met. A full analysis of error bounds can be found in chapter 5. In this section, we will focus our discussion on the actual operations, and will only state the resulting error terms or bounds.

After encryption, a BFV ciphertext is given as an ordered pair $\mathtt{ct}_i = (a_i, b_i)$ where a_i is chosen uniform randomly from $R_{n,q}$ and $b_i + a_i s \equiv Dm_i + e_i \mod (\Phi(x), q) \in R_{n,q}$ for some message $m_i \in R_{n,t}$. Addition between two BFV ciphertexts can be computed via algorithm 3.3.

	$\mathtt{BFV.Add}(\mathtt{ct}_0, \mathtt{ct}_1)$
Input:	$\mathtt{ct}_0 = (a_0, b_0), \mathtt{ct}_1 = (a_1, b_1) \in R_{n,q}^2$ BFV ciphertexts.
Output:	$\mathtt{ct}_2 \in R_{n,q}^2$.
Step 1.	Compute $\mathtt{ct}_2 = \mathtt{ct}_0 + \mathtt{ct}_1 \mod q$.
Step 2.	Return $\mathtt{ct}_2$.

Algorithm 3.3: BFV Addition

The following lemma shows that algorithm 3.3 is correct, as well as what exact error $\mathtt{ct}_2$

contains.

Lemma 3.3.2 *Let* $\mathtt{ct}_2 = \mathtt{BFV.Add}(\mathtt{ct}_0, \mathtt{ct}_1)$ *be the output of algorithm 3.3 where* $\|e_0\|_\infty, \|e_1\|_\infty \leq E$*. Then,* $\mathtt{ct}_2$ *is a BFV ciphertext with error term* e_{add} *bounded by* $\|e_{\text{add}}\|_\infty \leq 2E + t$*.*

Proof. Let $r \in R_{n,q}$ such that $m_0 + m_1 = [m_0 + m_1]_t + tr$ and $\epsilon = q/t - D$. We have that

$$\begin{aligned}(b_0 + a_0 s) + (b_1 + a_1 s) &\equiv D(m_0 + m_1) + e_0 + e_1 \mod (\Phi(x), q) \\ &\equiv D[m_0 + m_1]_t + e_0 + e_1 + Dtr \mod (\Phi(x), q) \\ &\equiv D[m_0 + m_1]_t + e_0 + e_1 - \epsilon tr \mod (\Phi(x), q) \\ &\equiv D[m_0 + m_1]_t + e_{\text{add}} \mod (\Phi(x), q)\end{aligned}$$

where $e_{\text{add}} = e_0 + e_1 - \epsilon tr$, which is a BFV ciphertext with error term e_{add}. Furthermore, as $\|\epsilon\|_\infty \leq 1$, we have that $\|e_{\text{add}}\|_\infty \leq 2E + t$, as desired. □

The third equivalence follows from the fact that since $\epsilon t = q - Dt$, we have $Dt \equiv -\epsilon t \mod q$. We also note that since addition between polynomials in $R_{n,q}$ remains as a polynomial of degree less than n, reduction modulo $\Phi(x)$ is not necessary for addition in the actual ciphertext computation. The error bound immediately generalizes to any number of additions. A full analysis of error bounds will be presented in chapter 5.

3.3.3 BFV Multiplication

Although addition between two ciphertexts is fairly easy, multiplication is more complicated. First, we define algorithm 3.4 for the first step of BFV multiplication which we call the initial multiplication.

We call this the initial multiplication due to the fact that although it technically multiplies two BFV ciphertexts together and preserves the message, we also obtain a degree 2 polynomial in the secret key rather than a degree 1. Roughly speaking, we will obtain a ciphertext of the form $c'_0 + c'_1 s + c'_2 s^2 \equiv Dm_0+$error, rather than $b_0 + a_0 s \equiv Dm_0+$error. We will call this a degree 2 BFV ciphertext, or a degree n ciphertext more generally if we can write a BFV ciphertext as a degree n polynomial under s. To verify this algorithm, we will first introduce the following lemma

	$\texttt{BFV.Int.Mult}(\texttt{ct}_0, \texttt{ct}_1)$
Input:	$\texttt{ct}_0 = (a_0, b_0), \texttt{ct}_1 = (a_1, b_1) \in R^2_{n,q}$ BFV ciphertexts.
Output:	$(c'_0, c'_1, c'_2) \in R^3_{n,q}$.
Step 1.	Compute $c_0 = b_0 b_1 \mod \Phi(x), c_1 = b_1 a_0 + b_0 a_1 \mod \Phi(x)$, and $c_2 = a_0 a_1 \mod \Phi(x)$.
Step 2.	Compute $c'_0 = \lfloor t c_0 / q \rceil, c'_1 = \lfloor t c_1 / q \rceil$, and $c'_2 = \lfloor t c_2 / q \rceil$.
Step 3.	Return (c'_0, c'_1, c'_2).

Algorithm 3.4: BFV Initial Multiplication

and subsequent proof in order to show that this multiplication of BFV ciphertexts returns a BFV ciphertext of a higher degree.

Lemma 3.3.3 *Let* $(c'_0, c'_1, c'_2) = \texttt{BFV.Int.Mult}(\texttt{ct}_0, \texttt{ct}_1) \in R^3_{n,q}$ *be the output of algorithm 3.4. Then,* (c'_0, c'_1, c'_2) *is a degree 2 ciphertext under* s *with error term* e^* *bounded by* $\|e^*\|_\infty \leq 2\delta_R t E(1 + \delta_R \|s\|_\infty) + 2\delta_R^2 t^2 (\|s\|_\infty + 1)^2$.

Proof. Suppose we are given two BFV ciphertexts $\texttt{ct}_0 = (a_0, b_0)$ and $\texttt{ct}_1 = (a_1, b_1)$. Recall that

$$b_0 + a_0 s \equiv D m_0 + e_0 \mod (\Phi(x), q)$$
$$b_1 + a_1 s \equiv D m_1 + e_1 \mod (\Phi(x), q)$$

We can then write

$$b_0 + a_0 s \equiv D m_0 + e_0 + q r_0 \mod \Phi(x)$$
$$b_1 + a_1 s \equiv D m_1 + e_1 + q r_1 \mod \Phi(x)$$

for some $r_0, r_1 \in R_n$. Then,

$$(b_0 + a_0 s)(b_1 + a_1 s) \equiv (Dm_0 + e_0 + qr_0)\,(Dm_1 + e_1 + qr_1) \mod \Phi(x)$$

$$\equiv D^2 m_0 m_1 + D(m_0 e_1 + m_1 e_0) + q(e_0 r_1 + r_0 e_1)$$
$$+ e_0 e_1 + qD(m_0 r_1 + r_0 m_1) + q^2 r_0 r_1 \mod \Phi(x)$$

$$\equiv c_0 + c_1 s + c_2 s^2 \mod \Phi(x)$$

as $(b_0 + a_0 s)(b_1 + a_1 s) \equiv b_0 b_1 + (b_0 a_1 + b_1 a_0)s + a_0 a_1 s^2$. Let $r_t(q) = q - tD$. Since $D = \lfloor q/t \rfloor$, we can write $D = q/t - \epsilon$ for some ϵ, or $tD = q - r_t(q)$. Now, write

$$m_0 m_1 \equiv [m_0 m_1]_t + t r_m \mod \Phi(x)$$
$$e_0 e_1 \equiv [e_0 e_1]_D + D r_e \mod \Phi(x)$$

for some $r_e \in R_n$ and $r_m \in R_n$. Then, we can rewrite our earlier equation

$$c_0 + c_1 s + c_2 s^2 \equiv D^2 m_0 m_1 + D(m_0 e_1 + m_1 e_0) + q(e_0 r_1 + r_0 e_1)$$
$$+ e_0 e_1 + qD(m_0 r_1 + r_0 m_1) + q^2 r_0 r_1 \mod \Phi(x)$$

$$\equiv D^2 [m_0 m_1]_t + D^2 t r_m + D(m_0 e_1 + m_1 e_0) + q(e_0 r_1 + r_0 e_1)$$
$$+ [e_0 e_1]_D + D r_e + qD(m_0 r_1 + r_0 m_1) + q^2 r_0 r_1 \mod \Phi(x)$$

Now, we scale by t/q

$$
\begin{aligned}
(t/q)(c_0 + c_1 s + c_2 s^2) \equiv & \; (tD^2/q)[m_0 m_1]_t + (tD^2/q)2tr_m + (tD/q)(m_0 e_1 + m_1 e_0) \\
& + (tq/q)(e_0 r_1 + r_0 e_1) + (t/q)[e_0 e_1]_D + (tD/q) r_e \\
& + (tqD/q)(m_0 r_1 + r_0 m_1) + (tq^2/q) r_0 r_1 \mod \Phi(x)
\end{aligned}
$$

$$
\begin{aligned}
& \equiv ((q - r_t(q))D/q)[m_0 m_1]_t + ((q - r_t(q))D/q)2tr_m \\
& + ((q - r_t(q))/q)(m_0 e_1 + m_1 e_0) + (t)(e_0 r_1 + r_0 e_1) \\
& + (t/q)[e_0 e_1]_D + ((q - r_t(q))/q) r_e + (q - r_t(q))(m_0 r_1 + r_0 m_1) \\
& + (tq^2/q) r_0 r_1 \mod \Phi(x)
\end{aligned}
$$

which, after simplification, gives

$$
\begin{aligned}
(t/q)(c_0 + c_1 s + c_2 s^2) \equiv & \; D[m_0 m_1]_t + (m_0 e_1 + m_1 e_0) + t(e_0 r_1 + r_0 e_1) + r_e \\
& + (q - r_t(q))(r_m + m_0 r_1 + r_0 m_1) + qtr_0 r_1 + (t/q)[e_0 e_1]_D \\
& - (r_t(q)/q)(Dm_0 m_1 + (m_0 e_1 + m_1 e_0) + r_e) \mod \Phi(x)
\end{aligned}
$$

Our goal is to reduce this equation modulo q. However, we can not yet guarantee that these terms form a polynomial with integer coefficients as opposed to fractional coefficients. Therefore, it does not make sense to reduce modulo q. Now, on the other hand we have

$$
(t/q)(c_0 + c_1 s + c_2 s^2) \equiv c'_0 + c'_1 s + c'_2 s^2 + r_a \mod \Phi(x)
$$

for $r_a = (t/q)(b_0 + a_0 s)(b_1 + a_1 s) - c'_0 - c'_1 s - c'_2 s^2$. In this case, r_a can be considered a rounding

error accumulated. Then, we have that

$$c_0' + c_1's + c_2's^2 \equiv D[m_0m_1]_t + (m_0e_1 + m_1e_0) + t(e_0r_1 + r_0e_1) + r_e$$
$$+ (q - r_t(q))(r_m + m_0r_1 + r_0m_1) + qtr_0r_1 + (t/q)[e_0e_1]_D$$
$$- (r_t(q)/q)(Dm_0m_1 + (m_0e_1 + m_1e_0) + r_e) - r_a \mod \Phi(x)$$

$$\equiv D[m_0m_1]_t + (m_0e_1 + m_1e_0) + t(e_0r_1 + r_0e_1) + r_e$$
$$+ (q - r_t(q))(r_m + m_0r_1 + r_0m_1) + qtr_0r_1 + r_r - r_a \mod \Phi(x)$$

where $r_r = (t/q)[e_0e_1]_D - (r_t(q)/q)(Dm_0m_1 + (m_0e_1 + m_1e_0) + r_e)$. It is easy to see that $\lfloor tc_0/q \rceil + \lfloor tc_1/q \rceil s + \lfloor tc_2/q \rceil s^2$ has coefficients in $\mathbb{Z}$. Also, $D[m_0m_1]_t + (m_0e_1 + m_1e_0) + t(e_0r_1 + r_0e_1) + r_e + (q - r_t(q))(r_m + m_0r_1 + r_0m_1) + qtr_0r_1$ has coefficients in $\mathbb{Z}$ too. Therefore, we know for a fact that $r_r - r_a$ must have integer coefficients as well. For this reason, we can now reduce the above modulo q. We then have

$$c_0' + c_1's + c_2's^2 \equiv D[m_0m_1]_t + (m_0e_1 + m_1e_0) + t(e_0r_1 + r_0e_1) + r_e$$
$$+ (-r_t(q))(r_m + m_0r_1 + r_0m_1) + r_r - r_a \mod (\Phi(x), q)$$

Letting $e^* = (m_0e_1 + m_1e_0) + t(e_0r_1 + r_0e_1) + r_e + (-r_t(q))(r_m + m_0r_1 + r_0m_1) + r_r - r_a$ be the error above, we can write this slightly neater as

$$c_0' + c_1's + c_2's^2 \equiv D[m_0m_1]_t + e^* \mod (\Phi(x), q)$$

which is a degree 2 BFV ciphertext with error term e^*. The final error bound can deduced from this error term. □

The deduced bound to the norm of this error term is important in discussing the practicality of the system. That is, we want to see how multiplication of two ciphertexts will affect the error term. The upper bound to $\|e^*\|_\infty$ takes a decent amount of work, so we will return to this bound in chapter 5 and exclude the proof here for now.

3.4 Computing $c_2's^2$ without knowing s^2

From the proof of lemma 3.3.3, we can see that we do indeed have a BFV ciphertext. However, the issue now is that we no longer have an encryption under only s, but both s and s^2. In order to get our BFV ciphertext back to a form that is only under s, we must relinearize the term which includes s^2. That is, given a polynomial $c_2' \in R_{n,q}$, we want to compute polynomials $d_0, d_1 \in R_{n,q}$ such that $d_0 + d_1 s$ is approximately $c_2's^2$ modulo $(\Phi(x), q)$. Or,

$$d_0 + d_1 s \equiv c_2's^2 + e'' \mod (\Phi(x), q) \tag{3.2}$$

for some small error $e'' \in R_n$. The significance of this equation is that we want to find these polynomials without knowing either of s or s^2. In this next portion, we will present two methods of relinearizing this $c_2's^2$ term in order to obtain the desired d_0 and d_1 polynomials.

3.4.1 Relinearization Version 1

Our first relinearization strategy is to flatten c_2' with respect to a base B. Note that B is chosen by the user. For any $B \in \mathbb{Z}^+$, with the smallest $\gamma \in \mathbb{Z}$ such that $B^\gamma > q$, we write

$$c_2' = \sum_{j=0}^{\gamma-1} h_j B^j$$

where $h_j \in R_{n,q}$ such that $\|h_j\|_\infty \leq B/2$. Let $\mathbf{h}$ be the $1 \times \gamma$ vector

$$\mathbf{h} = \begin{pmatrix} h_0, h_1, \ldots, h_{\gamma-1} \end{pmatrix}$$

and $\mathbf{g}$ be the $\gamma \times 1$ vector

$$\mathbf{g} = \begin{pmatrix} 1, B, B^2, \ldots, B^{\gamma-1} \end{pmatrix}^T$$

Then, $c_2's^2 = (\mathbf{hg})s^2 = (\sum_{j=0}^{\gamma-1} h_j B^j)s^2$. Let $\mathbf{u}$ a $\gamma \times 1$ vector where each entry is drawn uniform randomly from $R_{n,q}$. Sample a random $\gamma \times 1$ vector $\mathbf{w}$, where each entry of $\mathbf{w}$ is in R_n and bounded by ρ. Let $\mathbf{v}$ be the $\gamma \times 1$ vector

$$\mathbf{v} = s^2\mathbf{g} - \mathbf{u}s + \mathbf{w}$$

where each entry is computed modulo $(\Phi(x), q)$. We call the vector pair $(\mathbf{u}, \mathbf{v})$ the *evaluation key*, which we denote $\texttt{evk}_{\text{flat}} = (\mathbf{u}, \mathbf{v})$. The process described above can be written more formally with the following key generation algorithm.

	$\texttt{Evk.Flat.Gen}(B, \texttt{sk})$
Input:	$B \in \mathbb{Z}^+$ a base,
	$\texttt{sk} = s \in R_{n,3}$ secret key.
Output:	$\texttt{evk}_{\text{flat}} = (\mathbf{u}, \mathbf{v}) \in R_{n,q}^{\gamma\times 1} \times R_{n,q}^{\gamma\times 1}$ evaluation key.
Step 1.	Compute $\gamma = \lceil \log_B(q) \rceil$ and $\mathbf{g}^T = \left(1, B, B^2, \ldots, B^{\gamma-1}\right)$.
Step 2.	Create $\mathbf{u} \in R_{n,q}^{\gamma\times 1}$ such that each entry is sampled uniform randomly from $R_{n,q}$ and a random $\mathbf{w} \in R_n^{\gamma\times 1}$ such that each entry is in R_n and bounded by ρ.
Step 3.	Compute $\mathbf{v} = s^2\mathbf{g} - \mathbf{u}s + \mathbf{w}$ where each entry is computed mod $(\Phi(x), q)$.
Step 4.	Return $\texttt{evk}_{\text{flat}} = (\mathbf{u}, \mathbf{v})$.

Algorithm 3.5: Flattened Evaluation Key Generation

We use the word *flat* as a subscript in this evaluation key to clearly acknowledge that this relinearization key is to be used in a flattening process. In the next subsection, we will discuss an alternate relinearization technique which requires a different evaluation key. Notice that for the jth coordinate in the vector pair $(\mathbf{u}, \mathbf{v})$, we have the relationship

$$v_j + u_j s = s^2 B^j + w_j \mod (\Phi(x), q)$$

As the equation above is a simple RLWE encryption of $s^2 B^j$, we can publish the evaluation key without compromising security of the secret key s. With this publicly known evaluation key, we can finally define our relinearization process with the following lemma.

Lemma 3.4.1 *Let $c_2' \in R_{n,q}$ be any polynomial and $\texttt{evk}_{\text{flat}}$ the output of algorithm 3.5. Then,*

$$c_2' s^2 + \mathbf{hw} \equiv d_0 + d_1 s \mod (\Phi(x), q)$$

where $d_0 = \mathbf{hv}$ and $d_1 = \mathbf{hu}$.

Proof. By our key generation process, we have computed $\mathbf{h}$ and $\mathbf{g}$ so that $c_2' = (\mathbf{hg})$ with $\left\|h_j\right\|_\infty \leq$

$B/2$. Notice,

$$\begin{aligned} c_2' s^2 &= (\mathbf{hg}) s^2 \\ &= \left(\sum_{j=0}^{\gamma-1} h_j B^j \right) s^2 \\ &\equiv \sum_{j=0}^{\gamma-1} h_j (v_j + u_j s - w_j) \mod (\Phi(x), q) \end{aligned}$$

Then, we have

$$\begin{aligned} c_2' s^2 + \sum_{j=0}^{\gamma-1} h_j w_j &\equiv \sum_{j=0}^{\gamma-1} h_j v_j + \left(\sum_{j=0}^{\gamma-1} h_j u_j \right) s \mod (\Phi(x), q) \\ &\equiv \mathbf{hv} + (\mathbf{hu}) s \mod (\Phi(x), q) \end{aligned}$$

Or equivalently, $c_2' s^2 + \mathbf{hw} \equiv d_0 + d_1 s \mod (\Phi(x), q)$. □

From this lemma, we can see that we now have our desired relationship from equation 3.3. This allows us to define our relinearization algorithm.

	$\texttt{Relin.V1}((c_0', c_1', c_2'), \texttt{evk}_{\text{flat}})$
Input:	$(c_0', c_1', c_2') \in R_{n,q}^3$ polynomials,
	$\texttt{evk}_{\text{flat}}$ evaluation key.
Output:	$(c_0' + d_0, c_1' + d_1) \in R_{n,q}^2$.
Step 1.	Compute $\mathbf{h} = (h_0, \ldots, h_{\gamma-1})$ such that $c_2' = \sum_{j=0}^{\gamma-1} h_j B^j$.
Step 2.	Compute $d_0 = \mathbf{hv}$ and $d_1 = \mathbf{hu}$.
Step 3.	Return $(c_0' + d_0, c_1' + d_1)$.

Algorithm 3.6: Relinearization Version 1

Using this algorithm, we can now clearly reduce a degree 2 BFV ciphertext to a degree 1 BFV ciphertext using the following lemma.

