A Summary of the FV Homomorphic Encryption Scheme and the Average-Case Noise Growth

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Mathematics and Statistics
University of Saskatchewan
Saskatoon

By
Derek Perrin

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Mathematics and Statistics
142 McLean Hall
106 Wiggins Road
University of Saskatchewan
Saskatoon, Saskatchewan S7N 5E6
Canada

Or

Dean
College of Graduate and Postdoctoral Studies
University of Saskatchewan
116 Thorvaldson Building, 110 Science Place
Saskatoon, Saskatchewan S7N 5C9
Canada

Abstract

Homomorphic encryption is a method of encryption that allows for secure computation of data. Many industries are moving away from owning expensive high-powered computers and instead delegating costly computations to the cloud. In an age of data breaches, there is an inherent risk when putting sensitive data on the cloud. Homomorphic encryption allows one to securely perform computations on the cloud without allowing the host or any other party access to the raw data itself. One application being explored is encrypting health data on low-powered embedded devices, uploading it to a cloud application, performing computations to assess health risks, and send the results back to the user's device for decryption and interpretation. Another application being explored is digital voting.

This thesis aims to provide a summary of the current state-of-the-art of homomorphic encryption. We will begin by providing the reader with sources for the current main implementations and schemes they are based on. We will then present the mathematical background used in existing schemes. This includes a background on lattices, cyclotomic fields, rings of integers, and the underlying believed-to-be-hard problems existing schemes take advantage of. We will then shift our attention to the FV scheme which is based on the ring-LWE problem and is one of the main schemes used today. We will then briefly discuss some optimizations used in FV implementations. Finally, we will look at some probabilistic experiments which suggest the noise growth in FV is significantly lower than the theoretical maximum in the average case, and will explore some of the benefits that can be gained.

Acknowledgements

First and foremost, I am very thankful for my supervisor Dr. Cameron Franc. He provided me with my first research experience, inspired me to pursue graduate studies, and also provided a ton of support and guidance throughout this journey. I am always impressed with his knowledge and am forever grateful for his willingness and patience to teach me all of the background required, offer reading courses and discussion groups, answer my questions, and work with me as an equal. This thesis would not have been possible without him.

I would like to thank Dr. Jenna Rajchgot and Dr. Steven Rayan for their support while taking courses with them, and their outstanding work behind the scenes in preparing those courses. I gained a deep appreciation and interest in algebra and algebraic geometry as a result.

I would like to thank fellow graduate students Sheldon Miller, Gagandeep Virk, Christopher Mahadeo, Mahmood Tarayrah, Jarrod Pas, Jason Goertzen, and Seth Dueck, as well as my dear friend Dr. Ashton Reimer for all of their support, encouragement, and providing a positive work environment during my studies.

Finally, I would like to thank all of the developers of the open-source software my research relied upon. I relied on SageMath, Numpy, SciPy, SEAL, and Lattigo.

Dedicated to: my loving partner, Karen; my daughter, Ayla; and my parents, My and Valinda.

Contents

List of Tables

List of Figures

List of Abbreviations

HE	Homomorphic Encryption
SHE	Somewhat Homomorphic Encryption
FHE	Fully Homomorphic Encryption
LWE	Learning with Errors
RLWE	Ring Learning with Errors
SVP	Shortest Vector Problem
CVP	Closest Vector Problem
BDD	Bounded Distance Decoding
DFT	Discrete Fourier Transform
FFT	Fast Fourier Transform
NTT	Number Theoretic Transform
RNS	Residue Number System
CRT	Chinese Remainder Theorem
KS	Kolmogorov-Smirnov
UFD	Unique Factorization Domain
CRT	Chinese Remainder Theorem
SIMD	Single Instruction Multiple Data

1 Introduction

We assume standard undergrad material and terminology. See [DF04] for a comprehensive background in algebra, and [HPS08] for an introductory background in mathematical cryptography.

Cryptography is the art of concealing communications such that only intended recipients can read the underlying message. To anyone eavesdropping on the conversation, the encrypted message, called a ciphertext, appears to be nonsense. In order to decrypt the ciphertext, a secret decryption key is required. Modern asymmetric cryptography schemes are constructed such that their security relies on the assumption of some underlying mathematical problem being difficult to solve. The RSA scheme [RSA78] relies on the assumption that factoring is difficult, and the Diffie-Hellman key exchange [DH76] on the assumption that solving the discrete log problem is difficult.

In 1978 [RAD+78], Rivest et al. proposed the idea of performing computations on ciphertexts without requiring the decryption key.

Definition 1.1. For some encryption scheme $\mathcal{E}$, let $\mathcal{M}, \mathcal{C}$ be rings known as the plaintext and ciphertext spaces respectively, along with a set $\mathcal{K}$ referred to as the key space. Then if we have encryption and decryption functions

$$e_k : \mathcal{M} \to \mathcal{C}$$
$$d_k : \mathcal{C} \to \mathcal{M}$$

both ring homomorphisms for every $k \in \mathcal{K}$, we call $\mathcal{E}$ a *fully homomorphic encryption (FHE) scheme*. If there is a bound on the number of additions and multiplications which can be performed in $\mathcal{C}$ such that decryption yields an incorrect result if that bound is exceeded, we call $\mathcal{E}$ a *somewhat homomorphic encryption (SHE) scheme*. An SHE scheme is often referred to as a *levelled homomorphic encryption scheme* in the literature.

There exist cryptosystems which have homomorphic properties with respect to a single operation, rather than both ring operations. Some literature may refer to these cryptosystems as being somewhat homomorphic. To avoid confusion, we adopt no such convention here and SHE will always mean as in definition 1.1.

Example 1.2 (ElGamal [ElG85])**.** Let $G = \langle g \rangle$ be a cyclic group of prime order p. Alice chooses a private key $a \in \{1, 2, \ldots, p-1\}$, and computes $A = g^a$. Alice's public key is (g, p, A). Bob wishes to encrypt messages $m_1, m_2 \in G$. He chooses two ephemeral keys $k_1, k_2 \in \mathbb{Z}$ and computes the corresponding ciphertexts $c_1 = (g^{k_1}, m_1 \cdot A^{k_1}), c_2 = (g^{k_2}, m_2 \cdot A^{k_2})$ and sends their coordinate-wise product $(g^{k_1+k_2}, m_1 \cdot m_2 \cdot A^{k_1+k_2})$ to Alice. Alice is able to decrypt by computing

$$\begin{aligned}(g^{k_1+k_2})^{-a} \cdot m_1 \cdot m_2 \cdot A^{k_1+k_2} &= (g^{k_1+k_2})^{-a} \cdot m_1 \cdot m_2 \cdot (g^a)^{k_1+k_2} \\ &= m_1 \cdot m_2.\end{aligned}$$

We see that the ElGamal scheme is multiplicatively homomorphic.

For 30 years, there was a search for an FHE scheme. In 2009, Gentry introduced the first FHE scheme in their groundbreaking thesis [Gen09]. They introduce the concept of bootstrapping, and show that any SHE scheme which has a decryption circuit of low complexity and can evaluate its own decryption circuit is bootstrappable to an FHE scheme. This relies on a circular security assumption in which an attacker gains no advantage if they have access to an encrypted key that is encrypted under itself. In section 2 we will introduce the concept of noise and see how it can lead to incorrect decryption when too much is accumulated. The bootstrapping procedure evaluates the decryption circuit homomorphically, and therefore "resets" the noise back down to some base level. The result is that we can perform arbitrarily many operations without requiring decryption. In this thesis, we will look at the scheme described in [FV12], some optimizations which are commonly used, and examine ways to increase the depth of a circuit before decryption or bootstrapping is required.