Lemma 3.4.2 *Let $c_0', c_1', c_2' \in R_{n,q}$ such that $c_0' + c_1' s + c_2' s^2 \equiv D[m_0 m_1]_t + e^* \mod (\Phi(x), q)$ for $m_0, m_1 \in R_{n,t}$ and $e^* \in R_n$. Then, $\texttt{Relin.V1}((c_0', c_1', c_2'), \texttt{evk}_{\text{flat}})$ returns a degree 1 BFV ciphertext with error term e_{mult} bounded by $\|e_{\text{mult}}\|_\infty \leq \|e^*\|_\infty + \gamma \delta_R \rho B/2$.*

Proof. By lemma 3.4.1, we have that $c_2's^2+\mathbf{hw} \equiv d_0+d_1 s \mod (\Phi(x), q)$. Therefore, if $(c_0'+d_0, c_1'+d_1)$ is the output of algorithm 3.6, we have

$$\begin{aligned}(c_0'+d_0)+(c_1'+d_1)s &\equiv c_0'+c_1's+c_2's^2+\mathbf{hw} \mod (\Phi(x),q)\\ &\equiv D[m_0m_1]_t+e^*+\mathbf{hw} \mod (\Phi(x),q)\\ &\equiv D[m_0m_1]_t+e_{\text{mult}} \mod (\Phi(x),q)\end{aligned}$$

where $e_{\text{mult}} = e^* - \mathbf{hw}$. As $\|\mathbf{h}\|_\infty \leq B/2$ and $\|\mathbf{w}\|_\infty \leq \rho$, we have $\|\mathbf{hw}\|_\infty \leq \gamma\delta_R\rho B/2$. Then, $\|e_{\text{mult}}\|_\infty \leq \|e^*\|_\infty + \gamma\delta_R\rho B/2$, as desired. □

Note that when e^* is obtained by lemma 3.3.3, we obtain the error bound $\|e_{\text{mult}}\|_\infty \leq 2\delta_R tE(1+\delta_R\|s\|_\infty)+2\delta_R^2t^2(\|s\|_\infty+1)^2+\gamma\delta_R\rho B/2$.

3.4.2 Relinearization Version 2

We now present an alternative to the first relinearization process. As before, given a polynomial $c_2' \in R_{n,q}$, we want to compute polynomials $d_0', d_1' \in R_{n,q}$ such that $d_0'+d_1's$ is approximately $c_2's^2$ modulo $(\Phi(x), q)$. Or,

$$d_0'+d_1's \equiv c_2's^2+e'' \mod (\Phi(x),q) \tag{3.3}$$

for some small error $e'' \in R_n$. We use d_0' and d_1' as opposed to d_0 and d_1 above to avoid confusing terms with the earlier relinearization process. First, we must define a new evaluation key for this new relinearization process.

	$\texttt{Evk.Scale.Gen}(p, \texttt{sk})$
Input:	$p \in \mathbb{Z}^+$ large integer with $p \gg q$,
	$\texttt{sk} = s \in R_{n,3}$ secret key.
Output:	$\texttt{evk}_{\text{scale}} = (a', b') \in R_{n,pq}^2$ the evaluation key.
Step 1.	Sample $a' \leftarrow U(R_{n,pq})$ and $e' \leftarrow \chi'(R_{n,pq})$.
Step 2.	Compute $b' \in R_{n,pq}$ via $b' \equiv -a's+ps^2+e' \mod (\Phi(x), pq)$.
Step 3.	Return $\texttt{evk}_{\text{scale}} = (a', b')$.

Algorithm 3.7: Scaled Evaluation Key Generation

As before, we call $\mathtt{evk_{scale}}$ the *evaluation key*, which we use as an additional public key for ciphertext multiplications. Notice that since a' is uniform random in $R_{n,pq}$, the second coordinate of $\mathtt{evk_{scale}}$ is essentially a modified encryption of ps^2 in the ring $R_{n,pq}$, and thus s is properly hidden. The reason we take modulo pq here rather than modulo q is specifically because of the multiplication $c_2'b'$. As c_2' is essentially a uniform random element in $R_{n,q}$, this would yield a huge error term $c_2'e'$. So, we must scale this term by $1/p$ in order to reduce its size but keep $c_2's^2$ intact, which is why we compute $\mathtt{evk_{scale}}$ in the ring $R_{n,pq}$. Notice also that instead of drawing the error from $\chi(R_n, pq)$, we draw the error from $\chi'(R_n, pq)$. In this case, χ' will be chosen according to [10]. Write $pq = q^k$ for some real k. Then, we pick χ' such that $\left\|\chi'(R_{n,pq})\right\|_\infty > \alpha^{1-\sqrt{k}}q^{k-\sqrt{k}}\rho^{\sqrt{k}}$ with $\alpha \approx 3.758$. The purpose for choosing χ' this way is to ensure that security is not compromised. Using this definition, we can now find an alternate relinearization similar to before with the following algorithm and subsequent lemma.

	$\mathtt{Relin.V2}((c_0', c_1', c_2'), \mathtt{evk_{scale}})$
Input:	$c_0', c_1', c_2' \in R_{n,q}$ polynomials,
	$\mathtt{evk_{scale}} = (a', b') \in R_{n,pq}^2$ evaluation key.
Output:	$(c_0' + d_0', c_1' + d_1') \in R_{n,q}^2$.
Step 1.	Compute $\beta_0, \beta_1 \in R_{n,pq}$ such that $\beta_0 \equiv c_2'b' \mod (\Phi(x), pq)$
	and $\beta_1 \equiv c_2'a' \mod (\Phi(x), pq)$.
Step 2.	Compute $d_0' = \left\lfloor \frac{\beta_0}{p} \right\rceil$ and $d_1' = \left\lfloor \frac{\beta_1}{p} \right\rceil$.
Step 3.	Return $(c_0' + d_0', c_1' + d_1')$

Algorithm 3.8: Relinearization Version 2

Lemma 3.4.3 *Let* $(c_0' + d_0', c_1' + d_1') = \mathtt{Relin.V2}((c_0', c_1', c_2'), \mathtt{evk_{scale}}) \in R_{n,q}^2$ *be polynomials output by algorithm 3.8 and* $s \in R_{n,q}$ *a secret key. Then,*

$$c_2's^2 + e'' \equiv d_0' + d_1's \mod (\Phi(x), q)$$

where $e'' = \frac{c_2'e'}{p} + \epsilon_0 + \epsilon_1 s$ *for some* $\epsilon_0, \epsilon_1 \in \mathbb{R}[x]$ *with* $\|\epsilon_0\|_\infty, \|\epsilon_1\|_\infty \leq 1/2$.

Proof. Notice that as $\beta_1 \equiv c_2'a' \mod (\Phi(x), pq)$, we can write

$$\beta_1 \equiv c_2'a' + pq\alpha_1 \mod \Phi(x)$$

for some $\alpha_1 \in \mathbb{Z}$. Then,

$$\begin{aligned}
d_1' &= \left\lfloor \frac{\beta_1}{p} \right\rceil \\
&\equiv \left\lfloor \frac{c_2' a' + pq\alpha_1}{p} \right\rceil \mod (\Phi(x), q) \\
&\equiv \frac{c_2' a'}{p} + q\alpha_1 + \epsilon_1 \mod (\Phi(x), q) \\
&\equiv \frac{c_2' a'}{p} + \epsilon_1 \mod (\Phi(x), q)
\end{aligned}$$

for some $\epsilon_1 \in \mathbb{R}[x]$ with $\|\epsilon_1\|_\infty \leq 1/2$. Note also that as $c_2' b' \equiv -c_2' a' s + c_2' e' + c_2' p s^2 \mod (\Phi(x), pq)$, we have that for some $\alpha_0 \in \mathbb{Z}$,

$$\beta_0 \equiv c_2' b' \equiv -c_2' a' s + c_2' e' + c_2' p s^2 + pq\alpha_1 \mod \Phi(x)$$

Then,

$$\begin{aligned}
d_0' &= \left\lfloor \frac{\beta_0}{p} \right\rceil \\
&\equiv \left\lfloor \frac{-c_2' a' s + c_2' e' + c_2' p s^2 + pq\alpha_1}{p} \right\rceil \mod (\Phi(x), q) \\
&\equiv \left\lfloor \frac{-c_2' a' s}{p} + \frac{c_2' e'}{p} + c_2' s^2 + q\alpha_0 \right\rceil \mod (\Phi(x), q) \\
&\equiv \frac{-c_2' a' s}{p} + \frac{c_2' e'}{p} + c_2' s^2 + q\alpha_0 + \epsilon_0 \mod (\Phi(x), q) \\
&\equiv \frac{-c_2' a' s}{p} + \frac{c_2' e'}{p} + c_2' s^2 + \epsilon_0 \mod (\Phi(x), q)
\end{aligned}$$

Therefore,

$$\begin{aligned}
d_0' + d_1' s &\equiv \left(\frac{-c_2' a' s}{p} + \frac{c_2' e'}{p} + c_2' s^2 + \epsilon_0\right) + \left(\frac{c_2' a'}{p} + \epsilon_1\right)s \mod (\Phi(x), q) \\
&\equiv c_2' s^2 - \frac{c_2' a' s}{p} + \frac{c_2' e'}{p} + \epsilon_0 + \frac{c_2' a' s}{p} + \epsilon_1 s \mod (\Phi(x), q) \\
&\equiv c_2' s^2 + \frac{c_2' e'}{p} + \epsilon_0 + \epsilon_1 s \mod (\Phi(x), q) \\
&\equiv c_2' s^2 + e'' \mod (\Phi(x), q)
\end{aligned}$$

as desired. □

As with other lemmas, a full discussion of the corresponding error bound of e'' will take place in chapter 5. Given a degree 2 BFV ciphertext, we now present the following lemma to show that `Relin.V2` returns a valid degree 1 BFV ciphertext.

Lemma 3.4.4 *Let $c'_0, c'_1, c'_2 \in R_{n,q}$ such that $c'_0 + c'_1 s + c'_2 s^2 \equiv D[m_0 m_1]_t + e^* \mod (\Phi(x), q)$ for $m_0, m_1 \in R_{n,t}$ and $e^* \in R_n$. Let $(c'_0 + d'_0, c'_1 + d'_1) = \texttt{Relin.V2}((c'_0, c'_1, c'_2), \texttt{evk}_{\texttt{scale}}) \in R^2_{n,q}$ be the output of algorithm 3.8. Then, $(c'_0 + d'_0, c'_1 + d'_1)$ is a degree 1 BFV ciphertext with error term $e_{\text{mult}} = e^* + e''$.*

Proof. By lemma 3.4.3, we have that $c'_2 s^2 + e'' \equiv d'_0 + d'_1 s \mod (\Phi(x), q)$. Therefore,

$$\begin{aligned}
(c'_0 + d'_0) + (c'_1 + d'_1)s &\equiv c'_0 + c'_1 s + c'_2 s^2 + e'' \mod (\Phi(x), q) \\
&\equiv D[m_0 m_1]_t + e^* + e'' \mod (\Phi(x), q) \\
&\equiv D[m_0 m_1]_t + e_{\text{mult}} \mod (\Phi(x), q)
\end{aligned}$$

□

3.4.3 BFV Full Multiplication

Now that we have described both the partial multiplication and relinearization processes, we can finally define a full BFV multiplication. In fact, we can define two versions of multiplication using the two different versions of relinearization we have constructed. The full multiplications are shown in algorithms 3.9 and 3.10.

It is easy to see that these algorithms have two BFV ciphertexts of degree 1 as an input, and output a single BFV ciphertext of degree 1. The corresponding error bounds are not trivial however, and will be fully discussed in chapter 5.

	$\texttt{BFV.Multiply.V1}(\texttt{ct}_0, \texttt{ct}_1, \texttt{evk}_{\texttt{flat}})$
Input:	$\texttt{ct}_0 = (a_0, b_0), \texttt{ct}_1 = (a_1, b_1) \in R_{n,q}^2$ BFV ciphertexts,
	$\texttt{evk}_{\texttt{flat}}$ evaluation key.
Output:	$\texttt{ct}_2$ a BFV ciphertext.
Step 1.	Compute $(c_0', c_1', c_2') = \texttt{BFV.Int.Mult}(\texttt{ct}_0, \texttt{ct}_1)$.
Step 2.	Compute $(c_0' + d_0, c_1' + d_1) = \texttt{Relin.V1}((c_0', c_1', c_2'), \texttt{evk}_{\text{flat}})$.
Step 3.	Return $\texttt{ct}_2 = (c_0' + d_0, c_1' + d_1)$.

Algorithm 3.9: BFV Multiplication Version 1

	$\texttt{BFV.Multiply.V2}(\texttt{ct}_0, \texttt{ct}_1, \texttt{evk}_{\texttt{scale}})$
Input:	$\texttt{ct}_0 = (a_0, b_0), \texttt{ct}_1 = (a_1, b_1) \in R_{n,q}^2$ BFV ciphertexts,
	$\texttt{evk}_{\texttt{scale}}$ evaluation key.
Output:	$\texttt{ct}_2$ a BFV ciphertext.
Step 1.	Compute $(c_0', c_1', c_2') = \texttt{BFV.Int.Mult}(\texttt{ct}_0, \texttt{ct}_1)$.
Step 2.	Compute $(c_0' + d_0', c_1' + d_1') = \texttt{Relin.V2}((c_0', c_1', c_2'), \texttt{evk}_{\text{scale}})$.
Step 3.	Return $\texttt{ct}_2 = (c_0' + d_0', c_1' + d_1')$.

Algorithm 3.10: BFV Multiplication Version 2

3.5 BGV

We now move to another homomorphic RLWE scheme originally presented in [3] known as BGV. The BGV scheme is still based on the RLWE problem, but the ciphertexts take a slightly different form compared to BFV.

3.5.1 BGV Key Generation, Encryption, and Decryption

Choose a secret $s \in R_{n,q}$ with coefficients in $\{-1, 0, 1\}$. The secret key is $\texttt{sk} = s \in R_{n,3}$. Sample a uniform random $a \leftarrow U(R_{n,q})$ and a random error polynomial $e \leftarrow \chi(R_n)$ such that $\|e\|_\infty \leq \rho$. In BGV, we define the public key $\texttt{pk} = (a, b) \in R_{n,q}^2$ where

$$b \equiv -(as + te) \mod (\Phi(x), q) \tag{3.4}$$

It is easy to see that $\texttt{pk}$ is simply an RLWE sample. Using this public key, we can now define the public key encryption algorithm.

	$\texttt{BGV.Encrypt}(m_0, \texttt{pk})$
Input:	$m_0 \in R_{n,t}$ message,
	$\texttt{pk} = (a, b) \in R_{n,q}^2$ public key,
Output:	$\texttt{ct}_0 = (a_0, b_0) \in R_{n,q}^2$ BGV ciphertext.
Step 1.	Generate a random $u \in R_{n,3}$.
Step 2.	Sample $e_0', e_0'' \leftarrow \chi(R_n)$ such that $\left\|e_0'\right\|_\infty, \left\|e_0''\right\|_\infty \leq \rho$.
Step 3.	Compute $a_0 \in R_{n,q}$ with $a_0 \equiv au + te_0' \mod (\Phi(x), q)$ and $b_0 \in R_{n,q}$ with $b_0 \equiv bu + m_0 + te_0'' \mod (\Phi(x), q)$.
Step 4.	Return $\texttt{ct}_0 = (a_0, b_0)$.

Algorithm 3.11: BGV Encryption Algorithm

Observe that

$$\begin{aligned} b_0 + a_0 s &\equiv bu + m_0 + te_0'' + (au + te_0')s \mod (\Phi(x), q) \\ &\equiv -(as + te)u + m_0 + te_0'' + (au + te_0')s \mod (\Phi(x), q) \\ &\equiv m_0 + t(e_0'' + e_0's - eu) \mod (\Phi(x), q) \end{aligned}$$

Let $e_0 = e_0'' + e_0's - eu$. Then, we simply have

$$b_0 + a_0 s \equiv m_0 + te_0 \mod (\Phi(x), q) \tag{3.5}$$

When referring to a BGV ciphertext, we will always consider it in the format of equation 3.5. The following lemma gives a bound on the size of e_0.

Lemma 3.5.1 *Let $\texttt{ct}_0 = (a_0, b_0)$ be the ouput of algorithm 3.11, where $b_0 + a_0 s \equiv m_0 + te_0$. Then, $\left\|e_0\right\|_\infty \leq 2\rho\delta_R + \rho$.*

Proof. Since $e_0 = e_0'' + e_0's - eu$, we have that

$$\begin{aligned} \left\|e_0\right\|_\infty &= \left\|e_0'' + e_0's - eu\right\|_\infty \\ &\leq \left\|eu\right\|_\infty + \left\|e_0'\right\|_\infty + \left\|se_0''\right\|_\infty \\ &\leq \delta_R \left\|e\right\|_\infty \left\|u\right\|_\infty + \left\|e_0'\right\|_\infty + \delta_R \left\|s\right\|_\infty \left\|e_0''\right\|_\infty \\ &\leq 2\delta_R\rho + \rho \end{aligned}$$

as desired. □

Now, we can decrypt a BGV ciphertext via the algorithm 3.12.

	$\texttt{BGV.Decrypt}(\texttt{ct}_0, \texttt{sk})$
Input:	$\texttt{ct}_0 = (a_0, b_0) \in R_{n,q}^2$ BGV ciphertext,
	$\texttt{sk} = s \in R_{n,3}$ secret key.
Output:	$m_0 \in R_{n,t}$ message.
Step 1.	Compute $m_0 = (b_0 + a_0 s \mod (\Phi(x), q)) \mod t$.
Step 2.	Return m_0.

Algorithm 3.12: BGV Decryption Algorithm

Observe that

$$\begin{aligned}(b_0 + a_0 s \mod (\Phi(x), q)) \mod t &\equiv (te_0 + m_0 \mod (\Phi(x), q)) \mod t \\ &\equiv te_0 + m_0 \mod t \\ &\equiv m_0 \mod t\end{aligned}$$

The second line above follows from the fact that since $te_0 + m_0 \in R_{n,q}$, reducing modulo $(\Phi(x), q)$ results in no change. In other words, $te_0 + m_0 = te_0 + m_0 \mod (\Phi(x), q)$. The requirement we need in order for correct decryption is to have $\|e_0\|_\infty < \frac{q}{2t}$. If this is not the case, then $t\|e_0\|_\infty \geq q/2$, meaning that the modulo t component will be destroyed as te_0 will first need to be reduced modulo q to land in $R_{n,q}$. Not that this error condition is similar (and depending on choices of D, equivalent) to the condition stated in BFV, where we required $\|e_0\|_\infty < D/2$.

3.5.2 BGV Addition

Given two BGV ciphertexts $\texttt{ct}_0 = (a_0, b_0)$ and $\texttt{ct}_1 = (a_1, b_1)$ such that they are of the form

$$b_0 + a_0 s \equiv m_0 + te_0 \mod (\Phi(x), q) \tag{3.6}$$

$$b_1 + a_1 s \equiv m_1 + te_1 \mod (\Phi(x), q) \tag{3.7}$$

Then, we can define BGV ciphertext additions via algorithm 3.13. The subsequent lemma provides for proof of correctness.

	$\texttt{BGV.Add}(\texttt{ct}_0, \texttt{ct}_1)$
Input:	$\texttt{ct}_0 = (a_0, b_0), \texttt{ct}_1 = (a_1, b_1) \in R_{n,q}^2$ BFV ciphertexts.
Output:	$\texttt{ct}_2 \in R_{n,q}^2$.
Step 1.	Compute $\texttt{ct}_2 = \texttt{ct}_0 + \texttt{ct}_1 \mod q$.
Step 2.	Return $\texttt{ct}_2$.

Algorithm 3.13: BGV Addition

Lemma 3.5.2 *Let* $\texttt{ct}_2 = \texttt{BFV.Add}(\texttt{ct}_0, \texttt{ct}_1)$ *be the output of algorithm 3.13. Let* $r \in R_{n,q}$ *such that* $m_0 + m_1 = [m_0 + m_1]_t + tr$. *Then,* $\texttt{ct}_2$ *is a BGV ciphertext with error term*

$$e_{\text{add}} = e_0 + e_1 + r$$

Proof. It is easy to see that as $\texttt{ct}_2 = (a_0 + a_1, b_0 + b_1) \mod q$, we have

$$\begin{aligned}(b_0 + b_1) + (a_0 + a_1)s &\equiv m_0 + m_1 + t(e_0 + e_1) \mod (\Phi(x), q) \\ &\equiv [m_0 + m_1]_t + te_{\text{add}} \mod (\Phi(x), q)\end{aligned}$$

So $\texttt{ct}_2$ is a BGV ciphertext with error term e_{add}, as desired. □

3.5.3 BGV Multiplication

Consider two BGV ciphertexts $\texttt{ct}_0 = (a_0, b_0)$ and $\texttt{ct}_1 = (a_1, b_1)$. As we did for BFV, we will first define an initial multiplication algorithom to produce a degree 2 ciphertext, and then relinearize our terms. The intial multiplication is given by the following algorithm.