Table 1.1: Homomorphic Encryption Implementations

Library name	Languages	Supported schemes
HElib	C++	BGV, CKKS
SEAL	C++, .NET	BFV, CKKS
Palisade	C++	BFV, BGV, CKKS, TFHE
Lattigo	Go	BFV, CKKS
CuHE	CUDA	LTV
CuFHE	CUDA	TFHE

1.1 Literature

There are several standard sources used for the research area of homomorphic encryption we will focus on. When focusing on the hard problems that homomorphic encryption schemes are based off of, we primarily look at the ones based on learning with errors and ring learning with errors problems [Reg05, LPR10] which we will discuss in chapter 2. Most implementations are in the ring learning with errors setting. There are two primary schemes used in HE libraries [BGV14, FV12] and they perform exact integer arithmetic. In addition, there is a scheme for performing arithmetic on approximate numbers [CKKS17]. There are also more recent works [BLLN13, CGGI19] which perform integer arithmetic, with the latter work focusing on a fast bootstrapping operation and bootstrapping after every operation. In this thesis, we focus on the scheme introduced in [FV12] and discuss a variant of it in [BEHZ17] of which Microsoft has implemented and made open source. There is a homomorphic encryption standards organization trying to standardize homomorphic encryption; they provide a list of libraries implementing various schemes as well as papers [ACC+18] providing suggested parameters based on the currently best-known attacks. Some implementations along with the schemes they implement are listed in Table 1.1.

Remark 1.3. The FV scheme introduced in [FV12] is sometimes referred to as the BFV scheme in the literature. This is because the authors of [FV12] ported the scheme proposed in [Bra12] to the ring-LWE setting. In this thesis we will refer to the scheme as FV, except

as seen in table 1.1.

Parameter selection is often done using an online tool [APS15] based on [Pla18]. The primary homomorphic encryption scheme implementation we will refer to is [Lai17]. We make a note that this paper is slightly dated and there have been some significant changes in SEAL since publication of [Lai17]. When one wants to consider fully homomorphic encryption, there is no better resource than Gentry's thesis [Gen09] in which they introduce the concept of bootstrapping. At the time of this writing, bootstapping appears to be the only known way of achieving fully homomorphic encryption.

2 Background

This chapter aims to provide the mathematical background necessary for understanding modern homomorphic encryption schemes. We will provide a brief introduction to lattices and lattice problems, some algebraic number theory, and we will look at the problems that homomorphic encryption schemes are based on.

2.1 Notation

We will use the standard notation that is used in [FV12].

We recall the mth cyclotomic polynomial $\Phi_m(x)$ to be the unique irreducible polynomial in $\mathbb{Z}[x]$ whose roots are all primitive mth roots of unity $\zeta_m^j = e^{\frac{2\pi ij}{m}}$

$$\Phi_m(x) = \prod_{\substack{1 \le j < m \\ \gcd(j,m)=1}} (x - \zeta_m^j).$$

The degree of $\Phi_m(x)$ is $\varphi(m)$ where $\varphi(m)$ is the Euler phi function. We will typically work with m a power of 2 so that $\Phi_m(x) = x^{m/2} + 1$.

We will work with the quotient ring $R = \mathbb{Z}[x]/\langle \Phi_m(x) \rangle$. Here, we use $\langle \cdot \rangle$ to denote the ideal generated by elements within the angled brackets. We will denote elements of R in lowercase bold. For elements $\mathbf{r} \in R$, we denote the coefficients of x^i by r_i so that $r = \sum_{i=0}^{m-1} r_i x^i$ with $0 \le i < \varphi(m)$. In other words, we will work with the canonical representatives. Matrices will be denoted by uppercase bold, and by an abuse of notation, we will also denote vectors of a lattice in lowercase bold.

For $q \in \mathbb{Z}$, we let $\mathbb{Z}_q$ denote the set of canonical representatives of $\mathbb{Z}/q\mathbb{Z}$ in $(-q/2, q/2]$. We let R_q denote the ring R with coefficients in $\mathbb{Z}_q$. We stress that $\mathbb{Z}_q$ is not the ring $\mathbb{Z}/q\mathbb{Z}$, but is in fact a set, and so elements of R_q have coefficients in $(-q/2, q/2]$.

By a slight abuse of notation, we will let $\langle \cdot, \cdot \rangle$ denote the standard inner product keeping

in mind we are always working with elements whose degrees are less than $\varphi(m)$, and so $\langle \mathbf{f}, \mathbf{g} \rangle = \sum r_i \cdot s_i$ where r_i, s_i are the coefficients of $\mathbf{f}, \mathbf{g}$ respectively. Clarification will be provided if it is not clear from the context whether we are denoting an ideal or an inner product. We will let $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ denote rounding down and rounding up respectively. We let $\lfloor \cdot \rceil$ denote rounding to the nearest integer; in case of any ambiguity we round up. We let $[\cdot]_q$ denote reduction modulo q in the range $(-q/2, q/2]$ and $|\cdot|_q$ denote the least positive residue modulo q. When we say "reduce modulo q", we are applying $[\cdot]_q$ unless otherwise stated. When working with elements of R, the operations $\lfloor \cdot \rfloor_q$, $|\cdot|_q$, and $\lfloor \cdot \rceil$ are performed coefficient-wise.

When given a probability distribution $\mathcal{D}$, we denote sampling an element e from $\mathcal{D}$ by $e \leftarrow \mathcal{D}$. When $\mathcal{D}$ is the uniform distribution $\mathcal{U}$, we denote sampling by $e \xleftarrow{\$} \mathcal{U}$. We will often sample $\mathbf{e} \in R$ from $\mathcal{D}$. By this, we mean that the coefficients are each sampled individually from $\mathcal{D}$. We say $\mathcal{D}$ is *B-bounded* if it is supported on the interval $[-B, B]$.

We will take $\|\cdot\|$ to mean the infinity norm, that is $\|\mathbf{v}\| = \max\{|v_i|\}$. We will otherwise let $\|\cdot\|_p$ denote the ℓ_p norm, that is $\|\mathbf{v}\|_p = \sqrt[p]{\sum_i v_i^p}$. In the case when $\mathbf{v} \in R$, we will take $\|\mathbf{v}\|$ to mean *reduce modulo* $\Phi_m(x)$ before applying $\|\cdot\|$.

When working with elements $\mathbf{f} \in R_q$, we will let $\mathbf{f}[i]$ denote the ith coefficient in $\mathbf{f}$. This is the notation most commonly used in the literature. It is not to be confused with applying $[\cdot]_q$, and clarification will be provided if the context is not clear.

2.1.1 Ring Expansion Factor

When ring elements are multiplied together and reduced modulo $\Phi_m(x)$, it is possible for their norm to increase as a result. The maximum factor which they may expand by is called the *expansion factor of the ring* and is defined as $\delta_R = \sup\left\{ \frac{\|\mathbf{f} \cdot \mathbf{g}\|}{\|\mathbf{f}\| \cdot \|\mathbf{g}\|} \text{ for all } \mathbf{f}, \mathbf{g} \in R \setminus \{0\} \right\}$. We have δ_R finite if R is a ring that is finite-dimensional as a vector space. To see this, we note that the function in δ_R is homogeneous as a function of $\mathbf{f}$ and $\mathbf{g}$, so it is determined by its values on $\mathbf{f}, \mathbf{g}$ of norm 1. This is a compact set, and a continuous function attains its maximum on a compact set. In the case of R as defined above, we can work out this expansion factor explicitly.

Lemma 2.1. *Let $R = \mathbb{Z}[x]/\langle x^n + 1\rangle$. Then $\delta_R = n$.*

Proof. Let $\mathbf{f}, \mathbf{g} \in R$ with $f = \sum_{i=0}^{n-1} a_i x^i$, $g = \sum_{i=0}^{n-1} b_i x^i$. Let $\mathbf{f} \cdot \mathbf{g} = \mathbf{h} = \sum_{i=0}^{2(n-1)} c_i x^i$ be their product prior to reduction, with $c_k = \sum_{j=0}^{k} a_j b_{k-j}$. Reducing modulo $x^n + 1$ is done by negating the last $n-1$ terms and adding them to the first $n-1$ terms. Combining terms gives $\bar{c_k} = \sum_{j=0}^{k} a_j b_{k-j} - \sum_{j=k+1}^{n-1} a_j b_{n+k-j}$ which has at most n products to sum. Then

$$\|\mathbf{f} \cdot \mathbf{g}\| = \|\mathbf{h}\| \leq n \cdot \|\mathbf{f}\| \cdot \|\mathbf{g}\|$$

gives us an upper bound of n for δ_R. To see this is a least upper bound, take $\mathbf{f}, \mathbf{g}$ to have coefficients all 1s and we get $\delta_R = n$. □

We will see below in chapter 5 that this maximum bound is attained quite rarely.

2.2 Lattices

Let $\mathbf{v}_1, \ldots, \mathbf{v}_m$ be a set of linearly independent vectors in $\mathbb{R}^n$. The *lattice* Λ generated by $\mathbf{v}_1, \ldots, \mathbf{v}_m$ is the set of all integer linear combinations of these vectors

$$\Lambda = \left\{ \sum_{i=1}^{m} a_i \mathbf{v}_i \mid a_i \in \mathbb{Z} \right\}.$$

If $\mathbf{v}_1, \ldots, \mathbf{v}_m$ are linearly independent, they are said to be a *basis* for Λ. If Λ is a full-rank lattice, that is $m = n$, we can define its dual lattice

$$\Lambda^* = \{\mathbf{x} \in \mathbb{R}^n \mid \forall \mathbf{v} \in \Lambda, \langle \mathbf{x}, \mathbf{v} \rangle \in \mathbb{Z}\}.$$

If we have $U = \{\mathbf{u}_1, \ldots, \mathbf{u}_m\}$ and $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$ two bases for Λ, then we can write

$$\mathbf{u}_i = \sum_{j}^{m} \alpha_{ij} \mathbf{v}_j, \qquad \alpha_{ij} \in \mathbb{Z}.$$

The a_{ij}'s form an invertible $m \times m$ matrix $\mathbf{A}$. Then $\mathbf{U} = \mathbf{AV}$. We have that $1 = \det(\mathbf{A}\mathbf{A}^{-1}) = \det(\mathbf{A}) \det(\mathbf{A}^{-1})$. Both A and A^{-1} have integer coefficients, so $\det(A), \det(A^{-1}) \in \mathbb{Z}$. The only units in $\mathbb{Z}$ are ± 1, so $\det(A) = \pm 1$ and we say that A is a *unimodular matrix*.

An *integer lattice* is an additive subgroup of $\mathbb{Z}^n$. That is, it is a lattice whose vectors all have integer coordinates. If Λ satisfies $q\mathbb{Z}^n \subseteq \Lambda \subseteq \mathbb{Z}^n$, we say Λ is a *q-ary lattice*.

Definition 2.2. For a lattice Λ with a basis $\{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$. We define the *standard fundamental domain* of Λ to be the n-dimensional parallelepiped

$$\mathcal{F}(\mathbf{v}_1, \ldots, \mathbf{v}_m) = \left\{ \sum_{i=1}^{m} t_i \mathbf{v}_i \mid 0 \leq t_i < 1 \right\}.$$

Definition 2.3. For a lattice Λ of dimension m and fundamental domain $\mathcal{F}$, we define the determinant $\det(\Lambda)$ of Λ to be $\mathrm{Vol}(\mathcal{F})$ the volume of the standard fundamental domain. This is also called the covolume of Λ since it is the volume of the quotient group $\mathbb{R}^n/\Lambda$.

Remark 2.4. The volume of a fundamental domain does not depend on the choice of fundamental domain, at least as long as one avoids pathologies (for example, by insisting that the domain is Lebesgue measurable). We will work consistently with the standard fundamental domain.

If $\dim \Lambda = n$, then we can form a square matrix $F(V)$ by the rows of the basis vectors V of Λ and compute $\mathrm{Vol}(\mathcal{F})$ by

$$\mathrm{Vol}(\mathcal{F}) = |\det(F(V))|.$$

The volume of the standard fundamental domain is independent of the basis chosen for Λ. To see this, we let U, V be any two bases for Λ related by a unimodular matrix A. Then

$$\begin{aligned} \mathrm{Vol}(\mathcal{F}(V)) &= |\det(F(V))| \\ &= |\det(AF(U))| \\ &= |\det(A)\det(F(U))| \\ &= |\pm 1|\,|\det(F(U)|| \\ &= \mathrm{Vol}(\mathcal{F}(U)). \end{aligned}$$

Definition 2.5. The ith successive minimum, $\lambda_i = \lambda_i(\Lambda)$, of a lattice Λ is the radius r of the smallest ball such that there are i linearly independent vectors of length $\|\mathbf{v}_i\|_2 \leq r$. It is clear that $0 < \lambda_1 \leq \cdots \leq \lambda_n$.

2.2.1 Ideal Lattices

We can form lattices based on ideals of rings.

Definition 2.6. If we let $f(x) \in \mathbb{Z}[x]$ be a monic polynomial of degree n, and $R = \mathbb{Z}[x]/\langle f(x)\rangle$ be a ring then we have $R \cong \mathbb{Z}^n$ by $x^i \mapsto \mathbf{e}_i$ where the $\mathbf{e}_i$s are the standard basis vectors of $\mathbb{Z}^n$. Ideals of $I \subseteq R$ also form lattices which are isomorphic to sublattices of $\mathbb{Z}^n$. Such a lattice is called an *ideal lattice*.

We remark that although the ideals of R all form lattices in $\mathbb{Z}^n$, not all sublattices of $\mathbb{Z}^n$ correspond to some ideal in R.

Example 2.7. Let $\Lambda = \mathbb{Z}[x]/\langle x^2+1\rangle$. Consider the sublattice generated by the basis $\mathbf{b}_1 = (1+5x), \mathbf{b}_2 = (1+2x)$. In this sublattice, multiplication by x corresponds to $\pi/2$ rotation. The vector $-7\cdot\mathbf{b}_1 + 2\cdot\mathbf{b}_2 = -5-31x$ is in this sublattice, but $x\cdot(-5-31x) = (31-5x)$ is not in the sublattice, and therefore it is not generated by an ideal since ideals are closed under multiplication.

Ideal lattices and their applications to cryptography were first introduced [LM06] as a generalization of a cyclic lattice (i.e., a lattice Λ such that $\mathrm{rot}(\Lambda) = \Lambda$, where rot maps $x_i \mapsto x_{(i+1 \mod n)}$).

We recall that ideals are closed under both addition and scalar multiplication by elements in R. This additional structure is what makes ideal lattices an appealing candidate for HE schemes, and in fact is what Gentry uses in [Gen09] to construct an FHE scheme.

2.3 Cyclotomic Fields

We will now give some background on the algebraic structures used. We begin by recalling some definitions. In this section, modular arithmetic may be performed in the usual way and is not restricted to a set of representatives as in section 2.1. For the following definitions, we let K/F be a field extension.

Definition 2.8. An element $\alpha \in K$ is said to be *algebraic over* F if it is the root of some polynomial in $F[x]$. If every element of K is algebraic over F, then we say K/F is algebraic.

If K/F is a finite field extension, then K/F is algebraic. The converse is not true. There exist infinite field extensions which are algebraic over the basefield. For example,

$K = \mathbb{Q}(\sqrt{2}, \sqrt{3}, \sqrt{5}, \dots)$ the field extension generated by all square roots of prime numbers is an infinite, algebraic extension.