	$\texttt{BGV.Int.Mult}(\texttt{ct}_0, \texttt{ct}_1)$
Input:	$\texttt{ct}_0 = (a_0, b_0), \texttt{ct}_1 = (a_1, b_1) \in R_{n,q}^2$ BFV ciphertexts.
Output:	$(c_0', c_1', c_2') \in R_{n,q}^3$.
Step 1.	Compute $c_0' = b_0 b_1, c_1' = b_1 a_0 + b_0 a_1$, and $c_2' = a_0 a_1 \mod (\Phi(x), q)$.
Step 3.	Return (c_0', c_1', c_2').

Algorithm 3.14: BGV Initial Multiplication

Because there is no need to scale by a factor of t/q, BGV multiplication generally has a much easier analysis than that of BFV multiplication. The verification that algorithm 3.14 does indeed produce a degree 2 BGV ciphertext is given in the following lemma.

Lemma 3.5.3 *Let* (c'_0, c'_1, c'_2) *be the output of algorithm* 3.14. *Let* $r_m \in R_{n,q}$ *such that* $m_0 m_1 \equiv [m_0 m_1]_t + t r_m \mod \Phi(x)$. *Then,* (c'_0, c'_1, c'_2) *is a degree 2 BGV ciphertext with error term*

$$e_{\text{mult}} = m_0 e_1 + m_1 e_0 + t e_0 e_1 + r_m$$

Proof. We have that

$$\begin{aligned} c'_0 + c'_1 s + c'_2 s^2 &\equiv b_0 b_1 + (b_1 a_0 + b_0 a_1) s + a_0 a_1 s^2 \mod (\Phi(x), q) \\ &\equiv (b_0 + a_0 s)(b_1 + a_1 s) \mod (\Phi(x), q) \\ &\equiv (m_0 + t e_0)(m_1 + t e_1) \mod (\Phi(x), q) \\ &\equiv m_0 m_1 + t(m_0 e_1 + m_1 e_0 + t e_0 e_1) \mod (\Phi(x), q) \\ &\equiv [m_0 m_1]_t + t(m_0 e_1 + m_1 e_0 + t e_0 e_1 + r_m) \mod (\Phi(x), q) \\ &\equiv [m_0 m_1]_t + t e_{\text{mult}} \mod (\Phi(x), q) \end{aligned}$$

which is a BGV encryption of $m_0 m_1$ with error term e_{mult}, as desired. □

The final result is a degree 2 ciphertext, so we can again use relinearization. The only difference is we will need to compute $\mathbf{v}$ as $\mathbf{v} = s^2\mathbf{g} - \mathbf{u}s + t\mathbf{w}$ in algorithm 3.5 so we match the format of BGV. Thus, we can already fully define two separate multiplcation processes for BGV.

	$\texttt{BGV.Multiply.V1}(\texttt{ct}_0, \texttt{ct}_1, \texttt{evk}_{\texttt{flat}})$
Input:	$\texttt{ct}_0 = (a_0, b_0), \texttt{ct}_1 = (a_1, b_1) \in R^2_{n,q}$ BGV ciphertexts,
	$\texttt{evk}_{\texttt{flat}}$ evaluation key.
Output:	$\texttt{ct}_2$ a BGV ciphertext.
Step 1.	Compute $(c'_0, c'_1, c'_2) = \texttt{BGV.Int.Mult}(\texttt{ct}_0, \texttt{ct}_1)$.
Step 2.	Compute $(c'_0 + d_0, c'_1 + d_1) = \texttt{Relin.V1}((c'_0, c'_1, c'_2), \texttt{evk}_{\text{flat}})$.
Step 3.	Return $\texttt{ct}_2 = (c'_0 + d_0, c'_1 + d_1)$.

Algorithm 3.15: BGV Multiplication Version 1

	$\texttt{BGV.Multiply.V2}(\texttt{ct}_0, \texttt{ct}_1, \texttt{evk}_{\texttt{scale}})$
Input:	$\texttt{ct}_0 = (a_0, b_0), \texttt{ct}_1 = (a_1, b_1) \in R_{n,q}^2$ BGV ciphertexts,
	$\texttt{evk}_{\texttt{scale}}$ evaluation key.
Output:	$\texttt{ct}_2$ a BGV ciphertext.
Step 1.	Compute $(c_0', c_1', c_2') = \texttt{BGV.Int.Mult}(\texttt{ct}_0, \texttt{ct}_1)$.
Step 2.	Compute $(c_0' + d_0', c_1' + d_1') = \texttt{Relin.V2}((c_0', c_1', c_2'), \texttt{evk}_{\text{scale}})$.
Step 3.	Return $\texttt{ct}_2 = (c_0' + d_0', c_1' + d_1')$.

Algorithm 3.16: BGV Multiplication Version 2

3.6 CKKS

We now discuss the CKKS scheme, originally presented in [6]. Unlike BFV and BGV, CKKS allows for approximate homomorphic computation on data rather than exact integer arithmetic. CKKS achieves this by first taking in data as a vector over $\mathbb{C}$, mapping the data into R_n, and then performing homomorphic computation before mapping back into a vector over $\mathbb{C}$. This process of mapping to and from the $\mathbb{C}$-vector space is known as the encoding and decoding procedures, respectively.

3.6.1 CKKS Encoding and Decoding

Recall that $R_n = \mathbb{Z}[x]/(\Phi(x))$. We first must define the canonical embedding map $\sigma : \mathbb{R}[x]/(\Phi(x)) \to \mathbb{C}^n$. Note that since $\Phi(x) = x^n + 1$ is a cyclotomic polynomial of degree n, it has n primitive roots of unity $\zeta_k = e^{2\pi ik/(2n)}$ for $1 \leq k \leq 2n-1$ and $\gcd(k, 2n) = 1$. In this case, this is all odd k since $2n$ is a power of two. The canonical embedding map is then given by the following: for $m \in \mathbb{R}[x]/(\Phi(x))$, $\sigma(m) = (m(\zeta), m(\zeta_3), \ldots, m(\zeta_{2n-1}))$. However, it is not necessary for us to store every entry, as $\zeta_j = \overline{\zeta_{2n-j}}$ when dealing with primitive roots of this cyclotomic polynomial. Therefore, we use a variant of the canonical embedding $\tau : \mathbb{R}[x]/(\Phi(x)) \to \mathbb{C}^{n/2}$ via $\tau(m) = (m(\zeta), m(\zeta_5), \ldots, m(\zeta_{2n-3}))$. Given this canonical embedding, we define the canonical embedding norm as the ℓ_∞ norm of the canonical embedding, which we will denote $\|\cdot\|_\infty^{\texttt{can}}$. Hence, $\|m\|_\infty^{\texttt{can}} = \left\|\sigma(m)\right\|_\infty = \left\|\tau(m)\right\|_\infty$.

Now that we have defined the canonical embedding, we can define the encoding and decoding procedures. The purpose of the encoding procedure is to take a plaintext message $z \in \mathbb{C}^{n/2}$ represented as a fixed precision number and convert it into a polynomial in R_n. The decoding pro-

cedure will take a polynomial in R_n, and revert it back into a fixed precision number in $\mathbb{C}^{n/2}$ that is a good approximation of the plaintext. We now formally present the two procedures as follows: consider $z \in \mathbb{C}^{n/2}$ and a positive scaling factor $\Delta \in \mathbb{Z}$. Then, we have the encoding function

$$\texttt{Ecd}(z, \Delta) = \lfloor \tau^{-1}(\Delta z) \rceil$$

and the decoding function

$$\texttt{Dcd}(m, \Delta) = \Delta^{-1}\tau(m)$$

The rounding in the encoding is needed in order to ensure that m is strictly in $R_n = \mathbb{Z}[x]/(\Phi(x))$ rather than just $\mathbb{R}[x]/(\Phi(x))$ or $\mathbb{Q}[x]/(\Phi(x))$. It is important to note that the scaling factor Δ is directly related to a desired precision of our decimal approximation in our encoding and decoding procedure. As far as finding the inverse σ^{-1}, we can think of finding the unique $(n-1)$th degree polynomial which interpolates the n points:

$$(\zeta_1, z_1), (\zeta_3, z_2), \ldots, (\zeta_{2n-1}, z_n)$$

or, if given a vector $z = (z_1, \ldots, z_{n/2}) \in \mathbb{C}^{n/2}$, we note that $\zeta_j = \overline{\zeta_{2n-j}}$ and therefore for $m \in R_n$, $z_j = m(\zeta_j) = m(\overline{\zeta_{2n-j}}) = \overline{m(\zeta_{2n-j})}$, and consequently $m(\overline{\zeta_j}) = \overline{z_j}$. In other words, we not only know that evaluating the given polynomial at a primitive root of unity should yield a complex number, but also that evaluating the given polynomial at the conjugate of the primitive root should yield the conjugate result. Hence, we wish to construct an interpolation of the n points

$$(\zeta_1, z_1), (\overline{\zeta_1}, \overline{z_1}), (\zeta_5, z_2), (\overline{\zeta_5}, \overline{z_2}), \ldots, (\zeta_{2n-3}, z_{n/2}), (\overline{\zeta_{2n-3}}, \overline{z_{n/2}})$$

To find the actual maps σ or τ, one can use a modified fast fourier transform (FFT), which will be briefly discussed in chapter 4. To demonstrate the encoding and decoding procedure, we provide the following toy example.

Example. Let $R_n = \mathbb{Z}[x]/(x^4 + 1)$, $z = (1.280217 + 1.224319i, 3.332424 - 2.124512i)$, and $\Delta = 10^3$. Note that $x^4 + 1$ has primitive roots $\zeta_1 = e^{\pi i/4}$ and $\zeta_5 = e^{5\pi i/4}$, along with $\overline{\zeta_1}$ and $\overline{\zeta_5}$. In

order to begin our encoding procedure, we first scale and then embed.

$$
\begin{aligned}
\texttt{Ecd}(z, \Delta) &= \lfloor \tau^{-1}(10^3(1.280217 + 1.224319i, 3.332424 - 2.124512i)) \rceil \\
&= \lfloor \tau^{-1}(1280.217 + 1224.319i, 3332.424 - 2124.512i) \rceil \\
&= \lfloor 2306.32 + 458.42x - 450.09x^2 + 1909.55x^3 \rceil \\
&= 2306 + 458x - 450x^2 + 1910x^3
\end{aligned}
$$

We let $m = 2306 + 458x - 450x^2 + 1910x^3$. For decoding, we simply evaluate the polynomial m at the 1st and 5th primitive roots of unity of our cyclotomic polynomial, and then rescale back to a fixed precision decimal.

$$
\begin{aligned}
\texttt{Dcd}(m, \Delta) &= 10^{-3}\tau(2306 + 458x - 450x^2 + 1910x^3) \\
&= 10^{-3}(2306 + 458\zeta_1 - 450\zeta_1^2 + 1910\zeta_1^3, 2306 + 458\zeta_5 - 450\zeta_5^2 + 1910\zeta_5^3) \\
&= 10^{-3}(1279.280 + 1224.428i, 3332.719 - 2124.428i) \\
&= (1.279280 + 1.224428i, 3.332719 - 2.124428i)
\end{aligned}
$$

As we can see, after decoding we do indeed get our original input vector back, but with some small additional error. The scaling factor Δ obviously determines how precise the decoded vector is compared to the original input vector. An exact analysis of how accurate CKKS approximations are will be given in chapter 5.

3.6.2 CKKS Key Generation, Encryption, and Decryption

Choose a secret $s \in R_{n,q}$ with coefficients in $\{-1, 0, 1\}$. The secret key is $\texttt{sk} = s \in R_{n,3}$. Sample a uniform random $a \leftarrow U(R_{n,q})$ and a random error polynomial $e \leftarrow \chi(R_n)$ such that $\|e\|_\infty \leq \rho$. Compute the public key $\texttt{pk} = (a, b) \in R_{n,q}^2$ by computing $b = -as + e \mod (\Phi(x), q)$. Then, the encryption algorithm is given by algorithm 3.17, and decryption by algorithm 3.18.

Unlike BFV and BGV, CKKS decryption does not yield the exact message. From step 2. of the algorithm, we obtain m_0'. However, recall that $b_0 + a_0 s \equiv m_0 + e_0 \mod (\Phi(x), q)$, so in this case we really have that $m_0' = m_0 + e_0$. In other words, m_0' is just an approximation of m_0 so long as e_0 is small. We do not include proofs that decryption works, as it is directly

	`CKKS.Encrypt`$(m_0, \texttt{pk})$
Input:	$z_0 \in \mathbb{C}^{n/2}$ message,
	$\texttt{pk} = (a, b) \in R_{n,q}^2$ public key,
Output:	$\texttt{ct}_0 = (a_0, b_0) \in R_{n,q}^2$ CKKS ciphertext.
Step 1.	Encode z_0 by computing $m_0 = \texttt{Ecd}(z_0, \Delta)$.
Step 2.	Generate a random $u \in R_{n,3}$ such that $\mathrm{wt}(u) \leq \rho$.
Step 3.	Sample $e_0', e_0'' \leftarrow \chi(R_n)$ such that $\left\|e_0'\right\|_\infty, \left\|e_0''\right\|_\infty \leq \rho$.
Step 4.	Compute $a_0 \in R_{n,q}$ with $a_0 \equiv au + e_0' \mod (\Phi(x), q)$ and $b_0 \in R_{n,q}$ with $b_0 \equiv bu + m_0 + e_0'' \mod (\Phi(x), q)$.
Step 5.	Return $\texttt{ct}_0 = (a_0, b_0)$.

Algorithm 3.17: CKKS Encryption Algorithm

	`CKKS.Decrypt`$(\texttt{ct}_0, \texttt{sk})$
Input:	$\texttt{ct}_0 = (a_0, b_0) \in R_{n,q}^2$ CKKS ciphertext,
	$\texttt{sk} = s \in R_{n,3}$ secret key.
Output:	$z_0' \in \mathbb{C}^{n/2}$ message.
Step 1.	Compute $m_0' = b_0 + a_0 s \mod (\Phi(x), q)$.
Step 2.	Decode m_0' by computing $z_0' = \texttt{Dcd}(m_0', \Delta)$.
Step 3.	Return z_0'.

Algorithm 3.18: CKKS Decryption Algorithm

apparent from the algorithm that it is an approximation to the desired message. We also note that many texts, such as the original CKKS paper in [6], have separate steps for encoding/decoding and encryption/decryption. We however, include the encoding or decoding in the encryption or decryption algorithms respectively.

3.6.3 CKKS Addition

Given two CKKS ciphertexts $\texttt{ct}_0 = (a_0, b_0)$ and $\texttt{ct}_1 = (a_1, b_1)$ such that they are of the form

$$b_0 + a_0 s \equiv m_0 + e_0 \mod (\Phi(x), q)$$

$$b_1 + a_1 s \equiv m_1 + e_1 \mod (\Phi(x), q)$$

Then, we can define CKKS ciphertext additions via the following algorithm. The subsequent lemma provides for a proof of correctness.

	$\texttt{CKKS.Add}(\texttt{ct}_0, \texttt{ct}_1)$
Input:	$\texttt{ct}_0 = (a_0, b_0), \texttt{ct}_1 = (a_1, b_1) \in R_{n,q}^2$ CKKS ciphertexts.
Output:	$\texttt{ct}_2 \in R_{n,q}^2$.
Step 1.	Compute $\texttt{ct}_2 = \texttt{ct}_0 + \texttt{ct}_1 \mod q$.
Step 2.	Return $\texttt{ct}_2$.

Algorithm 3.19: CKKS Addition

Lemma 3.6.1 *Let* $\texttt{ct}_2 = \texttt{BFV.Add}(\texttt{ct}_0, \texttt{ct}_1)$ *be the output of algorithm 3.19. Then,* $\texttt{ct}_2$ *is a CKKS ciphertext with error term*

$$e_{\text{add}} = e_0 + e_1$$

Proof. It is easy to see that for the ciphertext $\texttt{ct}_2$, we have

$$b_0 + b_1 + (a_0 + a_1)s \equiv m_0 + m_1 + e_0 + e_1 \mod (\Phi(x), q)$$

which is an encryption of $m_0 + m_1$ with error term $e_{\text{add}} = e_0 + e_1$. □

It is worth mentioning that unlike BFV and BGV, the ring $R_{n,t}$ is not used for the plaintext space. That is, m_0, m_1, and $m_0 + m_1$ are all elements in R_n instead, so reducing $m_0 + m_1$ modulo t is not necessary as it was in BFV or BGV.

3.6.4 CKKS Multiplication

The initial multiplication algorithm and correctness lemma for CKKS are given as follows.

	$\texttt{CKKS.Int.Mult}(\texttt{ct}_0, \texttt{ct}_1)$
Input:	$\texttt{ct}_0 = (a_0, b_0), \texttt{ct}_1 = (a_1, b_1) \in R_{n,q}^2$ CKKS ciphertexts.
Output:	$(c_0', c_1', c_2') \in R_{n,q}^3$.
Step 1.	Compute $c_0' = b_0 b_1, c_1' = b_1 a_0 + b_0 a_1$, and $c_2' = a_0 a_1$.
Step 3.	Return (c_0', c_1', c_2').

Algorithm 3.20: CKKS Initial Multiplication

Lemma 3.6.2 *Let (c_0', c_1', c_2') be the output of algorithm 3.20. Then, (c_0', c_1', c_2') is a degree 2 CKKS ciphertext with error term $e_{\text{mult}} = m_0 e_1 + m_1 e_0 + e_0 e_1$.*

Proof. Consider the two CKKS ciphertexts $\texttt{ct}_0 = (a_0, b_0)$ and $\texttt{ct}_1 = (a_1, b_1)$. Then,

$$\begin{aligned} c_0' + c_1' s + c_2' s^2 &= (b_0 + a_0 s)(b_1 + a_1 s) \\ &\equiv (m_0 + e_0)(m_1 + e_1) \mod (\Phi(x), q) \\ &\equiv m_0 m_1 + m_0 e_1 + m_1 e_0 + e_0 e_1 \mod (\Phi(x), q) \\ &\equiv m_0 m_1 + e_{\text{mult}} \mod (\Phi(x), q) \end{aligned}$$

which is a CKKS encryption of $m_0 m_1$ with error e_{mult}, as desired. □

The algorithm is and lemma is essentially identical to multiplication for BGV, only without needing to reduce modulo t at all. Just as before, full multiplication can then be achieved by simply adding in a slightly modified relinearization process where $\mathbf{v} = s^2\mathbf{g} - \mathbf{u}s + \mathbf{w}$ in algorithm 3.5.

	$\texttt{CKKS.Multiply.V1}(\texttt{ct}_0, \texttt{ct}_1, \texttt{evk}_{\texttt{flat}})$
Input:	$\texttt{ct}_0 = (a_0, b_0), \texttt{ct}_1 = (a_1, b_1) \in R_{n,q}^2$ CKKS ciphertexts,
	$\texttt{evk}_{\texttt{flat}}$ evaluation key.
Output:	$\texttt{ct}_2$ a CKKS ciphertext.
Step 1.	Compute $(c_0', c_1', c_2') = \texttt{CKKS.Int.Mult}(\texttt{ct}_0, \texttt{ct}_1)$.
Step 2.	Compute $(c_0' + d_0, c_1' + d_1) = \texttt{Relin.V1}((c_0', c_1', c_2'), \texttt{evk}_{\text{flat}})$.
Step 3.	Return $\texttt{ct}_2 = (c_0' + d_0, c_1' + d_1)$.