Definition 2.9. If $F = \mathbb{Q}$, and $K = \mathbb{Q}(\alpha)$ a finite extension for some α algebraic over $\mathbb{Q}$, then we say α is an *algebraic number* and K is an *algebraic number field*, or simply a *number field*. If α is a root of a monic polynomial in $\mathbb{Z}[x]$, we call α an *algebraic integer*.

Recall that for a number field $K = \mathbb{Q}(\alpha)$, there exists a unique irreducible polynomial $f(x) \in \mathbb{Q}[x]$ whose root is α. This polynomial is called the *minimal polynomial* of α, and $\deg f(x) = [K : \mathbb{Q}]$ (i.e. the dimension of K as a vector space over $\mathbb{Q}$). If we let the minimal polynomial of α be of degree n, then

$$K = \left\{ \sum_i a_i \alpha^i \mid a_i \in \mathbb{Q},\ 0 \leq i < n \right\}.$$

We then clearly have $K \cong \mathbb{Q}[x]/\langle f(x) \rangle$ given by $\alpha \mapsto x$.

Definition 2.10. For any positive integer m, and ζ_m a primitive mth root of unity, the *mth cyclotomic polynomial* is the minimal polynomial

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

whose roots are all the primitive mth roots of unity.

We obtain the mth cyclotomic field $K = \mathbb{Q}(\zeta_m)$ by adjoining an mth primitive root of unity ζ_m to $\mathbb{Q}$. The dimension of K over $\mathbb{Q}$ as a vector space is equal to $\varphi(m)$. In this paper, we primarily concern ourselves with cyclotomic number fields.

2.3.1 Rings of Integers

Definition 2.11. Let K be a number field. The set of all algebraic integers in K is called the *ring of integers* of K and is denoted $\mathcal{O}_K$.

It is worth noting $\mathcal{O}_K$ is indeed a subring of K. When we say "ring of integers," it is implied we are referring to the ring of integers of some number field K. To avoid confusion, we will refer to $\mathbb{Z}$ as the rational integers.

Definition 2.12. A *unique factorization domain* (UFD) is an integral domain in which every non-zero element can be written uniquely as a product of irreducible elements up to permutation and multiplication by units.

The rational integers are a UFD as all integers factor uniquely into a product of prime powers, but in general this is not the case with the ring of integers.

Example 2.13. Let $K = \mathbb{Q}[\sqrt{-5}]$, then we have $\mathcal{O}_K = \mathbb{Z}[\sqrt{-5}]$. In this ring, $6 = 2 \cdot 3$ and $6 = (1 + \sqrt{-5})(1 - \sqrt{-5})$, but $2, 3, (1 + \sqrt{-5})$, and $(1 - \sqrt{-5})$ are all irreducible elements in $\mathcal{O}_K$, so we have two different factorizations of 6 into irreducible elements.

In general, unique factorization does not hold in $\mathcal{O}_K$, but as it turns out this concept of unique factorization may be generalized. We make use of the following results from algebraic number theory.

Theorem 2.14. *Every ideal in a Dedekind domain factors uniquely into a product of prime ideals.*

Proof. See [Neu13, thm. 3.3] □

Theorem 2.15. *The ring of integers is a Dedekind domain.*

Proof. See [Neu13, thm. 3.1] □

So it is clear that although *elements* may not factor uniquely in the ring of integers, ideals of $\mathcal{O}_K$ factor uniquely into products of prime ideals. This allows us to begin discussion on the splitting of primes in the ring of integers.

Definition 2.16. Let K be a number field and $\mathcal{O}_K$ its ring of integers. A prime $p \in \mathbb{Z}$ is said to *split* if $\langle p \rangle \in \mathcal{O}_K$ can be written as a non-trivial product of prime ideals in $\mathcal{O}_K$. If this product is square-free, we say p *splits completely*, and p is ramified if it is not square-free. If the only factorization of p is the trivial one, we say p remains prime in $\mathcal{O}_K$.

Example 2.17. Consider the ring of Gaussian integers $\mathbb{Z}[i]$. In this ring we can write $5 = (1+2i)(1-2i)$, and so 5 splits. It can be shown that every prime p of the form $p \equiv 1 \pmod 4$ splits. Conversely, if $p \equiv 3 \pmod 4$, then p does not split and remains prime. Furthermore, we have $2 = u(1 + i)^2$ where u is some unit in $\mathbb{Z}[i]$, and so 2 is ramified.

Theorem 2.18. *The ring of integers of $\mathbb{Q}(\zeta_m)$ is $\mathbb{Z}[\zeta_m]$.*

Proof. See [Was97, thm. 2.6]. □

Remark 2.19. The examples we have seen so far have involved number rings of the form $K = \mathbb{Q}(\alpha)$ and their corresponding rings of integers have been $\mathcal{O}_K = \mathbb{Z}[\alpha]$; in general we do not always have $\mathcal{O}_K = \mathbb{Z}[\alpha]$. One nice feature of cyclotomic fields is that their ring of integers is always monogenic.

Theorem 2.20. *Let $K = \mathbb{Q}(\zeta_m)$, p prime with $p \nmid m$, and f the order of $p \mod m$. Then p splits into $\varphi(m)/f$ distinct primes in $\mathbb{Z}[\zeta_m]$.*

Proof. See [Was97, thm. 2.13]. □

When considering the mth cyclotomic field, we have $\mathcal{O}_K = \mathbb{Z}[\zeta_m] \cong \mathbb{Z}[x]/\langle \Phi_m(x) \rangle$, and Theorem 2.20 tells us that primes of the form $p \equiv 1 \mod m$ split completely. Another result from number theory tells us exactly how to find these prime factors. We give the special instance of this method, namely when $\mathcal{O}_K = \mathbb{Z}[\alpha]$ for $K = \mathbb{Q}(\alpha)$.

Theorem 2.21. *Let $p \in \mathbb{Z}$ prime, and K and $\mathcal{O}_K$ as above. Let f be the minimal polynomial of α over $\mathbb{Q}$. Let $\bar{f}$ denote the image of f in $(\mathbb{Z}/p\mathbb{Z})[x]$. We factor $\bar{f}$ as*

$$\bar{f}(x) = \bar{f}_1 \cdot \bar{f}_2 \cdot \cdots \cdot \bar{f}_r.$$

Then $p\mathcal{O}_K$ factors uniquely in $\mathcal{O}_K$ as

$$p\mathcal{O}_K = \mathfrak{p}_1^{e_1} \mathfrak{p}_2^{e_2} \dots \mathfrak{p}_r^{e_r}$$

where $\mathfrak{p}_i = p\mathcal{O}_K + \langle f_i(\alpha) \rangle = \langle p, f_i(\alpha) \rangle$ and $e_i = \deg f_i$.

Proof. See [Neu13, prop. 8.3] □

For algorithms on factoring polynomials modulo p, see [Coh13].

Theorem 2.21 shows we can find a factorization of p simply by factoring a cyclotomic polynomial modulo p and evaluating each of its factors at ζ_m. Since we know primes of the form $p \equiv 1 \mod m$ split completely in $\mathbb{Z}[\zeta_m]$, $\Phi_m(x)$ must factor into exactly $n = \varphi(m)$ linear factors. Since $(\mathbb{Z}/p\mathbb{Z})^\times$ is cyclic of order $p - 1$, it has some generator g. We have $m \mid p - 1$,

and so there exists a unique subgroup of order m with a generator of the form $a = (g^{(p-1)/m})$. Each linear factor in the factorization of Φ_m modulo p is given by $(x - a^i)$ for $i \in (\mathbb{Z}/m\mathbb{Z})^\times$. Finally, by the CRT this gives an isomorphism

$$(\mathbb{Z}/p\mathbb{Z})[x]/\langle \Phi_m \rangle \cong (\mathbb{Z}/p\mathbb{Z})[\zeta_m] \cong \prod_{\substack{1 \le i < m, \\ (i,m)=1}} (\mathbb{Z}/p\mathbb{Z})[\zeta_m]/\left\langle p, \zeta_m^i - a^i \right\rangle.$$

Each of the factors in the product of quotients is prime and therefore maximal, so each factor is isomorphic to a field, namely $\mathbb{Z}/p\mathbb{Z}$. This gives $(\mathbb{Z}/p\mathbb{Z})[x]/\Phi_m \cong (\mathbb{Z}/p\mathbb{Z})^n$. We conclude this section with an example of splitting a prime.

Example 2.22. Let $K = \mathbb{Q}(\zeta_8)$ be the eighth cyclotomic field. The minimal polynomial of ζ_8 is $\Phi_8(x) = x^4 + 1$. If we choose $p = 17$, then Φ_8 factors as $(x+2) \cdot (x+8) \cdot (x+9) \cdot (x+15)$ modulo p. By theorem 2.21, we have that 5 splits as $5\mathcal{O}_K = \langle 5, \zeta_8 + 2\rangle \cdot \langle 5, \zeta_8 + 8\rangle \cdot \langle 5, \zeta_8 + 9\rangle \cdot \langle 5, \zeta_8 + 15\rangle$.

Implementations will typically impose congruence conditions on p for this splitting property, and allows us to work over a finite field of order p rather than some extension field. This CRT isomorphism allows us to perform pointwise multiplication. An application of this is single instruction, multiple data (SIMD).

2.4 Cryptographically Hard Problems

In this section we will examine the underlying hard problems that homomorphic encryption schemes are based on.

Definition 2.23. Given a lattice Λ, the *Shortest Vector Problem(SVP)* is to find a shortest non-zero vector in the lattice Λ. In terms of the notation used above, find a vector $\mathbf{v} \in \Lambda$ such that $\|\mathbf{v}\|_2 = \lambda_1(\Lambda)$.

Remark 2.24. There may be more than one shortest vector in Λ. For example, if we work in the integer lattice $\mathbb{Z}^2$, the vectors $(0, \pm 1), (\pm 1, 0)$ are solutions to SVP. In a lattice, there are always finitely many shortest vectors.

The SVP problem is known to be NP-hard under randomized reductions [Ajt98].

Definition 2.25. Given a lattice Λ and a target vector $\mathbf{t} \in \mathbb{R}^n$ such that $\mathbf{t} \notin \Lambda$, the *Closest Vector Problem (CVP)* is to find a vector $\mathbf{v} \in \Lambda$ such that $\|\mathbf{t} - \mathbf{v}\|_2$ is minimized.

There are several variants of SVP and CVP. Below we will let $\gamma \geq 1$ be an approximation factor and we will let $d(\mathbf{t}, \Lambda)$ be the distance of a vector $\mathbf{t} \in \mathbb{R}^n$ to the closest lattice point in Λ. In the $\gamma = 1$ case, we get the versions of these problems introduced above.

Definition 2.26. Given a lattice Λ, the γ-*Approximate Shortest Vector Problem (SVP$_\gamma$)* is to find a non-zero vector $\mathbf{v} \in \Lambda$ such that $\|\mathbf{v}\|_2 \leq \gamma \cdot \lambda_1(\Lambda)$.

Definition 2.27. Given a lattice Λ and a target vector $\mathbf{t} \in \mathbb{R}^n$, the γ-*Approximate Closest Vector Problem (CVP$_\gamma$)* is to find a vector $\mathbf{v} \in \Lambda$ such that $\|\mathbf{t} - \mathbf{v}\|_2 \leq \gamma \cdot d(\mathbf{t}, \Lambda)$.

Definition 2.28. Given a lattice Λ, the γ-*Shortest Independent Vector Problem (SIVP$_\gamma$)* is to find linearly independent vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ such that $\|\mathbf{v}_i\| \leq \gamma \lambda_n(\Lambda)$

Definition 2.29. Given a basis $\mathbb{B}$ of a lattice $\Lambda(\mathbb{B})$, and a target vector $\mathbf{t}$ such that $d(\mathbf{t}, \Lambda) \leq \gamma \cdot \lambda_1(\Lambda)$, the γ-*Bounded Distance Decoding Problem (BDD$_\gamma$)* is to find a vector $\mathbf{y} \in \Lambda$ such that $\|\mathbf{y} - \mathbf{t}\|_2 < d(\mathbf{t}, \Lambda)$.

Definition 2.30. We let $\mathbb{Z}_q^n$ denote n-dimensional vectors with coefficients modulo q. Given vectors $\mathbf{a}_1, \ldots \mathbf{a}_m \in \mathbb{Z}_q^n$, and a bound $\beta < q$, *The Short Integer Solutions Problem(SIS)* is to find a non-trivial, "short" $\mathbf{z} \in \mathbb{Z}^m$ such that

$$\mathbf{A}\mathbf{z} = \mathbf{0} \pmod{q}$$

where $\mathbf{A} = [\mathbf{a}_1 | \ldots | \mathbf{a}_m]$. We require $\|\mathbf{z}\| \leq \beta$ otherwise we can simply use Gaussian elimination to solve for $\mathbf{z}$.

In SIS, the matrix $\mathbf{A}$ defines a lattice

$$\Lambda(\mathbf{A})^\perp = \{\mathbf{z} \in \mathbb{Z}^m : \mathbf{A}\mathbf{z} = \mathbf{0} \pmod{q}\}$$

and the problem becomes finding a shortest vector in $\Lambda(\mathbf{A})$. In 1996, Ajtai gave a reduction from worst-case GapSVP and SIVP to average-case SIS[Ajt96].

2.4.1 Learning with Errors

The Learning with Errors problem (LWE) was first introduced by Regev in 2005 [Reg05]. We look at two versions of the LWE problem.

Definition 2.31. Let $n, q \in \mathbb{Z}$ and χ a probability distribution on $\mathbb{Z}$ often referred to as the error distribution. Then the *Search-LWE problem* is to find a vector $\mathbf{s} \in \mathbb{Z}_q^n$ given m noisy inner products

$$\begin{aligned} b_1 &= \langle \mathbf{s}, \mathbf{a}_1 \rangle + e_1 \\ &\vdots \\ b_m &= \langle \mathbf{s}, \mathbf{a}_m \rangle + e_m \end{aligned}$$

where $\mathbf{a}_i \xleftarrow{\$} \mathbb{Z}_q^n$ and $e_i \leftarrow \chi$.

Definition 2.32. Let n, q, χ as in definition 2.31, $\mathbf{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_m)$ with $\mathbf{a}_i \xleftarrow{\$} \mathbb{Z}_q^n$ and $\mathbf{e} = (e_1, \ldots, e_m)$ with $e_i \leftarrow \chi$. Then for some $\mathbf{s} \in \mathbb{Z}_q^n$, the *Decision-LWE problem* is to distinguish between $(\mathbf{A}, \mathbf{s}^T\mathbf{A} + \mathbf{e}^T)$ and $(\mathbf{A}, \mathbf{v}^T)$ with $\mathbf{v} \xleftarrow{\$} \mathbb{Z}_q^m$.

We typically choose χ to be a spherical Gaussian distribution centred around the origin with width $\alpha q > \sqrt{n}$ for some α.

Let $A \in \mathbb{Z}_q^{m \times n}$ to be the $m \times n$ matrix representing each random vector $\mathbf{a}_i$. If we have $m \leq n$, we can solve the above system using Gaussian elimination. However, since Gaussian elimination uses row operations, this makes any error terms grow and our solution will effectively be a random vector. If we have $m > n$, then adding the error term to $\mathrm{Im}(A)$ moves outside of $\mathrm{Im}(A)$ due to being in a higher dimensional space. This makes it impossible for Gaussian elimination to return anything since there is a high probability an arbitrary vector in a higher-dimensional space won't have a pre-image; in other words, the system will almost certainly be inconsistent.