Algorithm 3.21: CKKS Multiplication Version 1

	$\texttt{CKKS.Multiply.V2}(\texttt{ct}_0, \texttt{ct}_1, \texttt{evk}_{\texttt{scale}})$
Input:	$\texttt{ct}_0 = (a_0, b_0), \texttt{ct}_1 = (a_1, b_1) \in R_{n,q}^2$ CKKS ciphertexts,
	$\texttt{evk}_{\texttt{scale}}$ evaluation key.
Output:	$\texttt{ct}_2$ a CKKS ciphertext.
Step 1.	Compute $(c_0', c_1', c_2') = \texttt{CKKS.Int.Mult}(\texttt{ct}_0, \texttt{ct}_1)$.
Step 2.	Compute $(c_0' + d_0', c_1' + d_1') = \texttt{Relin.V2}((c_0', c_1', c_2'), \texttt{evk}_{\text{scale}})$.
Step 3.	Return $\texttt{ct}_2 = (c_0' + d_0', c_1' + d_1')$.

Algorithm 3.22: CKKS Multiplication Version 2

3.7 Remarks on Differences of HE Cryptosytems

Before concluding this chapter, it with worth mentioning the key differences of the three cryptosystems presented thus far. All three systems rely on the hardness of the RLWE problem. The key difference lies in the structure of the ciphertexts presented. For a message m_0 and an error e_0, the three systems roughly follow the structures below.

$$\begin{aligned} \text{BFV}: &\ b_0 + a_0 s \equiv Dm_0 + e_0 \mod (\Phi(x), q) \\ \text{BGV}: &\ b_0 + a_0 s \equiv m_0 + te_0 \mod (\Phi(x), q) \\ \text{CKKS}: &\ b_0 + a_0 s \equiv m_0 + e_0 \mod (\Phi(x), q) \end{aligned}$$

BFV and BGV both return the exact values of m_0 after decryption, whereas CKKS only returns an approximation. CKKS also requires the cyclotomic polynomial to be $x^n + 1$ for n a power of two thanks to the encoding and decoding procedure, whereas any generic cyclotomic will work for the other schemes. BFV requires a constant D in front of the message, which in turn makes multiplication much more difficult (as far as analyzing errors goes), as a scaling step in the form of a modulus reduction needs to be implemented that is not required in BGV or CKKS.

Chapter 4

Advanced Computing Techniques

At this point, we have discussed everything needed for basic homomorphic encryption schemes in theory. This does not mean our techniques are viable in practice however. Since cryptosystems need to be secure, parameters are usually huge, and thus computation is slow. For any of the RLWE based cryptosystems to be viable, we need to be able to do quick computations on data. Many speedups can be developed using a variety of different methods. In this chapter, we will introduce techniques to both improve parameter sizes in cryptosystems, as well techniques to improve computation speeds. We will not actually discuss any running times of algorithms, as this chapter is merely focused on describing the algorithms used for improvements. However, examples and comparisons of parameter choices will be specified in chapter 5 after a more thorough error analysis.

4.1 Modulus Reduction and Leveling

For any homomorphic encryption style (BFV, BGV, or CKKS), recall that in order to decrypt a ciphertext correctly, the noise term in that ciphertext must not be larger than a specified decryption bound. For instance, for a BFV ciphertext pair $(a_0, b_0) \in R_{n,q}^2$ where

$$b_0 + a_0 s \equiv Dm_0 + e_0 \mod (\Phi(x), q)$$

with error $e_0 \in R_{n,q}$, we require that $\|e_0\|_\infty \leq D/2$ in order to ensure correct decryption. When doing arithmetic operations on multiple ciphertexts, say (a_0, b_0) and (a_1, b_1), we can also bound the additional error that is accumulated by addition or multiplication between these ciphertexts by e_{add} or e_{mult} respectively.

In theory, if we have a set of ciphertexts and a desired number and ordering of additions and multiplications, we could simply select q and D large enough so that the resulting total error is within bounds of decryption, thus always guaranteeing a correct decryption. However, in practice the error (particularly from multiplication) grows quickly enough that this is not feasible. In order to deal with this, we will introduce the concept of modulus reduction. The following algorithm describes the process of performing a modulus reduction for BFV.

	$\texttt{BFV.Modreduce}(Q, q, \texttt{ct}_0)$
Input:	$Q \in \mathbb{Z}^+$ an integer,
	$q \in \mathbb{Z}^+$ an integer,
	$\texttt{ct}_0 = (a_0, b_0) \in R^2_{n,Q}$ BFV ciphertext such that $b_0 + a_0 s \equiv D_Q m_0 + e_0 \mod (\Phi(x), Q)$.
Output:	$\texttt{ct}'_0 \in R^2_{n,q}$ such that $b'_0 + a'_0 s \equiv D_q + e_{\text{MR}} \mod (\Phi(x), q)$ for some e_{MR}.
Step 1.	Compute $a'_0 = \lfloor \frac{q a_0}{Q} \rceil$ and $b'_0 = \lfloor \frac{q b_0}{Q} \rceil$
Step 2.	Return $\texttt{ct}'_0 = (a'_0, b'_0) \in R^2_{n,q}$

Algorithm 4.1: BFV Modulus Reduction Algorithm

Algorithm 4.1 takes a ciphertext with integer modulus Q, and returns a ciphertext with integer modulus q. Here, it is assumed that $D_Q = \lfloor Q/t \rfloor$ and $D_q = \lfloor q/t \rfloor$, the standard choice for BFV given a particular modulus. Not only does this algorithm reduce the integer modulus, but also reduces the norm of the error term, thus allowing for further ciphertext computation. The following lemma shows the correctness of algorithm 4.1.

Lemma 4.1.1 *Let $(a'_0, b'_0) = \texttt{BFV.Modreduce}(Q, q, \texttt{ct}_0) \in R^2_{n,q}$ be the output of algorithm 4.1. Let $\epsilon_Q = Q/t - D_Q$, $\epsilon_q = q/t - D_q$, $\epsilon_{a_0} = qa_0/Q - a'_0$, and $\epsilon_{b_0} = qb_0/Q - b'_0$. Then, (a'_0, b'_0) is a BFV ciphertext with error term*

$$e_{\text{MR}} = \frac{qe_0}{Q} + (\epsilon_q - q\epsilon_Q/Q)m_0 - \epsilon_{b_0} + \epsilon_{a_0} s$$

and if $Q > q$, then $\|e_{\text{MR}}\|_\infty \leq \frac{q}{Q}E + \frac{t+1}{2} + \frac{\delta_R \|s\|_\infty}{2}$.

Proof. Recall that $D_Q = \lfloor Q/t \rfloor$ and $D_q = \lfloor q/t \rfloor$. By assumption, we first note that $(a_0, b_0) \in R_{n,Q}$ is a BFV ciphertext. That is,

$$b_0 \equiv -a_0 s + D_Q m_0 + e_0 \mod (\Phi(x), Q)$$

Therefore, there is some integer $r_Q \in \mathbb{Z}$ such that $b_0 + a_0 s \equiv D_Q m_0 + e_0 + Q r_Q \mod \Phi(x)$. Then,

$$\begin{aligned} b_0' &= \frac{q b_0}{Q} - \epsilon_{b_0} \\ &\equiv -\frac{q a_0 s}{Q} + \frac{q D_Q}{Q} m_0 + \frac{q e_0}{Q} - \epsilon_{b_0} + q r_Q \mod \Phi(x) \end{aligned}$$

Note that as $D_Q = Q/t - \epsilon_Q$, we have that $qD_Q/Q = q/t - q\epsilon_Q/Q$. Since $q/t = D_q + \epsilon_q$, we have $qD_Q/Q = D_q + \epsilon_q - q\epsilon_Q/Q$. Therefore,

$$\begin{aligned} b_0' &\equiv -\frac{q a_0 s}{Q} + \frac{q D_Q}{Q} m_0 + \frac{q e_0}{Q} - \epsilon_{b_0} + q r_Q \mod \Phi(x) \\ &\equiv -a_0' s + \epsilon_{a_0} s + D_q m_0 + (\epsilon_q - \frac{q\epsilon_Q}{Q}) m_0 + \frac{q e_0}{Q} - \epsilon_{b_0} + q r_Q \mod \Phi(x) \\ &\equiv -a_0' s + D_q m_0 + e_{\mathrm{MR}} \mod (\Phi(x), q) \end{aligned}$$

Therefore, $b_0' + a_0' s \equiv D_q m_0 + e_{\mathrm{MR}} \mod (\Phi(x), q)$. Furthermore, if $Q > q$, then we have that $0 < q/Q < 1$. Noting that as $Q/t = D_Q + \epsilon_Q$ and $Q/t \geq D_Q$, then $0 \leq \epsilon_Q \leq 1$. Similarly, $0 \leq \epsilon_q \leq 1$. Then $|\epsilon_q - q\epsilon_Q/Q| \leq 1$. So,

$$\begin{aligned} \|e_{\mathrm{MR}}\|_\infty &\leq \left\| \frac{q e_0}{Q} + (\epsilon_q - q\epsilon_Q/Q) m_0 - \epsilon_{b_0} + \epsilon_{a_0} s \right\|_\infty \\ &\leq \frac{q}{Q} \|e_0\|_\infty + |\epsilon_q - q\epsilon_Q/Q| \frac{t}{2} + \|\epsilon_{b_0}\|_\infty + \|\epsilon_{a_0} s\|_\infty \\ &\leq \frac{q}{Q} E + \frac{t+1}{2} + \frac{\delta_R \|s\|_\infty}{2} \end{aligned}$$

□

Although this bound always holds, it is not very good for large values of t. However, if we place a specific restriction on t, we will see that this bound greatly improves.

Corollary 4.1.1 *If $t|(Q-1)$ and $t|(q-1)$, then $\|e_{\mathrm{MR}}\|_\infty < \frac{q}{Q} E + 1 + \frac{\delta_R \|s\|_\infty}{2}$*

Proof. If $t|(Q-1)$ and $t|(q-1)$, then

$$\frac{Q-1}{t} = \frac{Q}{t} - \frac{1}{t} = \left\lfloor \frac{Q}{t} \right\rfloor = D_Q$$

so $D_Q = (Q-1)/t$ and $\epsilon_Q = 1/t$. Similarly, $D_q = (q-1)/t$ and $\epsilon_q = 1/t$. Then,

$$|\epsilon_q - q\epsilon_Q/Q| = \frac{1}{t}(1 - \frac{q}{Q}) < \frac{1}{t}$$

Then, the bound on e_{MR} from lemma 4.1.1 becomes

$$\|e_{\mathrm{MR}}\|_\infty < \frac{q}{Q}E + 1 + \frac{\delta_R \|s\|_\infty}{2}$$

□

We will now discuss a technique known as modulus leveling in order to design an efficient homomorphic encryption scheme. Let $q_{\ell+1} > q_\ell > \cdots > q_0$ be distinct primes, and define $Q_0, \ldots, Q_{\ell+1}$ as

$$Q_i = \prod_{j=0}^{i} q_j$$

We call Q_i the modulus at level i. It is easy to see that $Q_i/Q_{i-1} = q_i$ for any $i \in \{1, \ldots, \ell+1\}$. With this ascending chain of integers, we can use algorithm 4.1 to reduce the modulus as computation happens in a systematic way. If we run $\texttt{BFV.Modreduce}(Q_i, Q_{i-1}, \texttt{ct}_0)$ for any $1 \leq i \leq \ell+1$, then we will reduce our modulus by q_i. In fact, by lemma 4.1.1, we know that performing modulus reduction produces a ciphertext with e_{MR} such that

$$\begin{aligned} \|e_{\mathrm{MR}}\|_\infty &< \frac{Q_{i-1}}{Q_i}E + \frac{t+1}{2} + \frac{\delta_R \|s\|_\infty}{2} \\ &= \frac{1}{q_i}E + \frac{t+1}{2} + \frac{\delta_R \|s\|_\infty}{2} \end{aligned}$$

or equivalently,

$$q_i > \frac{2E}{2\|e_{\mathrm{MR}}\|_\infty - t - 1 - \delta_R \|s\|_\infty}$$

If we choose some constant $C > \|e_{\mathrm{MR}}\|_\infty$ in the equation above, then we can guarantee that modulus reduction will always return an error bounded by C so long as this bound on q_i above is met. The

case used in corollary 4.1.1 provides for an even better bound,

$$q_i > \frac{2E}{2C - 2 - \delta_R \|s\|_\infty}$$

We use C as the choice for the upper bound $\|e_{\mathrm{MR}}\|_\infty$. What these bounds mean is that given parameters δ_R, C, t, and a current ciphertext error E, we can choose q_i such that this bound holds. This is useful because we can compute on ciphertexts, and then periodically refresh the ciphertext back to an error bound of C if the q_i's are chosen well. The difficulty in this is the fact that E varies with computation. This result is very comparable to the result in [4], but here we use BFV rather than BGV, which requires the extra condition of t dividing $Q-1$ and $q-1$ to achieve the same bound. It is worth noting too that in the case of $t=2$ and assuming $\mathtt{wt}(s) = \rho$, the bound can be improved to

$$q_i > \frac{2E}{\rho - 3}$$

We will return to this idea in chapter 5, as it does require a good amount of thought and error analysis in order to make good choices of q_i.

Though we have yet to pick our actual moduli, we can still define an alternate encryption method to include a built in modulus reduction. The idea is that we will encrypt our message as normal, and then perform a modulus reduction immediately to ensure we are working with a fresh ciphertext bounded by C. Algorithm 4.2 gives the modified encryption algorithm.

	$\mathtt{BFV.Encrypt.MR}(m_0, \mathtt{pk})$
Input:	$m_0 \in R_{n,t}$ message,
	$\mathtt{pk} = (a, b) \in R^2_{n,Q_{\ell+1}}$ public key.
Output:	$\mathtt{ct}'_0 \in R^2_{n,Q_\ell}$.
Step 1.	Generate a random $u \in R_{n,3}$.
Step 2.	Sample $e'_0, e''_0 \leftarrow \chi(R_n)$ such that $\|e'_0\|_\infty, \|e''_0\|_\infty \leq \rho$.
Step 3.	Compute $\mathtt{ct}_0 = (a_0, b_0) \in R^2_{n,Q_{\ell+1}}$ where $a_0 \equiv au + e'_0 \mod (\Phi(x), Q_{\ell+1})$ and $b_0 \equiv bu + Dm_0 + e''_0 \mod (\Phi(x), Q_{\ell+1})$.
Step 4.	Compute $(a'_0, b'_0) = \mathtt{BFV.Modreduce}(Q_{\ell+1}, Q_\ell, \mathtt{ct}_0)$.
Step 5.	Return $\mathtt{ct}'_0 = (a'_0, b'_0) \in R^2_{n,Q_\ell}$.

Algorithm 4.2: BFV Modulus Reduction Encryption

We do not cover the case of BGV in this thesis regarding modulus reduction and leveling, but techniques of error bound control can be implemented there too. It is worth discussing that in

CKKS, a similar procedure known as *rescaling* occurs. The rescaling procedure is given in algorithm 4.3.

	$\texttt{CKKS.RS}(Q,q,\texttt{ct}_0)$
Input:	$Q \in \mathbb{Z}^+$ an integer,
	$q \in \mathbb{Z}^+$ an integer,
	$\texttt{ct}_0 = (a_0, b_0) \in R_{n,Q}^2$, a CKKS ciphertext.
Output:	$\texttt{ct}_0' = (a_0', b_0') \in R_{n,q}^2$, a CKKS ciphertext.
Step 1.	Compute $a_0' = \lfloor \frac{qa_0}{Q} \rceil$ and $b_0' = \lfloor \frac{qb_0}{Q} \rceil$.
Step 2.	Return $\texttt{ct}_0' = (a_0', b_0') \in R_{n,q}^2$.

Algorithm 4.3: CKKS Rescaling

This procedure is essentially a modulus reduction like we've explained for BFV. The key difference however is the purpose of the procedure. Rather than using modulus reduction as a form of error control, here it is used to control sizes of bit representations. For two messages $m_0, m_1 \in R_n$, ciphertext multiplication yields an encryption of the term $m_0 m_1$, which is larger as far as storing goes. Seeing as we will convert it back into a complex vector, we want to get rid of the least significant bits (LSBs) while doing computation. That is, we will get rid of the very small decimal terms accumulated during multiplication that do not matter, as CKKS is only concerned with approximating computations.

Lemma 4.1.2 *Let* $(a_0', b_0') = \texttt{CKKS.RS}(Q, q, \texttt{ct}_0) \in R_{n,q}^2$ *be the output of algorithm 4.3. Let* $\epsilon_{a_0} = qa_0/Q - a_0'$, *and* $\epsilon_{b_0} = qb_0/Q - b_0'$. *Then,* (a_0', b_0') *is a CKKS ciphertext with error term*

$$e_{\mathrm{MR}} = \frac{qe_0}{Q} + (\epsilon_q - q\epsilon_Q/Q)m_0 - \epsilon_{b_0} + \epsilon_{a_0} s$$

and if $Q > q$, *then* $\|e_{\mathrm{MR}}\|_\infty \leq \frac{q}{Q}E + \frac{t+1}{2} + \frac{\delta_R \|s\|_\infty}{2}$.

Proof. By assumption, we first note that $(a_0, b_0) \in R_{n,Q}$ is a CKKS ciphertext. That is,

$$b_0 \equiv -a_0 s + m_0 + e_0 \mod (\Phi(x), Q)$$

Therefore, there is some integer $r_Q \in \mathbb{Z}$ such that $b_0 + a_0 s \equiv m_0 + e_0 + Qr_Q \mod \Phi(x)$. Then,

$$\begin{aligned}
b_0' &= \frac{qb_0}{Q} - \epsilon_{b_0} \\
&\equiv -\frac{qa_0 s}{Q} + \frac{q}{Q}m_0 + \frac{qe_0}{Q} - \epsilon_{b_0} + qr_Q \mod \Phi(x) \\
&\equiv -a_0' s + \epsilon_{a_0} s + \frac{q}{Q}m_0 + (\epsilon_q - \frac{q\epsilon_Q}{Q})m_0 + \frac{qe_0}{Q} - \epsilon_{b_0} + qr_Q \mod \Phi(x) \\
&\equiv -a_0' s + \frac{q}{Q}m_0 + e_{\mathrm{MR}} \mod (\Phi(x), q)
\end{aligned}$$

Therefore, $b_0' + a_0' s \equiv \frac{q}{Q}m_0 + e_{\mathrm{MR}} \mod (\Phi(x), q)$. The bound on e_{MR} remains the same as in lemma 4.1.1, with

$$\|e_{\mathrm{MR}}\|_\infty \leq \frac{q}{Q}E + \frac{t+1}{2} + \frac{\delta_R \|s\|_\infty}{2}$$

□

A notable difference is the CKKS.RS algorithm returns an encryption of $\frac{q}{Q}m_0$ rather than the original m_0. As mentioned, this is intentional, as we wish to reduce the size of m_0 bit usage becomes an issue. The reason we use a modulus reduction algorithm rather than simply trying to scale down the ciphertext is because we are taking entries modulo Q. For a ciphertext $(a_0, b_0) \in R_{n,Q}^2$, if we computed a scaled ciphertext $(\lfloor a_0/\Delta \rceil, \lfloor b_0/\Delta \rceil)$ for some scaling factor Δ, we would first need to write $b_0 + a_0 s \equiv m_0 + e_0 + Qr_q$. This would result in a term approximately equal to Qr_q/Δ after dividing through by Δ, which is no longer equivalent to 0 mod Q and would result in a huge error term.

4.2 RNS BFV

In particular, modulus leveling comes in handy when discussing residue number system (RNS) designs of systems. An RNS design of a homomorphic encryption scheme uses the Chinese Remainder Theorem (CRT) to break down a ciphertext modulus into individual pieces such that computation can be done component wise. As of 2018, RNS variants exist for all of BFV, BGV, and CKKS. In this section, we will discuss the basics of the Chinese Remainder theorem and the RNS variant of BFV first presented in [12]. Though we do not discuss RNS variants of BGV and CKKS,

we point the reader to sources such as [5] and [15] for discussions on these systems.