The LWE problem is provably as hard as some worst-case lattice problems. Regev gives a quantum reduction from GapSVP and SIVP to LWE [Reg05], and a classical reduction from GapSVP to LWE is given in [BLP+13].

2.4.2 Cryptographic Applications

Using the LWE problem, we can create an asymmetric key cryptography scheme. We begin by looking at the scheme proposed by Regev [Reg05], and later examine the variant in [FV12]. First, we choose public parameters n, q, m, σ with the variable n being the dimension of the vector space we are working with, q a prime modulus, m the number of equations in our system, and σ the standard deviation of the probability distribution χ on $\mathbb{Z}_q$. The remainder of the scheme is as follows:

- **Private Key:** Choose a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$.
- **Public Key:** Choose a matrix $A \in \mathbb{Z}_q^{m \times n}$ uniformly at random, and sample $\mathbf{e} \in \chi_q^m$. Return the public key as $pk = (A, \mathbf{b} = A \cdot \mathbf{s} + \mathbf{e}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^m$. We note that the noise $\mathbf{e}$ is exposed by taking $\texttt{pk}[0] \cdot \mathbf{s} - \texttt{pk}[1]$.
- **Encrypt:** Encryption is done on each bit x of the message as follows: choose a uniformly random vector $\mathbf{s} \in \mathbb{Z}_2^m$. Return $(A \cdot \mathbf{s}, \langle \mathbf{b}, \mathbf{s} \rangle) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ if $x = 0$, and return $(A \cdot \mathbf{s}, \lfloor \frac{q}{2} \rfloor + \langle \mathbf{b}, \mathbf{s} \rangle) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ if $x = 1$.
- **Decrypt:** Given the ciphertext pair $(\mathbf{a}, b)$, return 0 if $b - \langle \mathbf{a}, \mathbf{s} \rangle$ is closer to 0 than to $\lfloor \frac{q}{2} \rfloor$, and 1 otherwise.

2.5 Ring-LWE

For the Ring Learning with Errors (Ring-LWE) problem, we modify the LWE problem slightly. First, we let $\Phi_m(x)$ be the mth cyclotomic polynomial. We choose $q \in \mathbb{Z}$ with $q \equiv 1 \mod m$. We let $R = \mathbb{Z}[x] / \langle \Phi_m(x) \rangle$ be a polynomial quotient ring and we let $R_q = R/\mathbb{Z}_q$ be the polynomial quotient ring with coefficients in $(-q/2, q/2]$. For efficiency reasons, we usually choose m to be a power of 2. This gives more efficient algorithms for multiplication discussed in chapter 4. We choose a secret polynomial $\mathbf{s} \in R_q$ as before. We let $A_{\mathbf{s},\chi}$ be the distribution in $R_q \times R_q$ obtained by sampling $\mathbf{a} \in R_q$ uniformly at random, $\mathbf{e} \in R_q$ the polynomial with coefficients obtained by sampling from χ like before, and returning the pair $(\mathbf{a}, \mathbf{b} = \mathbf{a} \cdot \mathbf{s} + \mathbf{e}) \in R_q \times R_q$.

Definition 2.33. The *Search Ring Learning with Errors Problem (Search Ring-LWE)* is to determine the secret $\mathbf{s} \in R_q$ by sampling pairs $(\mathbf{a}_i, \mathbf{b}_i)$ from $A_{\mathbf{s},\chi}$.

Definition 2.34. Given samples of pairs $(\mathbf{a}_i, \mathbf{b}_i)$ as above, the *Decision Ring Learning with Errors Problem (Decision Ring-LWE)* is to distinguish between pairs sampled uniformly at random and pairs sampled according to the distribution $A_{\mathbf{s},\chi}$.

2.5.1 Cryptographic Application

We will first look at a basic scheme introduced in [LPR10] when the ring-LWE problem was first introduced.

As with LWE, we choose a prime modulus q with the extra condition that $q \equiv 1 \mod 2d$ where d is a power of 2, and χ an error distribution on $\mathbb{Z}_q$. We work with elements in the ring $R_q = \mathbb{Z}_q[x]/\langle x^d + 1\rangle$. The steps are then as follows:

- **Private Key:** Choose a secret $\mathbf{s} \in R$ with coefficients sampled from χ.

- **Public Key:** Choose an element $\mathbf{a} \xleftarrow{\$} R_q$ and output the pair $\mathtt{pk} = (\mathbf{a}, \mathbf{b} = \mathbf{a} \cdot \mathbf{s} + \mathbf{e}) \in R_q \times R_q$ where $\mathbf{e}$ is an error term chosen from the error distribution. As with LWE, we see that the noise $\mathbf{e}$ is exposed by taking $\mathtt{pk}[0] \cdot \mathbf{s} - \mathtt{pk}[1]$.

- **Encrypt:** The plaintext space is R_t. The authors of [LPR10] use the $t = 2$ case, but we are not restricted to working with $t = 2$, and so will consider the general case here. To encrypt a message $\mathbf{m} \in R_t$, we begin by sampling three small error terms $\mathbf{u}, \mathbf{e}_1, \mathbf{e}_2$ from our error distribution. We then compute

$$\begin{aligned} \mathbf{c}_1 &= [\mathbf{a} \cdot \mathbf{u} + \mathbf{e}_1]_q \\ \mathbf{c}_2 &= \left[\mathbf{b} \cdot \mathbf{u} + \mathbf{e}_2 + \left\lfloor \frac{q}{t} \right\rfloor \cdot \mathbf{m}\right]_q . \end{aligned}$$

  The ciphertext we output is $\mathtt{ct} = (\mathbf{c}_1, \mathbf{c}_2) \in R_q \times R_q$.

- **Decrypt:** To decrypt a pair $\mathtt{ct} = (\mathbf{c}_1, \mathbf{c}_2)$, we first compute

$$\begin{aligned}\mathbf{c}_2 - \mathbf{c}_1 \cdot \mathbf{s} &= \mathbf{b} \cdot \mathbf{u} + \mathbf{e}_2 + \left\lfloor \frac{q}{t} \right\rfloor \cdot \mathbf{m} - \mathbf{a} \cdot \mathbf{u} \cdot \mathbf{s} - \mathbf{e}_1 \cdot \mathbf{s} \mod q \\ &= \mathbf{a} \cdot \mathbf{u} \cdot \mathbf{s} + \mathbf{e} \cdot \mathbf{u} + \mathbf{e}_2 + \left\lfloor \frac{q}{t} \right\rfloor \cdot \mathbf{m} - \mathbf{a} \cdot \mathbf{u} \cdot \mathbf{s} - \mathbf{e}_1 \cdot \mathbf{s} \mod q \\ &= \left\lfloor \frac{q}{t} \right\rfloor \cdot \mathbf{m} + (\mathbf{e} \cdot \mathbf{u} + \mathbf{e}_2 - \mathbf{e}_1 \cdot \mathbf{s}) \mod q.\end{aligned}$$

  We refer to the term in parentheses as the noise term, and it is denoted by $\mathbf{v}$. We see that if we scale the above expression by $\frac{t}{q}$ and round, that as long as the noise isn't too large, we recover the message $\mathbf{m}$.

Lemma 2.35. *For a ciphertext* $\mathbf{c}$ *with noise as above, and* $\mathbf{a} \xleftarrow{\$} R_q$, $\mathbf{s}, \mathbf{u}, \mathbf{e}_1, \mathbf{e}_2 \leftarrow \mathcal{D}$ *where* $\mathcal{D}$ *is a* B*-bounded distribution, decryption is correct if* $\|\mathbf{v}\| < \frac{1}{2}\left(\frac{q}{t} - |q|_t\right)$.