Let $q = \prod_{i=1}^{\ell} q_i$ where each q_i is pairwise coprime. In this thesis, we will often just choose each q_i to be prime. By the Chinese Remainder Theorem,

$$\mathbb{Z}_q \simeq \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_\ell}$$

There are also unique integers $E_1, \ldots, E_\ell$ such that

$$E_i \equiv 1 \mod q_i \quad \text{and} \quad E_i \equiv 0 \mod q_j \text{ for } i \neq j$$

We can write each $a \in \mathbb{Z}_q$ uniquely as

$$a \equiv \sum_{i=1}^{\ell} E_i a_i \mod q$$

where $a_i \in \mathbb{Z}_{q_i}$. We call $(E_1, \ldots, E_\ell)$ as *the CRT basis* of $\mathbb{Z}_q$ for $q = q_1 \cdots q_\ell$. Under this basis, we write a as

$$\mathrm{CRT}(a) = (a_1, \ldots, a_\ell)$$

where $a \equiv a_i \pmod{q_i}$ for $1 \leq i \leq \ell$. Naturally, we can extend this to the ring $R_{n,q}$, as

$$R_{n,q} \cong \prod_{i=1}^{\ell} R_{n,q_i} \tag{4.1}$$

This isomorphic mapping is given by applying the Chinese Remainder Theorem. Now, we can use the direct product ring to do our computations, as we can simply do coordinate-wise operations in the product ring $\prod_{i=1}^{\ell} R_{n,q_i}$ [5].

It is easy to see that homomorphic addition works in this product ring, as for any variant we simply add together our ciphertexts coordinate-wise and reduce modulo something as necessary in each component. What is not so easy is multiplication. Here, we will describe the process of BFV RNS multiplication as presented in [12]. First, we compute the coefficients $c_0 = b_0 b_1, c_1 = b_0 a_1 + b_1 a_0$, and $c_2 = a_0 a_1$ in R_n with double-precision floating point arithmetic. Then, we compute $c_i' = \lfloor \frac{t}{q} c_i \rceil \in R_{n,q}$ and apply relinearization. At the surface, this seems pretty reasonable. There is

difficulty in computing our polynomials in R_n however, as there is not an isomorphism to a product ring via the Chinese remainder theorem since R_n does not contain an integer modulus. Therefore, we will instead extend our CRT basis to the ring $R_{n,pq}$ for some p and ensure that no modular reduction takes place in order to yield a CRT representation of our polynomials with the same coefficients as we would have in R_n. In order to do this multiplication process, we will need to introduce a number of algorithms. The first algorithm which we call `BFV.RNS.Conv` will convert an integer from one CRT basis to another.

	`BFV.RNS.Conv`$(q, p, (x_1, \dots, x_L))$
Input:	$q = q_1 \cdots q_L \in \mathbb{Z}^+$ product of distinct primes,
	$p = p_1 \cdots p_{L'} \in \mathbb{Z}^+$ product of distinct primes coprime to q,
	$(x_1, \dots, x_L)$ the CRT respresentation of $x \in \mathbb{Z}_q$ with respect to the basis $\{q_1, \dots, q_L\}$.
Output:	$[x]_p \in \mathbb{Z}_p$.
Step 1.	For all $i \in \{1, \dots, L\}$, compute $q_i^* = q/q_i \in \mathbb{Z}$ and $\tilde{q}_i \equiv {q_i^*}^{-1} \bmod q_i \in \mathbb{Z}_{q_i}$.
Step 2.	For all $i \in \{1, \dots, L\}$, compute $y_i = [x_i \tilde{q}_i]_{q_i}$ in single-precision integer arithmetic.
Step 3.	For all $i \in \{1, \dots, L\}$, compute $z_i = y_i/q_i$ in double floating point arithmetic.
Step 4.	Compute $v = \left\lfloor \sum_{i=1}^{L} z_i \right\rceil$.
Step 5.	Compute $[x]_p = \left[(\sum_{i=1}^{L} y_i [q_i^*]_p) - v[q]_p \right]_p$.
Step 6.	Return $[x]_p$.

Algorithm 4.4: BFV RNS Conversion Algorithm

It is worth noting that we use L instead of ℓ in the above, as we can apply this to any modulus Q_L such that $1 \le L \le \ell$. The following lemma and proof shows the correctness of algorithm 4.4.

Lemma 4.2.1 *If all computation is done in IEEE 754 double floats, then the output of algorithm 4.4 is indeed* $[x]_p$.

Proof. For a real number a, let $fl(a)$ be the floating point approximation of a. First, by step 3 of the algorithm, we compute v as

$$v = \left\lfloor \sum_{i=1}^{L} z_i \right\rceil$$

where $z_i = fl(y_i/q_i)$. We define ϵ_{mach}, called machine epsilon, to be the smallest positive real

number such that $fl(1+\epsilon_{\text{mach}}) > 1$. Machine epsilon is an upper bound for the error in representing a nonzero real number in a floating point approximation [13]. It is well known that in IEEE 754 double floats, $\epsilon_{\text{mach}} = 2^{-53}$. Therefore, as each z_i is the floating point representation of y_i/q_i, we can write $z_i = y_i/q_i + \epsilon_i$ for some ϵ_i with $|\epsilon_i| < \epsilon_{\text{mach}}$. Now, note that

$$v = \left\lfloor \sum_{i=1}^{L} z_i \right\rceil = \left\lfloor \sum_{i=1}^{L} y_i/q_i + \epsilon_i \right\rceil$$

Let $\epsilon = \sum_{i=1}^{L} \epsilon_i$. Then, we have that $|\epsilon| \leq \sum_{i=1}^{L} |\epsilon_i| < L \cdot 2^{-53}$. Note that the value we wish to approximate, $\sum_{i=1}^{L} y_i/q_i$, is such that $\sum_{i=1}^{L} y_i/q_i \equiv x/q \mod q$. Therefore, if we require that $|x| < q/4$, we then have that $|\sum z_i| = |\sum y_i/q_i + \epsilon| < |x|/q + |\epsilon| \mod 1 < 1/4 + L \cdot 2^{-53}$. So, $\sum z_j \in (\mathbb{Z} - 1/4 - L \cdot 2^{-53}, \mathbb{Z} + 1/4 + L \cdot 2^{-53})$. Assuming L is reasonable, this set is disjoint from $[\mathbb{Z} + 1/2 - L \cdot 2^{-53}, \mathbb{Z} + 1/2 + L \cdot 2^{-53}]$. v will only have potential to be computed wrong if $\sum z_j$ lands within the set $[\mathbb{Z} + 1/2 - L \cdot 2^{-53}, \mathbb{Z} + 1/2 + L \cdot 2^{-53}]$, as the additional error of ϵ that occurs then has the possibility of producing an incorrect rounding. Considering this, we have

$$v = \left\lfloor \sum_{i=1}^{L} z_i \right\rceil = \left\lfloor \sum_{i=1}^{L} y_i/q_i + \epsilon_i \right\rceil = \sum_{i=1}^{L} y_i/q_i$$

Therefore, v is the unique integer satisfying

$$x = (\sum_{i=1}^{L} [x_i \cdot \tilde{q}_i]_{q_i} \cdot q_i^*) - vq$$

Taking this equation modulo p gives

$$[x]_p = \left[(\sum_{i=1}^{L} [x_i \cdot \tilde{q}_i]_{q_i} \cdot q_i^*) - vq \right]_p = \left[(\sum_{i=1}^{L} y_i [q_i^*]_p) - v[q]_p \right]_p$$

So, algorithm 4.4 outputs $[x]_p$, as desired. □

The next algorithm, called `CRT.Scale`, scales an integer in $\mathbb{Z}_q$ by approximately t/q to an integer in $\mathbb{Z}_t$. The procedure is outlined in algorithm 4.5.

	$\texttt{CRT.Scale}(q, t, (x_1, \ldots, x_L))$
Input:	$q = q_1 \cdots q_L \in \mathbb{Z}^+$ product of distinct primes,
	$t \in \mathbb{Z}^+$ an integer modulus,
	$(x_1, \ldots, x_L)$ the CRT respresentation of $x \in \mathbb{Z}_q$ with respect to the basis $\{q_1, \ldots, q_L\}$.
Output:	$y = [w + v]_t \in \mathbb{Z}_t$.
Precomp.	For each i, compute rational numbers ω_i, θ_i such that $\omega_i + \theta_i = t\tilde{q}_i/q_i$ with $\omega_i \in \mathbb{Z}_t, \theta_i \in [-\frac{1}{2}, \frac{1}{2})$.
Step 1.	Compute $w = \left[\sum_{i=1}^{L} x_i\omega_i\right]_t$ using integer arithmetic and $v = \left\lfloor \sum_{i=1}^{L} x_i\theta_i \right\rceil$ using floating point arithmetic.
Step 2.	Return $y = [w + v]_t$.

Algorithm 4.5: BFV RNS Scaling

Lemma 4.2.2 *Let $y = \texttt{CRT.Scale}(q, t, (x_1, \ldots, x_L))$ be the output of algorithm 4.5. Then, $y = \lfloor \frac{t}{q} x \rceil$.*

Though we do not provide a proof of lemma 4.2.2, a proof can be found in [12]. The only source of computational error comes from computing $\left\lfloor \sum_{i=1}^{L} x_i\theta_i \right\rceil$, which again can be shown to not have an impact if done in IEEE 754 double floats.

The final algorithm we introduce is a modification of CRT.Scale, but with a more complicated structure. Rather than scaling coefficients from $\mathbb{Z}_q$ to $\mathbb{Z}_t$ with a factor of t/q, we want to still scale by t/q, but going from rings $\mathbb{Z}_{pq}$ to $\mathbb{Z}_q$. The procedure is shown in algorithm 4.6. Additionally, we can now also compute the first initial step of multiplication, shown in algorithm 4.7. The initial multiplication step consists of extending the ring, performing the computations of c_0, c_1, and c_2, then performing the complex scaling procedure.

	$\texttt{Complex.CRT.Scale}(q, t, p, (x_1, \ldots, x_L, x'_1, \ldots, x_{L'}))$,
Input:	$q = q_1 \cdots q_L \in \mathbb{Z}^+$ product of distinct primes,
	$p = p_1 \cdots p_{L'} \in \mathbb{Z}^+$ product of distinct primes coprime to q,
	$t \in \mathbb{Z}^+$ integer modululus,
	$(x_1, \ldots, x_L, x'_1, \ldots, x_{L'})$ the CRT respresentation of $x \in \mathbb{Z}_{pq}$ with respect to the basis $\{q_1, \ldots, q_L, p_1, \ldots, q_{L'}\}$.
Output:	$y = \lfloor \frac{t}{q} x \rceil \in \mathbb{Z}_q$.
Step 1.	Compute $y' = \texttt{CRT.Scale}(pq, pt, (x_1, \ldots, x_L, x'_1, \ldots, x'_{L'}))$ and discard the modulo t component to obtain $y' \in \mathbb{Z}_p$.
Step 2.	Compute the CRT representation of y' with respect to the basis $\{p_1, \ldots, p_{L'}\}$ as $(y'_1, \ldots, y'_{L'})$.
Step 3.	Compute $[y']_{pq} = \texttt{BFV.RNS.Conv}(p, pq, (y'_1, \ldots, y'_{L'}))$ and discard the modulo p_j components to obtain $y = [y']_q$.
Step 4.	Return y.

Algorithm 4.6: BFV Complex CRT Scaling

	$\texttt{BFV.RNS.Int.Mult}(\texttt{ct}_0, \texttt{ct}_1)$,
Input:	$\texttt{ct}_0 = (a_0, b_0)$, $\texttt{ct}_1 = (a_1, b_1)$ BFV ciphertexts given in CRT representation.
Output	$(c'_0, c'_1, c'_2) \in R^3_{n,q}$.
Step 1.	For each coefficient x of a_0, b_0, a_1, b_1, compute $[x]_p = \texttt{BFV.RNS.Conv}(q, pq, (x_1, \ldots, x_L))$ and then compute the CRT components $(x'_1, \ldots, x'_{L'})$ with $x'_j = [x]_{p_j}$.
Step 2.	With each x represented by $(x_1, \ldots, x_L, x'_1, \ldots, x_{L'})$, compute $c_0 = [b_0 b_1]_{pq}$, $c_1 = [b_0 a_1 + b_1 a_0]_{pq}$, and $c_2 = [a_0 a_1]_{pq}$ in the ring $R_{n,pq}$.
Step 3.	For each coefficient x of c_0, c_1, c_2, compute $x^* = \texttt{Complex.CRT.Scale}(q, t, p, (x_1, \ldots, x_L, x'_1, \ldots, x_{L'}))$ and obtain c'_0, c'_1, c'_2 respectively.
Step 4.	Return $(c'_0, c'_1, c'_2) \in R^3_{n,q}$.

Algorithm 4.7: BFV RNS Conversion Algorithm

Now, just as we had with original BFV multiplication, we now have a degree 2 BFV ciphertext, so we must relinearize. The process we use is essentially what is used in [12], but we provide for some more in-depth explanation.

For our flattening we choose a $B \in \mathbb{Z}^+$ with the smallest integer γ such that $B^\gamma > q_i$ for all q_i. For each polynomial vector $\text{CRT}(z) = (z_1, \ldots, z_\ell) \in R_{n,q_1}[x] \times \cdots \times R_{n,q_\ell}[x]$, we can represent

each z_i under our flattening as

$$z_i = \sum_{j=0}^{\gamma-1} z_{ij} B^j \mod q_i, \quad 0 \le j \le \gamma - 1 \tag{4.2}$$

where $z_{ij} \in R_{n,q_i}[x]$ with $\|z_{ij}\|_\infty \le B/2$. Note that since $\|z_i\|_\infty < q_i/2$ for each i, then equation (4.2) can be computed in $\mathbb{Z}[x]$ rather than R_{n,q_i} since modular reduction is unnecessary. Define the flattened vector of each CRT component as

$$F(z_i) = (z_{i1}, \ldots, z_{i(\gamma-1)})$$

and the flattened vector of CRT(z) as

$$F(z) := (F(z_1), F(z_2), \ldots, F(z_\ell))$$

We can see that $F(z_i)$ represents the flattening of the ith component of CRT(z) with respect to base B, whereas $F(z)$ represents the flattening of all components in CRT(z) with respect to base B. Define the $(\ell\gamma) \times \ell$ matrix

$$G = I_\ell \otimes (1, B, \ldots, B^{\gamma-1})^T$$

Since we can write z as a sum of CRT components, namely $z = \sum E_i z_i$, we can express z with the following vector

$$\text{CRT}(z) = F(z) \cdot G,$$

Let U be an $\ell\gamma \times \ell$ matrix such that the kth column of U is drawn uniform randomly from R_{n,q_k}. For $1 \le i \le \ell$, $1 \le j \le \gamma$, pick a random $w_{ij} \in R_n$ such that $\|w_{ij}\|_\infty \le \rho$. Let W be the $\ell\gamma \times \ell$ matrix where every entry in row ij is w_{ij}. Define V as the $\ell\gamma \times \ell$ matrix

$$V := Us - s^2 G + W \tag{4.3}$$

where the kth column is computed modulo $(\Phi(x), q_k)$. Then, the matrix pair $ek = (U, V)$ is the evaluation key. The process can be summarized with the algorithm below.

	`RNS.Evk.Flat.Gen`$(B, \texttt{sk})$
Input:	$B \in \mathbb{Z}^+$ a base,
	$\texttt{sk} = s$ secret key.
Output:	$\texttt{RNS.evk}_{\text{flat}} = (U, V) \in R_n^{\ell\gamma \times \ell} \times R_n^{\ell\gamma \times \ell}$ evaluation key.
Step 1.	Compute $\gamma = \lceil \log_B(\max\{q_i\}) \rceil$ and matrix $G = I_\ell \otimes \left(1, B, B^2, \ldots, B^{\gamma-1}\right)$.
Step 2.	Create U a $\ell\gamma \times \ell$ matrix such that the kth column of U is drawn uniform randomly from R_{n,q_k}. For $1 \le i \le \ell$, $1 \le j \le \gamma$, pick a random $w_{ij} \in R_n$ such that $\left\|w_{ij}\right\|_\infty \le \rho$. Define W as the $\ell\gamma \times \ell$ matrix where every entry in row ij is w_{ij}.
Step 3.	Compute $V = Us - s^2 G + W$ where the kth column is computed modulo $(\Phi(x), q_k)$.
Step 4.	Return $\texttt{RNS.evk}_{\text{flat}} = (U, V)$.

Algorithm 4.8: RNS Flattened Evaluation Key Generation

4.2.1 CRT Relinearization Version 1

Let T be the current modulus level. In other words, the modulus q is given by $q = \prod_{i=1}^{T} q_i$ with $1 \le T \le \ell$. Using the evaluation key defined above, we can obtain a valid encryption under s of two ciphertexts. This is similar to what we did in the previous section, only with the additional condition that we are using a representation in the CRT basis for all of our polynomials. The following lemma describes the multiplication process using the evaluation key in the CRT basis.

Lemma 4.2.3 *Given the CRT representations of two ciphertext pairs* $(a_0, b_0), (a_1, b_1)$ *and the evaluation key* (U, V)*, define the polynomials*

$$r_0 := \langle F(c_2')U, 1\rangle, \quad r_1 := \langle F(c_2')V, 1\rangle,$$

Then, $r_1 s + r_0 \equiv c_2' s^2 + w$ *for some error* w.

The above lemma can be applied to any modulus level by simply replace U and V with the submatrix of U and V consisting of the first T columns and $T\gamma$ rows of U and V, respectively. We now present the proof of lemma 4.2.3.

Proof. Let $\text{CRT}(c_2') = (d_1, \ldots, d_\ell)$ be the CRT representation of c_2'. By definition, there exists

integers $E_1, \ldots, E_T$ such that

$$c_2' \equiv \sum_{k=1}^{\ell} E_k d_k \mod q$$

For each individual d_k, we can write

$$d_k \equiv \sum_{j=1}^{\gamma} B^{j-1} d_{kj} \mod q_k$$

where $d_{kj} \in R_{n,q_k}$ with $\left\|d_{kj}\right\|_\infty \leq B/2$. Notice that here we let j go from 1 to γ, rather than 0 to $\gamma - 1$. Combining these, we have

$$c_2' s^2 \equiv \sum_{k=1}^{\ell} \sum_{j=1}^{\gamma} d_{kj} E_k B^{j-1} s^2 \mod q \tag{4.4}$$

For each i, j, k we have some $v_{ij,k}$ where

$$v_{ij,k} \in R_{n,q_k} \equiv -u_{ij,k} s + E_i B^{j-1} s^2 + w_{ij} \mod (\Phi(x), q_k)$$

Notice that for the $E_i B^{j-1} s^2$ term,

$$E_i B^{j-1} s^2 \equiv \begin{cases} B^{j-1} s^2 \mod q_k & \text{if } i = k \\ 0 \mod q_k & \text{if } i \neq k \end{cases}$$

We can then rewrite equation 4.4 as

$$c_2' s^2 \equiv \sum_{k=1}^{T} \sum_{j=1}^{\gamma} d_{kj} E_k B^{j-1} s^2 \mod q \equiv \sum_{k=1}^{T} \sum_{i=1}^{T} \sum_{j=1}^{\gamma} d_{kj} E_i B^{j-1} s^2 \mod q$$

This allows us to write $c_2's^2$ as a summation over all entries in U, V and W. Observe:

$$\begin{aligned}
c_2's^2 &\equiv \sum_{k=1}^{T}\sum_{j=1}^{\gamma} d_{kj}E_kB^{j-1}s^2 \mod q \\
&\equiv \sum_{k=1}^{T}\sum_{i=1}^{T}\sum_{j=1}^{\gamma} d_{kj}E_iB^{j-1}s^2 \mod q \\
&\equiv \sum_{k=1}^{T}\sum_{i=1}^{T}\sum_{j=1}^{\gamma} d_{kj}(v_{ij,k} + u_{ij,k}s - w_{ij}) \mod (\Phi(x), q)
\end{aligned}$$

Then,

$$\begin{aligned}
c_2's^2 + \sum_{k=1}^{T}\sum_{i=1}^{T}\sum_{j=1}^{\gamma} d_{kj}w_{ij} &\equiv \sum_{k=1}^{T}\sum_{i=1}^{T}\sum_{j=1}^{\gamma} d_{kj}(v_{ij,k} + u_{ij,k}s) \mod (\Phi(x), q) \\
&\equiv \sum_{k=1}^{T}\sum_{i=1}^{T}\sum_{j=1}^{\gamma} d_{kj}v_{ij,k} + \Big(\sum_{k=1}^{T}\sum_{i=1}^{T}\sum_{j=1}^{\gamma} d_{kj}u_{ij,k}\Big)s \mod (\Phi(x), q) \\
&\equiv r_1 + r_0 s \mod (\Phi(x), q)
\end{aligned}$$

□

The above proof is significant in the fact that the user can compute r_0 and r_1 with only the information of a ciphertext and the public key. The flattened vector $F(c_2')$ can be computed given only the CRT representation of c_2' and a desired base, and the matrices U and V are provided in the evaluation key.

4.3 DFT's, NTT's, and FFT's

When working with large polynomials, operations can be slow. In particular, polynomial to polynomial multiplication is very slow. In practice fast Fourier transforms (FFT's) are used for actual polynomial computation, such as ciphertext multiplication in homomorphic encryption. Though we will not discuss actual FFT algorithms, we will lay the groundwork and discuss discrete Fourier transforms (DFT's) and number theoretic transforms (NTT's).

We begin with the DFT as described in [17]. Consider a vector of complex numbers, $a =$

$(a_1, a_2, \ldots, a_n) \in \mathbb{C}^n$. Then, the DFT is computed as $\tilde{a} = \mathrm{DFT}(a) \in \mathbb{C}^n$ where

$$\tilde{a}_i = \sum_{j=0}^{n-1} a_j e^{-ij2\pi\sqrt{-1}/n}$$

for each $i = 0, \ldots, n-1$. Likewise, the IDFT is computed $b = \mathrm{IDFT}(\tilde{a})$ where

$$b_i = \frac{1}{n} \sum_{j=0}^{n-1} \tilde{a}_j e^{ij2\pi\sqrt{-1}/n}$$

With these DFT's, we are computing in the field $\mathbb{C}$ for each entry of $\tilde{a}$ or b. As all of the homomorphic encryption schemes described do computations in the quotient ring $\mathbb{Z}_q$ for our coefficients, we need a version of the DFT and IDFT described in finite spaces. A DFT in a finite field rather than $\mathbb{C}$ is known as a *number theoretic transform (NTT)*. For an element $a \in R_n$, a is simply a polynomial of degree less than or equal to $n-1$ that we can express as a summation

$$a = \sum_{i=0}^{n-1} a_i x^i$$

with each $a_i \in \mathbb{Z}$. Similarly, for $a \in R_{n,q}$, we can express a analogously but with each $a_i \in \mathbb{Z}_q$.

Consider the the coefficients vector of a, which is $a' = (a_0, a_1, \ldots, a_{n-1}) \in \mathbb{Z}_q^n$. For a NTT, we require to use the polynomial $\Phi(x) = x^n + 1$, where n is a power of two, as we have already been using. We also require that $q \equiv 1 \mod 2n$. Let ζ be an nth primitive root of unity so that $\zeta^n \equiv 1 \mod q$. The NTT is computed $\tilde{a} = \mathrm{NTT}(a') \in \mathbb{Z}_q^n$ by computing

$$\tilde{a}_i \equiv \sum_{j=0}^{n-1} a_j \zeta^{ij} \mod q$$

for each $i = 0, \ldots, n-1$. The inverse NTT, denoted INTT, is computed $b = \mathrm{INTT}(\tilde{a})$ where

$$b_i = n^{-1} \sum_{j=0}^{n-1} \tilde{a}_j \zeta^{-ij} \mod q$$

Then, we have that $\mathrm{INTT}(\mathrm{NTT}(a')) = a'$. In practice, these NTT's are computed via a fast Fourier transform (FFT). The most common algorithm to compute FFT's is the Cooley-Turkey algorithm [7]. We will not discuss the actualy algorithm for FFT's in this text, but cite the original Cooley-Turkey paper as a reference for the reader.

4.4 Non-arithmetic Operations

Though we do not formally present them in this thesis, it is worth mentioning that non-arithmetic functions can be implemented with in some homomorphic encryption schemes such as RNS CKKS. These functions rely on implementing the $\text{sgn}(x)$, where $\text{sgn}(x)$ outputs 1 if $x > 0$, -1 if $x < 0$, and 0 if $x = 0$. The comparison function, max function, and ReLU activation function can all be implemented with the sign function.

$$\begin{aligned}
\text{comp}(a, b) &= \frac{1}{2}(\text{sgn}(a - b) + 1) \\
\max(a, b) &= \frac{1}{2}(a + b + (a - b)\text{sgn}(a - b)) \\
\text{ReLU}(x) &= \frac{1}{2}(x + x\text{sgn}(x))
\end{aligned}$$

The $\text{sgn}(x)$ function is implemented in [15] with a failure rate of less than 2^{-15} in RNS CKKS. We do not discuss the actual implementation or techniques used in [15] however, as the process requires many details that are beyond the scope of this thesis.

Chapter 5

Error Bounds

The attractive feature of homomorphic encryption schemes is the ability to apply operations directly to ciphertexts. As apparent in chapter 4, the main concern with computing operations on ciphertexts is error growth. Error growth can determine parameter sizes of the cryptosystem, directly affecting the efficiency of said system. If error terms in ciphertexts become too large, then there is a possibility for incorrect decryption of a ciphertext. Techniques to manage error growth were briefly discussed in chapter 4. However, this discussion did not include a concrete analysis of how exactly error bounds grow. In this chapter, we will fully break down the resulting error bounds from previous chapters, error bound improvements, and error look at improved parameter choices based on these bounds.

5.1 BFV Error

The main focus of our error analysis will be on BFV, including the realinarization process. The structure of BFV generally makes a thorough analysis of error bounds more difficult than BGV or CKKS, which is why we choose to focus on BFV in this text.

5.1.1 BFV Encryption Error

Recall that a BFV Encryption takes the form of

$$b_0 + a_0 s \equiv Dm_0 + e_0 \mod (\Phi(x), q)$$

for some error $e_0 \in R_n$ with $e_0 = eu + e_0' - se_0''$. It is easy to see then that $\|e_0\|_\infty \leq \|eu\|_\infty + \|e_0'\|_\infty + \|se_0''\|_\infty \leq 2\rho\delta_R + \rho$, assuming that $\|e_0'\|_\infty, \|e_0''\|_\infty \leq \rho$ and $\|s\|_\infty = 1$. Here, δ_R is the expansion factor given by

$$\delta_R = \max\left\{\frac{\|x \cdot y\|_\infty}{\|x\|_\infty \cdot \|y\|_\infty} : x, y \in R_n\right\}$$

In our case of using the polynomial $\Phi(x) = x^n + 1$, the expansion factor becomes $\delta_R = n$. Therefore, encryption using BFV would yield an error bound of $\|e_0\|_\infty \leq 2\rho n + \rho$. Notice how in our encryption, the choice of D is not relevant. It will however be relevant when discussing multiplication. But first, we will consider the error bounds given in addition.

5.1.2 BFV Addition Error

As discussed in section 3.3.2, given two BFV ciphertexts $\texttt{ct}_0 = (a_0, b_0)$ and $\texttt{ct}_1 = (a_1, b_1)$, $\texttt{BFV.Add}(\texttt{ct}_0, \texttt{ct}_1)$ returns a BFV ciphertext with error term $e_0 + e_1 - \epsilon tr$. Since $\|\epsilon\|_\infty \leq 1$ and $\|r\|_\infty \leq 1$, it is easy to see that $\|e_0 + e_1 - \epsilon tr\|_\infty \leq 2E + t$ if e_0 and e_1 are both bounded by E.

Now, if we want a better bound, we can put a restriction on t. If we require that $t|(q-1)$, then $D = \lfloor q/t \rfloor = (q-1)/t$. Then, it is easy to see that $Dt \equiv -1 \mod q$. As before, we can write $m_0 + m_1 = [m_0 + m_1]_t + tr$ for some $r \in R_{n,q}$ with $\|r\|_\infty \leq 1$. Then,

$$\begin{aligned}
(b_0 + a_0 s) + (b_1 + a_1 s) &\equiv D(m_0 + m_1) + e_0 + e_1 \mod (\Phi(x), q) && (5.1)\\
&\equiv D[m_0 + m_1]_t + e_0 + e_1 + Dtr \mod (\Phi(x), q) && (5.2)\\
&\equiv D[m_0 + m_1]_t + e_0 + e_1 - r \mod (\Phi(x), q) && (5.3)\\
&\equiv D[m_0 + m_1]_t + e_{\text{add}} \mod (\Phi(x), q) && (5.4)
\end{aligned}$$

where $e_{\text{add}} = e_0 + e_1 - r$ is the error term. So, $\texttt{BFV.Add}(\texttt{ct}_0, \texttt{ct}_1)$ has error term bounded via $\|e_{\text{add}}\|_\infty \leq 2E + 1$. Noting that $t \geq 2$, we can see that this worst-case error bound is indeed better. The main restriction here is of course that we require t to divide $q - 1$. This error bound can easily be extended for any number of additions between ciphertexts. That is, performing ω additions of ciphertexts all bounded by E will result in error less than $(\omega + 1)E + \omega$.

We note choosing $D = q/t$ where t divides q is an even better choice, yielding an error bound of $2E$ for a single addition. This is true, as with basic BFV we do not require t and q to be coprime. However, when moving to more advanced computing techniques as we do in chapter 4, we

will in fact require t and q to be coprime, thus setting $D = (q-1)/t$ with t dividing q is the best we can do.

5.1.3 BFV Multiplication Error

In this section, we will discuss the error bound of BFV multiplication in three parts: error incurred in `BFV.Mul.Int`, error incurred in `Relin.V1`, and error incurred in `Relin.V2`. With `BFV.Mul.Int`, the proof of lemma 3.3.3 gives us a BFV ciphertext with error term

$$e^* = (m_0e_1 + m_1e_0) + t(e_0r_1 + r_0e_1) + r_e + (-r_t(q))(r_m + m_0r_1 + r_0m_1) + r_r - r_a$$

First, recall the following definitions of some of the terms in the above equation

$$\begin{aligned}
r_0 &\quad : \text{element in } R_n \text{ such that } b_0 + a_0s \equiv Dm_0 + e_0 + qr_0 \mod \Phi(x) \\
r_1 &\quad : \text{element in } R_n \text{ such that } b_1 + a_1s \equiv Dm_1 + e_1 + qr_1 \mod \Phi(x) \\
r_e &\quad : \text{element in } R_n \text{ such that } e_0e_1 \equiv [e_0e_1]_D + Dr_e \mod \Phi(x) \\
r_m &\quad : \text{element in } R_n \text{ such that } m_0m_1 \equiv [m_0m_1]_t + tr_m \mod \Phi(x) \\
r_a &\quad : \text{element in } \mathbb{R}[x],\ r_a = (t/q)(b_0 + a_0s)(b_1 + a_1s) - c_0' - c_1's - c_2's^2 \\
r_r &\quad : \text{element in } \mathbb{R}[x],\ r_r = (t/q)[e_0e_1]_D - (r_t(q)/q)(Dm_0m_1 + (m_0e_1 + m_1e_0) + r_e) \\
r_t(q) &\quad : \text{element in } \mathbb{Z},\ r_t(q) = q - tD
\end{aligned}$$

These terms have the following bounds given in [10],

$$\begin{aligned}
\|m_0\|_\infty, \|m_1\|_\infty \le t & \qquad \|e_0\|_\infty, \|e_1\|_\infty \le E \\
\|r_0\|_\infty, \|r_1\|_\infty \le \delta_R \|s\|_\infty & \qquad r_t(q) \le t \\
\|r_e\|_\infty \le E^2\delta_R/D & \qquad \|r_r\|_\infty \le \delta_R(t+1/2)^2 + 1/2 \\
\|r_m\|_\infty \le t\delta_R/4 & \qquad \|r_a\|_\infty \le (\delta_R \|s\|_\infty + 1)^2/2
\end{aligned}$$

We do not provide derivations of these error bounds given in [10], as we will provide our own bounds later in this chapter. By the bounds listed, we have

$$\|e^*\|_\infty \leq \|m_0e_1\|_\infty + \|m_1e_0\|_\infty + t(\|e_0r_1\|_\infty + \|r_0e_1\|_\infty) + \|r_e\|_\infty$$
$$+ r_t(q)(\|(r_m\|_\infty + \|m_0r_1\|_\infty + \|r_0m_1\|_\infty) + \|r_r\|_\infty + \|r_a\|_\infty$$

$$= 2\delta_R tE + 2t\delta_R E\delta_R \|s\|_\infty + E^2\delta_R/D + t^2\delta_R/4 + 2\delta_R t^2\delta_R \|s\|_\infty$$
$$+ \delta_R(t + 1/2)^2 + 1/2 + (\delta_R \|s\|_\infty + 1)^2/2$$

$$= 2\delta_R tE(1 + \delta_R \|s\|_\infty) + 2\delta_R^2 t^2 \|s\|_\infty$$
$$+ E^2\delta_R/D + t^2\delta_R/4 + \delta_R(t + 1/2)^2 + 1/2 + (\delta_R \|s\|_\infty + 1)^2/2$$

$$= 2\delta_R tE(1 + \delta_R \|s\|_\infty + \frac{1}{4t}) + 2\delta_R^2 t^2 \|s\|_\infty$$
$$+ t^2\delta_R/4 + \delta_R t^2 + \delta_R t + \delta_R/4 + 1/2 + \delta_R^2 \|s\|_\infty^2 /2 + \delta_R \|s\|_\infty + 1/2$$

It is easy to see that we have the following inequalities

$$\delta_R^2 \|s\|^2 /2 \leq 2t^2\delta_R^2 \|s\|^2$$
$$\delta_R \|s\| \leq 2t^2\delta_R^2 \|s\|$$
$$t^2\delta_R/4 + t^2\delta_R + t\delta_R + \delta_R/4 + 1 \leq 2t^2\delta_R^2$$

Therefore,

$$\|e^*\|_\infty \leq 2\delta_R tE(1 + \delta_R \|s\|_\infty + \frac{1}{4t}) + 2\delta_R^2 t^2 \|s\|_\infty^2 + 4\delta_R^2 t^2 \|s\|_\infty + 2\delta_R^2 t^2$$
$$= 2\delta_R tE(1 + \delta_R \|s\|_\infty + \frac{1}{4t}) + 2\delta_R^2 t^2(\|s\|_\infty + 1)^2$$
$$\approx 2\delta_R tE(1 + \delta_R \|s\|_\infty) + 2\delta_R^2 t^2(\|s\|_\infty + 1)^2$$

which is the bound obtained in [10]. With the inequality $\|e^*\|_\infty \leq 2\delta_R tE(1 + \delta_R \|s\|_\infty + \frac{1}{4t}) + 2\delta_R^2 t^2(\|s\|_\infty + 1)^2$, the term $\frac{1}{4t}$ is small compared to $1 + \delta_R \|s\|_\infty$, so it can essentially ignored. We

will now rework this to provide better worst case bounds that [10].

A small improvement can be made by assuming that e_0 and e_1 are not bounded by the same thing. Let $\|e_0\|_\infty \leq \rho_0$ and $\|e_1\|_\infty \leq \rho_1$. Then, $\|r_e\|_\infty \leq \delta_R \rho_0 \rho_1 / D$ rather than $\|r_e\|_\infty \leq \delta_R E^2/D$. We'll now look at how bounds of some of the terms in e^* change based on this. First, note that

$$\|m_0 e_1\|_\infty + \|m_1 e_0\|_\infty + t(\|e_0 r_1\|_\infty + \|r_0 e_1\|_\infty) + \|r_e\|_\infty \leq \delta_R t(\rho_0 + \rho_1)(1 + \delta_R \|s\|_\infty) + \delta_R \min\{\rho_0, \rho_1\}/2$$

The term $\delta_R t(\rho_0 + \rho_1)(1 + \delta_R \|s\|_\infty)$ is straightforward as a bound to $\|m_0 e_1\|_\infty + \|m_1 e_0\|_\infty + t(\|e_0 r_1\|_\infty + \|r_0 e_1\|_\infty)$. The term $\delta_R \min\{\rho_0, \rho_1\}/2$ comes from fact that we have both $\rho_0 < D/2$ and $\rho_1 < D/2$. Thus, we have both $\|r_e\|_\infty \leq \delta_R \rho_0/2$ and $\|r_e\|_\infty \leq \delta_R \rho_1/2$ being true. So, we choose the smaller of ρ_0 and ρ_1 in our bound. The remaining terms in e^* remain unchanged by the bound on e_0 and e_1. So,

$$\begin{aligned}
\left\|e^*\right\|_\infty &\leq \delta_R t(\rho_0 + \rho_1)(1 + \delta_R \|s\|_\infty) + \delta_R \min\{\rho_0, \rho_1\}/2 + 2\delta_R^2 t^2 \|s\|_\infty^2 \\
&\quad + t^2 \delta_R/4 + \delta_R t^2 + \delta_R t + \delta_R/4 + 1/2 + \delta_R^2 \|s\|_\infty^2 /2 + \delta_R \|s\|_\infty + 1/2 \\
&= \delta_R t(\rho_0 + \rho_1)(1 + \delta_R \|s\|_\infty) + \delta_R \min\{\rho_0, \rho_1\}/2 + 2\delta_R^2 t^2 \|s\|_\infty^2 \\
&\quad + \delta_R^2 t^2 (\frac{\|s\|_\infty^2}{2t^2}) + \delta_R^2 t^2 (\frac{\|s\|_\infty}{2\delta_R t^2}) + \delta_R^2 t^2 (\frac{5}{4\delta_R} + \frac{1}{4\delta_R t^2} + \frac{1}{\delta_R^2 t^2})
\end{aligned}$$

Assuming that $\|s\|_\infty = 1$, $t \geq 2$, and $\delta_R \geq 2$, it is easy see that

$$\begin{aligned}
&\delta_R^2 t^2 (\frac{\|s\|_\infty^2}{2t^2}) + \delta_R^2 t^2 (\frac{\|s\|_\infty}{2\delta_R t^2}) + \delta_R^2 t^2 (\frac{5}{4\delta_R} + \frac{1}{4\delta_R t^2} + \frac{1}{\delta_R^2 t^2}) \\
&\leq \delta_R^2 t^2 (1/8 + 1/16 + 5/8 + 1/32 + 1/16) \\
&= (29/32)\delta_R^2 t^2 \\
&\leq \delta_R^2 t^2
\end{aligned}$$

Therefore,

$$\left\|e^*\right\|_\infty \leq \delta_R t(\rho_0 + \rho_1)(1 + \delta_R) + \delta_R \min\{\rho_0, \rho_1\}/2 + 3\delta_R^2 t^2 \tag{5.5}$$

Though this is slightly better, assuming different bounds on e_0 and e_1 has little influence.