Proof. From above, we have $\mathbf{c}_2 - \mathbf{c}_1 \cdot \mathbf{s} = \left\lfloor \frac{q}{t} \right\rfloor \cdot \mathbf{m} + \mathbf{v} + q \cdot \mathbf{r}$ for some $\mathbf{r} \in R$. Using the fact that $q = \left\lfloor \frac{q}{t} \right\rfloor \cdot t + |q|_t$ where $|q|_t$ is the least positive residue of q modulo t. Scaling by $\frac{t}{q}$ gives

$$\begin{aligned}\frac{t}{q}(\mathbf{c}_2 - \mathbf{c}_1 \cdot \mathbf{s}) &= \mathbf{m} - \frac{|q|_t}{t}\mathbf{m} + \frac{t}{q}\mathbf{v} + t\mathbf{r} \\ &= \mathbf{m} + \frac{t}{q}\left(\mathbf{v} - \frac{|q|_t}{t}\mathbf{m}\right) + t\mathbf{r}.\end{aligned}$$

Since the final step of decryption is applying $[\cdot]$, we can simply drop $t\mathbf{r}$. Then to decrypt correctly we require $\left\| \frac{t}{q}\left(\mathbf{v} - \frac{|q|_t}{t}\mathbf{m}\right) \right\| < \frac{1}{2}$.

$$\begin{aligned}\left\| \frac{t}{q}\left(\mathbf{v} - \frac{|q|_t}{t}\mathbf{m}\right) \right\| &\leq \left\| \frac{t}{q}\mathbf{v} \right\| + \left\| \frac{t}{q} \cdot \frac{|q|_t}{t}\mathbf{m} \right\| \\ &\leq \frac{t}{q}\|\mathbf{v}\| + \frac{|q|_t}{q}\|\mathbf{m}\| \\ &\leq \frac{t}{q}\|\mathbf{v}\| + \frac{|q|_t}{q} \cdot \frac{t}{2}, \qquad \text{since } \|\mathbf{m}\| \leq \frac{t}{2} \\ &< \frac{1}{2} \\ \implies \|\mathbf{v}\| &< \frac{1}{2}\left(\frac{q}{t} - |q|_t\right).\end{aligned}$$

□

It is worth noting that the literature has conflicting values for the bound on $\mathbf{v}$. In [FV12], they give a bound of $\|\mathbf{v}\| < \frac{1}{2} \cdot \lfloor \frac{q}{t} \rfloor$ and omit the details of the proof; [BEHZ17] gives the

above bound, but also omits the details of the proof; and Player gives a bound of $\|\mathbf{v}\| < \frac{1}{2}\left(\frac{q}{t} - t\right)$[Pla18]. Player's work differs slightly from above and so uses the fact that $|q|_t < t$ to get their bound. Player's bound is more of a worst-case scenario, but not necessary since we know the values q, t. The bound in [FV12] appears to be incorrect, and we therefore use the bound above and found in [BEHZ17]. Proving the bound given in [FV12] would be a stronger result since it would permit a higher bound on the noise before decryption or bootstrapping is required.

An advantage of the ring-LWE scheme presented is that key sizes are reduced compared to the LWE scheme. The LWE cryptosystem required $m \times n$ samples and a matrix product to give m random scalars, while its ring-LWE counterpart only required sampling n coefficients followed by a polynomial multiplication. This increased efficiency is why most state-of-the-art implementations of FHE are based on ring-LWE rather than LWE.

On the matter of hardness of ring-LWE, a quantum reduction from worst-case SVP_γ on ideal lattices to search ring-LWE is given in [LPR10]. Furthermore, the best-known attacks on ideal lattices appear to have no significant advantage compared to those on a random lattice. Since LWE-based schemes are reducible to worst-case lattice problems, parameter selection for security of ring-LWE is often based on the best-known attacks against LWE [ACC$^+$18, APS15].

3 FV Scheme

In this thesis we will focus on the somewhat homomorphic encryption scheme proposed in 2012 by Fan and Vercauteren [FV12]. They introduce a ring-LWE variant of Brakerski's scheme in [Bra12] which builds from the scheme proposed in [LPR10]. There is a difference of a sign and order with the public keys in [FV12] compared to [LPR10], and we adopt the convention used in [FV12] for convenience with decryption, and the notation is as defined in chapter 2. We will also look at optimizations to FV, parameter selection, and an implementation of FV, namely Microsoft's Simple Encrypted Arithmetic Library (SEAL).

The problem with the scheme proposed above is that ciphertexts grow in size. Multiplying two ciphertexts together results in a ciphertext containing three terms. The FV scheme introduces the idea of *relinearization*. If we view the two components of our ciphertexts as being coefficients in $R_q[y]$, then we can see that ciphertexts are initially linear. If we multiply two ciphertexts $\mathbf{c}_1, \mathbf{c}_2 \in R_q[y]$, we end up with something quadratic. Relinearization allows us to shrink this quadratic ciphertext product into something linear. When we view ciphertexts as elements of $R_q[y]$, we can view the first step of decryption as evaluating the ciphertext at $\mathbf{s}$ where $\mathbf{s} \in R_q$ is the secret key.

3.1 Homomorphic Operations

The two operations we work with are addition and multiplication. As mentioned above, we will view our ciphertexts as elements in $R_q[y]$. We will write $\mathtt{ct}(y) = \mathbf{c}_0 + \mathbf{c}_1 \cdot y$. If we evaluate $\mathtt{ct}(y)$ at its secret key $\mathbf{s}$ and reduce modulo q, we get

$$[\mathtt{ct}(\mathbf{s})]_q = [\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s}]_q = \left\lfloor \frac{q}{t} \right\rfloor \cdot \mathbf{m} + \mathbf{v}. \tag{3.1}$$

The reversal of the terms using addition rather than subtraction here is due to the difference in public key mentioned above. Rounding and reducing modulo t then gives us our plaintext

as required.

3.1.1 Addition

The goal here is to take two ciphertexts $\mathtt{ct}_1$, $\mathtt{ct}_2$, add them together, and have their sum decrypt to the sum of their respective plaintexts $\mathbf{m}_1, \mathbf{m}_2 \in R_t$. If we evaluate the ciphertexts at $\mathbf{s}$ as above and add them, we obtain

$$\begin{aligned}\left[\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s}\right]_q &= \left\lfloor \frac{q}{t} \right\rceil \cdot \mathbf{m}_1 + \mathbf{v}_1 + \left\lfloor \frac{q}{t} \right\rceil \cdot \mathbf{m}_2 + \mathbf{v}_2 \\ &= \left\lfloor \frac{q}{t} \right\rceil \cdot (\mathbf{m}_1 + \mathbf{m}_2) + \mathbf{v}_1 + \mathbf{v}_2.\end{aligned}$$

In order to add the $\mathbf{m}_i$'s, we need to reduce modulo t afterwards. We have $\mathbf{m}_1 + \mathbf{m}_2 = [\mathbf{m}_1 + \mathbf{m}_2]_t + t \cdot \mathbf{r}$ for some $\mathbf{r} \in R$. Simplifying above gives

$$\begin{aligned}\mathbf{c}_1 + \mathbf{c}_2 \cdot \mathbf{s} &= \left\lfloor \frac{q}{t} \right\rceil \cdot [\mathbf{m}_1 + \mathbf{m}_2]_t + \left\lfloor \frac{q}{t} \right\rceil \cdot t \cdot \mathbf{r} + \mathbf{v}_1 + \mathbf{v}_2 + q \cdot (\alpha_1 + \alpha_2) \\ &= \left\lfloor \frac{q}{t} \right\rceil \cdot [\mathbf{m}_1 + \mathbf{m}_2]_t + \frac{q}{t} \cdot t \cdot \mathbf{r} - |q|_t \, \mathbf{r} + \mathbf{v}_1 + q \cdot (\alpha_1 + \alpha_2) + \mathbf{v}_2 \\ &= \left\lfloor \frac{q}{t} \right\rceil \cdot [\mathbf{m}_1 + \mathbf{m}_2]_t + \mathbf{v}_1 + \mathbf{v}_2 - \epsilon \cdot t \cdot \mathbf{r} + q \cdot (\mathbf{r} + \alpha_1 + \alpha_2)\end{aligned}$$

where $\epsilon = \frac{|q|_t}{t} = \frac{q}{t} - \left\lfloor \frac{q}{t} \right\rceil < 1$. From the appearance of the $\mathbf{v}_1 + \mathbf{v}_2$ term above, we see the noise grows additively. That is, the noise of our new ciphertext is simply the sum of the noise of the previous ciphertexts, up to some additional factor that is at most t.