5.2 The Case when $t = 2$

We'll now look at some more significant improvements, specifically for the case when $t = 2$. In our improvements, the biggest comes in the multiplication error. For this next portion we will let $t = 2, \rho = \mathrm{wt}(s)$, and $\|s\|_\infty = 1$. First, notice that $\|r_0\|_\infty \leq \rho$ and $\|r_1\|_\infty \leq \rho$ since $\mathrm{wt}(s) = \rho$. Consider

$$r_r = (t/q)[e_0e_1]_D - (r_t(q)/q)(Dm_0m_1 + (m_0e_1 + m_1e_0) + r_e)$$

First, notice that $\left\|[e_0e_1]_D\right\|_\infty \leq D\delta_R \leq \frac{q}{t}\delta_R$. Consider:

$$\begin{aligned}
\|r_r\|_\infty &\leq \frac{t}{q}\left\|[e_0e_1]_D\right\|_\infty + \frac{t}{q}\frac{q}{t}\|m_0m_1\|_\infty + \frac{t}{q}\|m_0e_1\|_\infty + \frac{t}{q}\|m_1e_0\|_\infty + \frac{t}{q}\|r_e\|_\infty \\
&\leq \frac{t}{q}\frac{q}{t}\delta_R + \delta_R\frac{t^2}{4} + \frac{t}{q}\delta_R tE + \frac{t}{q}\delta_R\frac{E^2}{D} \\
&\leq \delta_R + \delta_R\frac{t^2}{4} + \frac{t}{2}\delta_R + \delta_R/4
\end{aligned}$$

Since $t = 2$, we have

$$\|r_r\|_\infty \leq \frac{13}{4}\delta_R$$

Now, let's look at $\|r_a\|_\infty$. Recall

$$r_a = (t/q)(b_0 + a_0s)(b_1 + a_1s) - c_0' - c_1's - c_2's^2$$

where $c_0' = \left\lfloor \frac{tb_0b_1}{q} \right\rceil$, $c_1' = \left\lfloor \frac{t(b_0a_1+b_1a_0)}{q} \right\rceil$, and $c_2' = \left\lfloor \frac{ta_0a_1}{q} \right\rceil$. We can write

$$\begin{aligned}
c_0' &= \frac{tb_0b_1}{q} + \epsilon_0 \\
c_1' &= \frac{t(b_0a_1 + b_1a_0)}{q} + \epsilon_1 \\
c_2' &= \frac{ta_0a_1}{q} + \epsilon_2
\end{aligned}$$

with each $\epsilon_i \in R_n$ such that $\|\epsilon_i\|_\infty \leq \frac{1}{2}$. Then,

$$r_a = -\epsilon_0 - \epsilon_1 s - \epsilon_2 s^2$$

Therefore,

$$\|r_a\|_\infty \leq \frac{1}{2} + \frac{\rho}{2} + \frac{\rho^2}{2}$$

Using these new bounds, we obtain the following error bound on e^*.

$$\begin{aligned}\left\|e^*\right\|_\infty &\leq \|m_0 e_1\|_\infty + \|m_1 e_0\|_\infty + t(\|e_0 r_1\|_\infty + \|r_0 e_1\|_\infty) + \|r_e\|_\infty \\ &+ r_t(q)(\left\|(r_m\right\|_\infty + \|m_0 r_1\|_\infty + \|r_0 m_1\|_\infty) + \|r_r\|_\infty + \|r_a\|_\infty \\ &\leq \frac{\delta_R tE}{2} + \frac{\delta_R tE}{2} + t(\delta_R E\rho + \delta_R E\rho) + \frac{E^2\delta_R}{D} \\ &+ \frac{\delta_R}{4} + \frac{t^2\delta_R\rho}{2} + \frac{t^2\delta_R\rho}{2} + \frac{13\delta_R}{4} + \frac{1}{2} + \frac{\rho}{2} + \frac{\rho^2}{2} \\ &= \delta_R tE + 2t\delta_R E\rho + \frac{E^2\delta_R}{D} \\ &+ \frac{\delta_R}{4} + \frac{t^2\delta_R\rho}{2} + \frac{t^2\delta_R\rho}{2} + \frac{13\delta_R}{4} + \frac{1}{2} + \frac{\rho}{2} + \frac{\rho^2}{2} \\ &= 2\delta_R E + 4\delta_R E\rho + \frac{E^2\delta_R}{D} \\ &+ 4\delta_R\rho + \frac{7\delta_R}{2} + \frac{1}{2} + \frac{\rho}{2} + \frac{\rho^2}{2}\end{aligned}$$

Note that as long as $\delta_R \geq \rho \geq 9$ (which is always the case in practice), then

$$\begin{aligned}4\delta_R\rho + \frac{7\delta_R}{2} + \frac{1}{2} + \frac{\rho}{2} + \frac{\rho^2}{2} &= \delta_R\rho(4 + \frac{7}{2\rho} + \frac{1}{2\delta_R\rho} + \frac{1}{2\delta_R} + \frac{\rho}{2\delta_R}) \\ &\leq \delta_R\rho(4 + \frac{7}{18} + \frac{1}{162} + \frac{1}{18} + \frac{1}{2}) \\ &\leq \delta_R\rho(4 + \frac{77}{81}) \\ &\leq 5\delta_R\rho\end{aligned}$$

Then,

$$\begin{aligned}\left\|e^*\right\|_\infty &\leq 2\delta_R E + 4\delta_R E\rho + \frac{E^2\delta_R}{D} + 5\delta_R\rho \\ &\leq 2\delta_R E + 4\delta_R E\rho + \frac{E\delta_R}{2} + 5\delta_R\rho \\ &= E(\delta_R(\frac{5}{2} + 4\rho)) + 5\delta_R\rho\end{aligned}$$

5.2.1 Relinearization Errors

Recall that using `Relin.V1`, the first version of relinearization, we obtain an error of $\mathbf{hw}$, where $\mathbf{h} = (h_0, \ldots, h_{\gamma-1})$ and $\mathbf{w} = (w_0, \ldots, w_{\gamma-1})^T$. Recall too that $\|\mathbf{h}\|_\infty \leq B/2$ and $\|\mathbf{w}\|_\infty \leq \rho$. Therefore, $\|\mathbf{hw}\|_\infty \leq \gamma\delta_R\rho B/2$. Then, the total multiplication error for the case where $t = 2$ is

$$\begin{aligned}\|e_{\text{mult}}\|_\infty &\leq e^* + \mathbf{hw} \\ &\leq E(\delta_R(\frac{5}{2} + 4\rho)) + 5\delta_R\rho + \gamma\delta_R\rho B/2\end{aligned}$$

Noting that $B^\gamma > q$, then $B > q^{1/\gamma}$. So, we set $\gamma = \lceil \log_B(Q) \rceil$. Then, the bound above becomes

$$\|e_{\text{mult}}\|_\infty \leq E(\delta_R(\frac{5}{2} + 4\rho)) + 5\delta_R\rho + \delta_R B\rho\lceil(\log_B(Q))\rceil/2$$

For the second version of relinearization, recall that `Relin.V2` returns an error term of $e'' = \frac{c_2'e'}{p} + \epsilon_0 + \epsilon_1 s$ for some $\epsilon_0, \epsilon_1 \in \mathbb{R}[x]$ with $\|\epsilon_0\|_\infty, \|\epsilon_1\|_\infty \leq 1/2$. Seeing as c_2' is essentially indistinguishable from a random element in $R_{n,q}$, we can at best say $\left\|c_2'\right\|_\infty \leq q/2$. Therefore, we are given the error bound from [10].

$$\begin{aligned}\left\|e''\right\|_\infty &= \left\|\frac{c_2'e'}{p} + \epsilon_2 + \epsilon_1 s\right\|_\infty \\ &\leq \left\|\frac{c_2'e'}{p}\right\|_\infty + \|\epsilon_2\|_\infty + \|\epsilon_1 s\|_\infty \\ &\leq \frac{\left\|c_2'\right\|_\infty \left\|e'\right\|_\infty}{p}\delta_R + \|\epsilon_2\|_\infty + \text{wt}(s)\,\|\epsilon_1\|_\infty \|s\|_\infty \\ &\leq \frac{q\delta_R B_k}{p} + 1/2 + \text{wt}(s)/2\end{aligned}$$

where $B_k = \left\|e'\right\|_\infty$. Note that we do the bound on the norm of e' is different from most other errors.

The reason for this is as we are computing this relinearization in the ring $R_{n,pq}$, a slightly different error distribution must be considered to maintain security standards [10]. If our multiplication error is $e_{\text{mult}} = e^* + e''$, this would give a final mutliplication error bound of

$$\|e_{\text{mult}}\|_\infty \leq E(\delta_R(\frac{5}{2} + 4\rho)) + 5\delta_R\rho + \frac{q\delta_R\rho}{p} + 1/2 + \text{wt}(s)/2$$

5.2.2 Parameter Generation

We will now discuss a way to pick parameters in an effective way for the case when $t = 2$, as this is the case for which we can choose the smallest parameters. Recall too that $n = \delta_R$ and $\rho = \text{wt}(s)$. Let $\mathtt{d_{add}}$ be the additive depth at each modulus level, $\mathtt{d_{mult}}$ is the multiplicative depth of the circuit, and Q an approximation for the desired modulus $Q_{L+1} = q_0 \cdots q_{L+1}$. The following process describes how to pick the moduli q_i, and assumes that `Relin.V1` was the version of relinearization used.

1. Choose parameters: ρ, n, $\mathtt{d^L_{add}}$, $L = \mathtt{d_{mult}}, B, q_0, q_{L+1}, Q$.
2. Set $E = \mathtt{d^L_{add}}(\rho + 2)$, and choose primes $q_1, \ldots, q_L$ such that

$$q_i > \frac{2[E(\delta_R(\frac{5}{2} + 4\rho)) + 5\delta_R\rho + \delta_R B\rho\lceil(\log_B(Q))\rceil/2]}{\rho - 3}$$

3. Check that $\prod q_i = Q_{L+1} \leq Q$. If not, adjust the original parameters and try again.

The technique behind this parameter selection is to select some basic parameters as a start, with the desired number of addition depth per level and multiplication depth. By depth, we mean how many additions and multiplications we wish to perform on ciphertexts. Then, we look at the maximum possible bound at each level for doing up to $\mathtt{d_{add}}$ additions and a single multiplication. If we call that maximum bound E^*, then we know that if $q_i > \frac{2E^*}{\rho-3}$, we can perform modulus reduction and reduce the ciphertext error below ρ by corollary 4.1.1. Therefore, we select q_i such that this is always true. We then do this $\mathtt{d_{mult}}$ times, ensuring we can perform $\mathtt{d_{mult}}$ mutliplications and a subsequent modulus reduction after each multiplication.

The issue with this technique is that the maximum error bound E^* depends on the current modulus level Q_i, which we do not know if we have not yet selected each $q_1, \ldots, q_i$. Therefore, we make some sort of estimate for our biggest modulus Q to use in our computation of each q_i. After

computing these q_i, we must then verify that $\prod_{i=0}^{L+1} q_i = Q_{L+1} \leq Q$. If this is true, this means our maximum error bound E^* computed with Q is greater than or equal to the maximum error bound for each Q_i, which guarantees that each modulus reduction will still return an error less than or equal to ρ.

Example: Choose parameters $\rho = 64, n = 2048, \mathtt{d}^{\mathtt{L}}_{\mathtt{add}} = 8, L = 4, B = 2, Q = 2^{128}, q_0 = 2^{14}, q_{L+1} = 2^{15}$. Then, we have

$$\frac{2[E(\delta_R(\frac{5}{2} + 4\rho)) + 5\delta_R\rho + \delta_R B\rho\lceil(\log_B(Q))\rceil/2]}{\rho - 3} < 2^{24} \tag{5.6}$$

So, we choose primes $q_1 \approx q_2 \approx q_3 \approx 2^{24}$. Now, note that

$$\prod_{i=0}^{L+1} q_i \approx 2^{14} \cdot (2^{24})^4 \cdot 2^{15} = 2^{125} < Q$$

So, these parameters support 8 additions at each level, and 4 levels of multiplication. At level q_0, we only allow for decryption and no operations. In this case, we have $\log_2(Q_L) \approx 2^{125}$.

5.3 The Case when $t|(q-1)$

We now look at the case where $t|(q-1)$ for a general modulus t, rather than $t = 2$ specifically. This is the case in which we really see the improvements in parameter sizes. Again, the main improvements come in the multiplication bound. We will first rework this bound, and then discuss parameters. Recall that contained in e^*, we have the term

$$r_r = (t/q)[e_0e_1]_D - (r_t(q)/q)(Dm_0m_1 + (m_0e_1 + m_1e_0) + r_e)$$

First, note that $r_t(q) = q - tD = q - t((q-1)/t) = 1$. So in this case,

$$r_r = (t/q)[e_0e_1]_D - (1/q)(Dm_0m_1 + (m_0e_1 + m_1e_0) + r_e)$$

Turning to the norm on this, we have

$$\begin{aligned}
\|r_r\|_\infty &\leq (t/q)\left\|[e_0e_1]_D\right\|_\infty + (1/q)D\|m_0m_1\|_\infty + (1/q)(\|m_0e_1\|_\infty + \|m_1e_0\|_\infty) + (1/q)\|r_e\|_\infty \\
&\leq \frac{t}{q}\frac{D}{2}\delta_R + \frac{1}{q}D\delta_R\frac{t^2}{4} + \frac{1}{q}\delta_R tE + \frac{1}{q}\frac{E\delta_R}{2} \\
&\leq \frac{t}{q}\frac{q}{2t}\delta_R + \frac{1}{q}\frac{q}{t}\delta_R\frac{t^2}{4} + \frac{1}{q}\delta_R tE + \frac{1}{q}\frac{E\delta_R}{2} \\
&= \frac{\delta_R}{2} + \frac{\delta_R t}{4} + \frac{\delta_R tE}{q} + \frac{\delta_R E}{2q} \\
&\leq E\delta_R(9/8) + \frac{t\delta_R}{4}
\end{aligned}$$

Now, consider the bound on e^*:

$$\begin{aligned}
\left\|e^*\right\|_\infty &\leq \|m_0e_1\|_\infty + \|m_1e_0\|_\infty + t(\|e_0r_1\|_\infty + \|r_0e_1\|_\infty) + \|r_e\|_\infty \\
&+ r_t(q)(\left\|(r_m\right\|_\infty + \|m_0r_1\|_\infty + \|r_0m_1\|_\infty) + \|r_r\|_\infty + \|r_a\|_\infty \\
&\leq E\delta_R t + 2Et\delta_R^2\|s\|_\infty + \frac{E\delta_R}{2} + \frac{t\delta_R}{4} + t\delta_R^2\|s\|_\infty \\
&+ E\delta_R(9/8) + \frac{t\delta_R}{4} + \frac{1}{2} + \frac{\delta_R}{2}\|s\|_\infty + \frac{\delta_R^2}{2}\|s\|_\infty^2 \\
&\leq E\delta_R t + 2Et\delta_R^2\|s\|_\infty + \frac{E\delta_R}{2} + \frac{t\delta_R}{4} + t\delta_R^2\|s\|_\infty \\
&+ E\delta_R(9/8) + \frac{t\delta_R}{4} + \frac{1}{2} + \frac{\delta_R}{2}\|s\|_\infty + \frac{\delta_R^2}{2}\|s\|_\infty^2 \\
&= E\delta_R t(1 + 2\delta_R\|s\|_\infty + \frac{1}{2t} + \frac{9}{8t}) \\
&+ t\delta_R^2(\frac{1}{4\delta_R} + \|s\|_\infty + \frac{1}{4\delta_R} + \frac{1}{2\delta_R^2 t} + \frac{1}{t\delta_R}\|s\|_\infty + \frac{1}{2t}\|s\|_\infty^2) \\
&= E\delta_R t(1 + 2\delta_R\|s\|_\infty + \frac{1}{2t} + \frac{9}{8t}) \\
&+ t\delta_R^2(\frac{1}{2\delta_R} + \frac{1}{2\delta_R^2 t} + \|s\|_\infty(1 + \frac{1}{t\delta_R}) + \|s\|_\infty^2\frac{1}{2t}) \\
&\leq 2E\delta_R t(1 + \delta_R\|s\|_\infty) + t\delta_R^2(\frac{5}{16} + \frac{5}{4}\|s\|_\infty + \frac{1}{2t}\|s\|_\infty^2)
\end{aligned}$$

We'll compare this final line to this to the earlier bound from [10], which was

$$\left\|e^*\right\|_\infty \leq 2\delta_R tE(1+\delta_R \left\|s\right\|_\infty) + 2\delta_R^2 t^2 (\left\|s\right\|_\infty^2 + 2\left\|s\right\|_\infty + 1)$$

We can see with our new bound, the first term remains the same. However, we reduce the second term by a factor of t, as well as obtain better coefficients. Adding the relinearization error bound from before, this gives a final multiplication error bound

$$\left\|e_{\text{mult}}\right\|_\infty \leq 2E\delta_R t(1+\delta_R \left\|s\right\|_\infty) + t\delta_R^2(\frac{5}{16} + \frac{5}{4}\left\|s\right\|_\infty + \frac{1}{2t}\left\|s\right\|_\infty^2) + \delta_R B\rho\lceil(\log_B(Q))\rceil/2 \tag{5.7}$$

5.3.1 Parameter Generation

In the tables below, we give the results of the best BFV parameters from [8] and compare them to our own parameters using the condition that $t|q_i - 1$ for any modulus q_i. The notation used in [8] is slightly different from ours, but we will use our own notation to keep consistency. The differences in notation are listed below for convenience.

1. [8] uses ζ as the additive depth of the a circuit, whereas we use $\mathtt{d_{add}}$.
2. [8] assumes circuits of multiplicative depth $L-1$, and we use L or $\mathtt{d_{mult}}$ as the depth.
3. [8] uses h as the hamming weight, for which we use $\text{wt}(s)$.
4. [8] uses $\phi(m)$ as the euler phi function of m for the mth cyclotomic polynomial, which we also use here as $\phi(m) = n$.
5. [8] uses $q_{L-1} = p_0 \cdots p_{L-1}$ as the fresh ciphertext modulus obtained immediately after encrypting and reducing the modulus, whereas we use $Q_L = q_0 \cdots q_L$.
6. [8] uses p as the plaintext modulus, whereas we use t.

Table from [8] with $L = 1$.

$t \approx$	$\phi(m) \approx$	$\log_2 q_0 \approx$	$\log_2 q_i \approx$	$\log_2 q_L \approx$	$\log_2 Q_L \approx$
2	610	14	-	21	35
101	884	20	-	30	50
2^{32}	2714	46	-	104	150
2^{64}	5091	78	-	202	280
2^{128}	9847	143	-	397	540
2^{256}	19176	271	-	779	1050

Using our condition with $L = 1$.

$t \approx$	$\phi(m) \approx$	$\log_2 q_0 \approx$	$\log_2 q_i \approx$	$\log_2 q_L \approx$	$\log_2 Q_L \approx$
2	610	12	-	25	37
101	884	18	-	31	49
2^{32}	2714	45	-	59	84
2^{64}	5091	78	-	93	171
2^{128}	9847	143	-	160	303
2^{256}	19176	272	-	290	562

Table from [8] with $L = 4$.

$t \approx$	$\phi(m) \approx$	$\log_2 q_0 \approx$	$\log_2 q_i \approx$	$\log_2 q_L \approx$	$\log_2 Q_L \approx$
2	1616	14	18	22	90
101	2439	22	28	29	135
2^{32}	8567	48	105	106	470
2^{64}	16158	79	201	202	885
2^{128}	31431	144	394	394	1720
2^{256}	61886	273	778	778	3385

Using our condition with $L = 4$.

$t \approx$	$\phi(m) \approx$	$\log_2 q_0 \approx$	$\log_2 q_i \approx$	$\log_2 q_L \approx$	$\log_2 Q_L \approx$
2	1616	12	25	25	112
101	2439	19	35	35	159
2^{32}	8567	47	64	64	303
2^{64}	16158	79	98	98	471
2^{128}	31431	144	164	164	800
2^{256}	61886	273	294	294	1449

Table from [8] with $L = 9$.

$t \approx$	$\phi(m) \approx$	$\log_2 q_0 \approx$	$\log_2 q_i \approx$	$\log_2 q_L \approx$	$\log_2 Q_L \approx$
2	3354	16	18	23	185
101	5183	23	29	30	285
2^{32}	18170	48	105	106	995
2^{64}	34723	82	202	202	1900
2^{128}	67465	144	394	394	3690
2^{256}	133223	274	779	779	7285

Using our condition with $L = 9$.

$t \approx$	$\phi(m) \approx$	$\log_2 q_0 \approx$	$\log_2 q_i \approx$	$\log_2 q_L \approx$	$\log_2 Q_L \approx$
2	3354	14	30	30	284
101	5183	20	37	37	353
2^{32}	18170	48	66	66	642
2^{64}	34723	81	100	100	981
2^{128}	67465	146	166	166	1640
2^{256}	133223	275	296	296	2939

These tables above are the results from [8] compared to the results using our condition that $\|s\|_\infty = 1$. The values of $\phi(m)$ are purely for comparison, as in practice we do in fact also need to use the polynomial $\Phi(x) = x^n + 1$ in computation for our condition, which means $\phi(m)$ is a power of 2. In this case $\delta_R = \phi(m) = n$. Recall that for a chosen constant C, algorithm 4.1 will take in a

ciphertext at level i and will return a ciphertext at level $i-1$ with error bound C if

$$q_i > \frac{2E}{2C - 2 - \delta_R \|s\|_\infty}$$

where E is the current ciphertext error bound. In our case, we choose $C = \delta_R \|s\|_\infty$ so that we require

$$q_i > \frac{2E}{\delta_R \|s\|_\infty - 2}$$

The parameters above were achieved by finding the smallest q_i that satisfies this condition for the chosen parameters. In particular, we choose q_i so that

$$q_i > \frac{2[2E_{\text{add}}\delta_R t(1 + \delta_R \|s\|_\infty) + t\delta_R^2(\frac{5}{16} + \frac{5}{4}\|s\|_\infty + \frac{1}{2t}\|s\|_\infty^2) + \delta_R B\rho\lceil(\log_B(Q))\rceil/2]}{\delta_R \|s\|_\infty - 2}$$

where $E_{\text{add}} = (\mathtt{d_{add}} + 1)\delta_R \|s\|_\infty + \mathtt{d_{add}}$ and Q is an approximation of Q_ℓ. Since $C = \delta_R \|s\|_\infty$, we can always assume a ciphertext immediately after modulus reduction has error bounded by $\delta_R \|s\|_\infty$. Then, E_{add} acts as an upper error bound to the ciphertext obtained after performing $\mathtt{d_{add}}$ additions. The numerator in the above equation then acts as the upper bound to multiplying two ciphertexts with errors both bounded by E_{add}. In other words, this above inequality guarantees that if we perform no more than $\mathtt{d_{add}}$ additions followed by a single multiplication, we can reduce our modulus by q_i and always obtain a new ciphertext bounded by $\delta_R \|s\|_\infty$. The circuit is then constructed using our q_i's to obtain the desired multiplication depth, as we described in section 5.2.2.

It is worth noting that in order for these parameters to hold, we require q_i to be a Proth prime since $t|(q_i - 1)$. A Proth prime is a prime number of the form $k2^N + 1$ with $2^N > k$. In our specific case, we are looking for each q_i to be a Proth prime such that $N \geq \log(t)$.

5.4 CKKS Error

Recall that a CKKS ciphertext takes the form $(a_0, b_0) \in R_{n,q}^2$ such that

$$b_0 + a_0 s \equiv m_0 + e_0 \mod (\Phi(x), q) \tag{5.8}$$

Let $m_0' = m_0 + e_0$. Recall that for some complex vector $z_0 \in \mathbb{C}^{n/2}$ and positive Δ, encryption is done first by computing $m_0 = \mathtt{Ecd}(z_0, \Delta)$, and then proceeding with the remainder of algorithm 3.17 to

get the ciphertext $(a_0, b_0) \in R_{n,q}^2$. In this case, $\texttt{Ecd}(z_0, \Delta) = \lfloor \tau^{-1}(\Delta z) \rceil$ for the canonical embedding map $\tau : \mathbb{R}[x]/(\Phi(x)) \to \mathbb{C}^{n/2}$. Decryption then is simply done by computing $z_0' = \texttt{Dcd}(m_0', \Delta)$, where $\texttt{Dcd}(m_0', \Delta) = \Delta^{-1}\tau(m_0')$. In other words, we decrypt first by reducing $b_0 + a_0 s$ modulo $(\Phi(x), q)$ to obtain m_0', and then decode m_0'. For a full discussion of CKKS, see section 3.6.

Given a vector $z_0 \in \mathbb{C}_{n/2}$, and a vector z_0' that is obtained from encrypting and decrypting z_0, the natural question is to ask how close z_0' is to z_0. That is, what conditions must be met to ensure that $z_0' = z_0$ to some desired accuracy. In computing z_0' from z_0, there are two sources of error to consider: error introduced when computing $m_0 = \texttt{Ecd}(z, \Delta)$, and error when decrypting a ciphertext $\texttt{Dcd}(m_0 + e_0, \Delta)$. To better analyze this, we will consider two questions: how close is z_0 to $\texttt{Dcd}((\texttt{Ecd}(z_0, \Delta), \Delta)$, and how close is $\texttt{Dcd}(m_0, \Delta)$ to $\texttt{Dcd}(m_0 + e_0, \Delta)$.

Let $r \in \mathbb{C}$, and let $\alpha(x) \in \mathbb{R}[x]/(\Phi(x))$ such that $\alpha(\zeta) = r$ for some primitive root ζ. That is, we can write r as

$$\alpha_0 + \alpha_1\zeta + \cdots + \alpha_{n-1}\zeta^{n-1} = r$$

Now, let $\beta(x) = \lfloor \alpha(x) \rceil \in R_n$. Then, there is some r' where

$$\beta_0 + \beta_1\zeta + \cdots + \beta_{n-1}\zeta^{n-1} = r'$$

Since $|\zeta^d| = 1$ for any $d \in \mathbb{N}$ and $|\alpha_i - \beta_i| \le 1/2$ for all $i \in [n-1]$, we have

$$\begin{aligned}
|(r - r')| &= |(\alpha_0 - \beta_0) + (\alpha_1 - \beta_1)\zeta + \cdots + (\alpha_{n-1} - \beta_{n-1})\zeta^{n-1}| \\
&\le |(\alpha_0 - \beta_0)| + |(\alpha_1 - \beta_1)||\zeta| + \cdots + |(\alpha_{n-1} - \beta_{n-1})||\zeta^{n-1}| \\
&\le n/2
\end{aligned}$$

Now, since this is true for any r, r' defined this way, we can easily extend this to a vector in $\mathbb{C}^{n/2}$ as we need, as in CKKS we simply have a vector for which each complex component is written as a polynomial evaluation of primitive roots. Furthermore, replacing r and r' with Δz_i and $\Delta z_i'$, we can see that $|(z_i - z_i')| \le \Delta^{-1}n/2$. Therefore, letting $z = (z_1, \ldots, z_{n/2}) \in \mathbb{C}^{n/2}$ we have that

$$\left\| z - \texttt{Dcd}(\texttt{Ecd}(z, \Delta), \Delta) \right\|_\infty \le \Delta^{-1}n/2$$

Here we note that the infinity norm of a complex vector means the maximum infinity norm of each

coefficient. For a real number $a \in \mathbb{R}$, we say a has precision s base D if

$$|a| = \sum_{j \geq 0} D^j b_j + \sum_{i=1}^{s} D^{-i} a_i$$

for $0 \leq a_i, b_j < D$. In other words, the floating point representation of a is accurate to s decimal places base D. If we desire a precision of s base D, we then require that

$$\left\| z - \texttt{Dcd}(\texttt{Ecd}(z, \Delta), \Delta) \right\|_\infty \leq \Delta^{-1} n/2 < D^{-s-1}$$

Recall that $\|\cdot\|_\infty^{\texttt{can}}$ denotes the canonical embedding norm. That is, for $a \in R_n$, $\|a\|_\infty^{\texttt{can}} = \left\| \tau(a) \right\|_\infty$. According to [6], the canonical embedding norm satisfies the following properties

1. For all $a, b \in R_n$, $\|ab\|_\infty^{\texttt{can}} \leq \|a\|_\infty^{\texttt{can}} \|b\|_\infty^{\texttt{can}}$.
2. For all $a \in R_n$, $\|a\|_\infty^{\texttt{can}} \leq \|a\|_1$.
3. There is a constant c_m depending only on m such that $\|a\|_\infty \leq c_m \|a\|_\infty^{\texttt{can}}$, where m is given by the polynomial $\Phi_m(x)$.

We consider the polynomial $m_0' = m_0 + e_0$. Let s be the desired precision in base D of $z_0' = \texttt{Dcd}(m_0', \Delta)$. Then, we require that $\Delta^{-1} \|e_0\|_1 < D^{-s-1}$. This comes from the difference in the norm of our messages. That is, if $\Delta^{-1} \|e_0\|_1 < D^{-s-1}$ we claim

$$\left\| \texttt{Dcd}(m_0, \Delta) - \texttt{Dcd}(m_0', \Delta) \right\|_\infty < D^{-s-1}$$

It is easy to see that

$$\left\| \texttt{Dcd}(m_0, \Delta) - \texttt{Dcd}(m_0', \Delta) \right\|_\infty = \left\| \Delta^{-1} m_0 - \Delta^{-1} m_0' \right\|_\infty^{\texttt{can}}$$

as $\texttt{Dcd}(m_0, \Delta)$ can be thought of as the canonical embedding of $\Delta^{-1} m_0$. Then, we have that

$$\left\| \Delta^{-1} m_0 - \Delta^{-1} m_0' \right\|_\infty^{\texttt{can}} = \left\| \Delta^{-1} e_0 \right\|_\infty^{\texttt{can}} \leq \Delta^{-1} \|e_0\|_1 < D^{-s-1}$$

And so

$$\left\| \texttt{Dcd}(m_0, \Delta) - \texttt{Dcd}(m_0', \Delta) \right\|_\infty < D^{-s-1}$$

From this, we can see that the error has potential to present in the $s+2$ decimal position for each floating point decimal, which could at most affect the $s+1$ decimal position. Thus, $z_i' = z_i$ to s decimal places.

In practice, as e_0 is an unknown polynomial, we have no way of computing $\|e_0\|_1$ without knowing e_0 explicitly. However, we do have estimates for $\|e_0\|_\infty$. Noting that $\|e_0\|_1 \leq n\|e_0\|_\infty$, ensuring that $n\|e_0\|_\infty < D^{-s-1}$ will also ensure the bound above holds.

Now, we consider the case where we have both encoding error and encryption error. Let $z \in \mathbb{C}^{n/2}$, $m_0 = \texttt{Ecd}(z, \Delta)$, and $m_0' = m_0 + e_0$ for some error e_0. Then, we are concerned with a bound on

$$\left\| z - \texttt{Dcd}(m_0', \Delta) \right\|_\infty$$

Consider

$$\begin{aligned}
\left\| z - \texttt{Dcd}(m_0', \Delta) \right\|_\infty &= \left\| z - \texttt{Dcd}(m_0, \Delta) + \texttt{Dcd}(m_0, \Delta) - \texttt{Dcd}(m_0', \Delta) \right\|_\infty \\
&\leq \left\| z - \texttt{Dcd}(m_0, \Delta) \right\|_\infty + \left\| \texttt{Dcd}(m_0, \Delta) - \texttt{Dcd}(m_0', \Delta) \right\|_\infty \\
&\leq \Delta^{-1}(n/2 + \|e_0\|_1) \\
&\leq \Delta^{-1} n(1/2 + \|e_0\|_\infty)
\end{aligned}$$

Thus, requiring that $\Delta^{-1}n(1/2 + \|e_0\|_\infty) < D^{-s-1}$ will ensure accuracy to s positions.

Appendices

Appendix A   Lattice Reduction Algorithms

In this appendix, we present some known results regarding work towards solution to hard lattice problems. The algorithms presented here involve taking an arbitrary basis of a lattice, and reducing it to another basis with better problems. In turn, this makes hard lattice problems easier to manage. We begin with the case of dimension 2. Let Λ be a lattice of dimension 2. We can then define a reduced basis via the following definition.

Definition A.1 *A basis (v_1, v_2) of Λ with $\|v_1\|_2 \leq \|v_2\|_2$ is called reduced if*

$$\frac{|\langle v_1, v_2 \rangle|}{\|v_1\|_2^2} \leq \frac{1}{2}$$

We call $\mu = \frac{\langle v_1, v_2 \rangle}{\|v_1\|_2^2}$ the (orthogonal) projection coefficient.

Note that the norm $\|\cdot\|_2$ used above is the 2-norm. We also use the word size to refer to the norm of a vector. Using this reduced basis, we can say a number of facts about the SVP in a lattice. For instance, we can bound this size of the shortest vector in a lattice with the following theorem.

Theorem A.1 *Give a two dimension lattice Λ of rank 2, if λ is the shortest vector in Λ, then*

$$\|\lambda\|_2 \leq \sqrt{\frac{2}{\sqrt{3}} \det(\Lambda)}$$

We do not include the proof of this theorem in this text, but one can be found in [9]. A reduced basis does not only give us a proof for the inequality above, but also a solution to the SVP.

Theorem A.2 *Let (v_1, v_2) be a reduced basis. Then, v_1 is a shortest vector in Λ.*

Proof. Let (v_1, v_2) be a reduced basis and v a shortest vector in Λ. Then, there are $a, b \in \mathbb{Z}$ such that

$$v = av_1 + bv_2$$

Therefore,

$$\begin{aligned}
\|v\|_2^2 &= \|av_1 + bv_2\|_2^2 \\
&= a^2 \|v_1\|_2^2 + 2ab\langle v_1, v_2\rangle + b^2 \|v_2\|_2^2 \\
&\geq a^2 \|v_1\|_2^2 - 2|ab||\langle v_1, v_2\rangle| + b^2 \|v_2\|_2^2 \\
&\geq a^2 \|v_1\|_2^2 - |ab| \|v_1\|_2^2 + b^2 \|v_1\|_2^2 \\
&= (a^2 - |a||b| + b^2) \|v_1\|_2^2
\end{aligned}$$

Now consider the coefficient $(a^2 - |a||b| + b^2)$. Let $\alpha = |a|$ and $\beta = |b|$. Then,

$$(\alpha^2 - \alpha\beta + \beta^2) = (\alpha - \frac{1}{2}\beta)^2 + \frac{3}{4}\beta^2 = \frac{3}{4}\alpha^2 + (\frac{1}{2}\alpha - \beta)^2$$

which is only 0 if $a = b = 0$. However, $v = av_1 + bv_2$ is a nonzero vector, so at least one of a or b is a nonzero integer. Therefore, we have $(a^2 - |a||b| + b^2) \geq 1$ and so

$$\|v\|_2 \geq \|v_1\|_2$$

Hence, v_1 is a shortest vector. □

This theorem makes the goal of solving the SVP more approachable. If we can find a reduced basis in Λ, then we have a shortest vector. This leads us to the following algorithm, known as Gaussian Lattice Reduction.

	$\texttt{GLR}(v_1, v_2)$
Input:	(v_1, v_2), a basis for Λ.
Output:	v_1, a shortest vector in Λ.
Step 1.	Compute $v_2' = v_2 - \lfloor \frac{\langle v_1, v_2\rangle}{\|v_1\|^2{}_2} \rceil v_1$
Step 2.	If $\|v_1\|_2 > \|v_2'\|_2$, set $v_2 := v_2'$, then swap v_1 and v_2 and go back to step 1. Else, continue.
Step 3.	Return v_1.

Algorithm 1: Gaussian Lattice Reduction Algorithm

Note that $\lfloor \cdot \rceil$ denotes rounding to the nearest integer. The final basis in the algorithm above is actually a reduced basis, and so the vector v_1 that is returned is a solution to the SVP. Notice

that $\|v_1\|_2 \leq \|v_2'\|_2$, which means $\left\lfloor \frac{\langle v_1, v_2 \rangle}{\|v_1\|_2^2} \right\rceil = 0$. So, $v_2' = v_2$ and

$$\frac{|\langle v_1, v_2' \rangle|}{\|v_1\|_2^2} \leq \frac{1}{2}$$

Therefore, (v_1, v_2') is a reduced basis. It is worth noting too that the Gaussian lattice reduction algorithm essentially uses a modified Gram-Schmidt process. As $\frac{\langle v_1, v_2 \rangle}{\|v_1\|^2}$ is generally not in $\mathbb{Z}$, we instead take its closer integer in the orthogonalization process, swap our vectors, and repeat until we have a shortest vector.

Now, consider a lattice Λ of dimension n rather that 2. The Lenstra–Lenstra–Lovász (LLL) Algorithm is an algorithm that uses similar techniques to Gaussian lattice reduction, only in higher dimensions. Let $\mu_{k,j} = \langle v_j^*, v_k \rangle / \|v_j^*\|_2^2$. Then, the algorithm is as follows.

	$\texttt{LLL}(B, B^*, \delta)$
Input:	$B = (v_1, \ldots, v_n)$, a basis for Λ,
	$B^* = (v_1^*, \ldots, v_n^*)$, the Gram-Schmidt basis for B,
	$\delta \in (-0.25, 1]$ a parameter (usually $\delta = 3/4$).
Output:	$B^{\text{LLL}} = (v_1, \ldots, v_n)$, an LLL basis for Λ.
Step 1.	Set $k = 2$ and $v_1^* = v_1$.
Step 2.	While $k \leq n$, for $j = k-1, \ldots, 2, 1$, set $v_k := v_k - \lfloor \mu_{k,j} \rceil v_j$, updating B^* and $\mu_{k,j}$ as needed. If $\|v_k^*\|_2^2 \geq (\delta - \mu_{k,k-1}^2) \|v_{k-1}^*\|_2^2$, set $k := k+1$. Else, swap v_k and v_{k-1}, update B^* and $\mu_{k,j}$, and set $k := \max(k-1, 2)$.
Step 3.	Return $B^{\text{LLL}} = (v_1, \ldots, v_n)$.

Algorithm 2: The LLL Algorithm

The LLL algorithm usually returns a much better basis for Λ than the original basis given. Let $B^{\text{LLL}} = (v_1, \ldots, v_n)$ be the basis obtained from the LLL algorithm, which we call the LLL reduced basis, and $B^* = (v_1^*, \ldots, v_n^*)$ the corresponding Gram-Schmidt basis. Then for all $1 \leq j < i \leq n$, the LLL reduced basis elements satisfy the following

$$|\mu_{i,j}| = \frac{|\langle v_j^*, v_i \rangle|}{\|v_j^*\|_2^2} \leq \frac{1}{2} \qquad \text{(Size Condition)}$$

$$\|v_i^*\|_2^2 \geq (\delta - \mu_{i,i-1}) \|v_{i-1}^*\|_2^2 \qquad \text{(Lovász Condition)}$$

We do not provide the proofs of the size and Lovász conditions, as they are fairly apparent from the construction of B^{LLL}. Just as the Gaussian lattice reduction algorithm, the consistent swapping of basis vectors based on size allows us to keep shortening vectors until $\lfloor \mu_{i,j} \rceil = 0$, which is equivalent to the size condition. The Lovász condition follows from step 2 of the algorithm.

What is significant about the LLL algorithm is it is a lattice reduction algorithm that runs in polynomial time. More precisely, let $L = \max\{\|v_i\|_2\}$. Then, the LLL algorithm executes the k-loop in time at most $\mathcal{O}(n^2 \log n + n^2 logL)$, which is by far the most costly portion of the algorithm.

Though this algorithm works, one may be presented with challenges when attempting to implement it. Most notably, computation of B^* and the $\mu_{k,j}$'s during the algorithm can be costly if done naively. This algorithm is also generally run on huge lattices, which means exact integer computations is not feasible and floating point calculations may be needed, leading to possibilities of error-round offs when computing $\lfloor \mu_{k,j} \rceil$.

Bibliography

[1] Matthew Baker. Algebraic number theory. Course Notes Math 8803, Georgia Tech, 2006. http://www.math.toronto.edu/~ila/ANTBook.pdf.

[2] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. Textbook, 2020. http://toc.cryptobook.us/book.pdf.

[3] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. Cryptology ePrint Archive, Report 2011/277, 2011. https://ia.cr/2011/277.

[4] Benjamin Case. Homomorphic encryption and cryptanalysis of lattice cryptography. All Dissertations. 2635., 2020. https://tigerprints.clemson.edu/all_dissertations/2635.

[5] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full rns variant of approximate homomorphic encryption. Cryptology ePrint Archive, Report 2018/931, 2018. https://ia.cr/2018/931.

[6] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. Cryptology ePrint Archive, Report 2016/421, 2016. https://ia.cr/2016/421.

[7] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[8] Anamaria Costache and Nigel P. Smart. Which ring based somewhat homomorphic encryption scheme is best? Cryptology ePrint Archive, Report 2015/889, 2015. https://ia.cr/2015/889.

[9] Xinyue Deng. An introduction to lenstra-lenstra-lovasz lattice basis reduction algorithm. Massachusetts Institute of Technology, 2016. https://math.mit.edu/~apost/courses/18.204-2016/18.204_Xinyue_Deng_final_paper.pdf.

[10] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. https://ia.cr/2012/144.

[11] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[12] Shai Halevi, Yuriy Polyakov, and Victor Shoup. An improved rns variant of the bfv homomorphic encryption scheme. Cryptology ePrint Archive, Report 2018/117, 2018. https://ia.cr/2018/117.

[13] M.T. Heath. *Scientific Computing: An Introductory Survey*. McGraw-Hill Education, 2005.

[14] Kenneth F. Ireland. *A classical introduction to modern number theory / Kenneth Ireland, Michael, Michael Rosen*. Graduate texts in mathematics ; 84. Springer, New York, 1982.

[15] Eunsang Lee, Joon-Woo Lee, Young-Sik Kim, and Jong-Seon No. Optimization of homomorphic comparison algorithm on rns-ckks scheme. Cryptology ePrint Archive, Report 2021/1215, 2021. https://ia.cr/2021/1215.

[16] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), sep 2009.

[17] S. Roberts. Lecture 7-the discrete fourier transform. Textbook, 2020. http://www.robots.ox.ac.uk/~sjrob/Teaching/SP/l7.pdf.

[18] Voyko Flores Rocha and Julio López. An overview on homomorphic encryption algorithms. 2019.