3.1.2 Multiplication

As previously mentioned, multiplication introduces a new problem: we end up with a ciphertext with three elements. To see why this is the case, we recall that ciphertexts are linear elements of $R[y]$ and so their product must be quadratic. It is worth noting this is a valid ciphertext that will decrypt if we evaluate it at $\mathbf{s}$ as in (3.1) prior to reduction modulo q. We explain below how this works.

Similar to what we did with addition, we begin by writing out our cipherexts evaluated at $\mathbf{s}$. This gives us

$$\mathtt{ct}_1(\mathbf{s}) = \left\lfloor \frac{q}{t} \right\rceil \cdot \mathbf{m}_1 + \mathbf{v}_1 + q \cdot \mathbf{r}_1, \qquad \mathtt{ct}_2(\mathbf{s}) = \left\lfloor \frac{q}{t} \right\rceil \cdot \mathbf{m}_2 + \mathbf{v}_2 + q \cdot \mathbf{r}_2.$$

If we carry out this multiplication in the natural way, we get

$$(\mathtt{ct}_1 \cdot \mathtt{ct}_2)(\mathbf{s}) = \left\lfloor \frac{q}{t} \right\rfloor^2 \cdot (\mathbf{m}_1 \cdot \mathbf{m}_2) + \left\lfloor \frac{q}{t} \right\rfloor \cdot (\mathbf{m}_1 \cdot \mathbf{v}_2 + \mathbf{m}_2 \cdot \mathbf{v}_1) \\ + q \cdot \left\lfloor \frac{q}{t} \right\rfloor \cdot (\mathbf{m}_1 \cdot \mathbf{r}_2 + \mathbf{m}_2 \cdot \mathbf{r}_1) + q(\mathbf{v}_1 \cdot \mathbf{r}_2 + \mathbf{v}_2 \cdot \mathbf{r}_1) + \mathbf{v}_1 \cdot \mathbf{v}_2 + q^2 \cdot \mathbf{r}_1 \cdot \mathbf{r}_2.$$

If we want to decrypt this in the usual way and recover $\mathbf{m}_1 \cdot \mathbf{m}_2$, we see there is an extra factor of $\lfloor q/t \rfloor$ we need to get rid of before we can scale by t/q and round. The natural way to deal with this is to scale the above product by $\lfloor q/t \rfloor^{-1}$. The problem with doing this is that the term $q^2 \cdot \mathbf{r}_1 \cdot \mathbf{r}_2$ will more than likely require rounding, and we will not be able to remove it when reducing modulo q. The term $q^2 \cdot \lfloor q/t \rfloor^{-1}$ can be as large as $q/2$ after reduction modulo q. The authors of [FV12] work around this by using the approximation

$$\frac{t}{q} \cdot (\mathtt{ct}_1 \cdot \mathtt{ct}_2)(\mathbf{s}) = \left\lfloor \frac{t}{q}\mathbf{c}_0 \right\rceil + \left\lfloor \frac{t}{q}\mathbf{c}_1 \right\rceil \cdot \mathbf{s} + \left\lfloor \frac{t}{q}\mathbf{c}_2 \right\rceil \cdot \mathbf{s}^2 + \mathbf{r}_a$$

where $\mathbf{r}_a$ is an error term from approximating. With this approximation, the authors are able to get rid of the extra factor of $\lfloor q/t \rceil$ at the cost of some extra error.

Lemma 3.1. *For ciphertexts* $\mathtt{ct}_1, \mathtt{ct}_2$*, with* $[\mathtt{ct}_i(\mathbf{s})]_q = \left\lfloor \frac{q}{t} \cdot \mathbf{m}_i + \mathbf{v}_i \right\rfloor$*, and* $\mathtt{ct}_1 \cdot \mathtt{ct}_2(y) = \mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s} + \mathbf{c}_2 \cdot \mathbf{s}^2$*, and the noise terms* $\mathbf{v}_i$ *are bounded by some value* E*, we have*

$$\left[\left\lfloor \frac{t}{q}\mathbf{c}_0 \right\rceil + \left\lfloor \frac{t}{q}\mathbf{c}_1 \right\rceil \cdot \mathbf{s} + \left\lfloor \frac{t}{q}\mathbf{c}_2 \right\rceil \cdot \mathbf{s}^2 \right]_q = \left\lfloor \frac{q}{t} \right\rfloor \cdot \mathbf{m}_1 \cdot \mathbf{m}_2 + \mathbf{v}_3.$$

where $\|\mathbf{v}_3\| < 2 \cdot \delta_R \cdot t \cdot E \cdot (\delta_R \cdot \|\mathbf{s}\| + 1) + 2 \cdot t^2 \cdot \delta_R^2 \cdot (\|\mathbf{s}\| + 1)^2$.

Proof. See [FV12, lem. 2]. □

The above is an example of multiplying two ciphertexts that are linear in $R[y]$; the result is a ciphertext that is quadratic in $R[y]$. Classical FV only allows multiplication of degree 1 ciphertexts in $R[y]$ and bundles the relinearization method described below as part of the overall multiplication process. As it turns out, we can repeatedly use the above technique to multiply ciphertexts of *arbitrary* size, and this is what is done in [Lai17]. More concretely, if we have two ciphertexts $\mathtt{ct}_1 = (\mathbf{c}'_0, \ldots, \mathbf{c}'_k)$, $\mathtt{ct}_2 = (\mathbf{d}'_0, \ldots, \mathbf{d}'_l) \in R[y]$ of degrees k, l, with

each encrypting a product of plaintexts $\prod_{i=1}^{k} \mathbf{m}_i$, $\prod_{j=k+1}^{k+l} \mathbf{m}_j$ respectively, then we can write

$$\begin{aligned}
\mathtt{ct}_3(\mathbf{s}) &= \left\langle (\mathbf{c}_0, \ldots, \mathbf{c}_{k+l}), (1, \mathbf{s}, \ldots, \mathbf{s}^{k+l}) \right\rangle \\
&= \left\lfloor \frac{q}{t} \right\rfloor \cdot [\mathbf{m}_1 \cdot \ldots \mathbf{m}_{k+l}]_t + \mathbf{v} \\
&= (\mathtt{ct}_1 \cdot \mathtt{ct}_2)(\mathbf{s}) \\
&= \left[\sum_{i=0}^{k+l} \left\lfloor \frac{t}{q} \mathbf{c}_i \right\rceil \cdot \mathbf{s}^i \right]_q
\end{aligned}$$

where $\mathbf{c}_i = \sum_{s+t=i} \mathbf{c}'_s \cdot \mathbf{d}'_t$.

3.2 Relinearization

We first concern ourselves with classical FV (i.e., dealing with ciphertexts consisting of three elements).

Now that we have a way to multiply ciphertexts, we see the problem is that our ciphertext is quadratic in y rather than linear (i.e., it has three terms instead of two). In the original FV scheme, this would only allow us to perform a single multiplication and then we would need to decrypt the result. The authors introduce a technique called *relinearization* to bring the ciphertext back down to having two terms. They give two versions of relinearization, but we will only focus our attention on the first version.

The idea of relinearizing is to convert the above quadratic ciphertext into an equivalent linear one at the cost of some extra noise. In other words, if we have a quadratic ciphertext $\mathtt{ct}(y) = \mathbf{c}_0 + \mathbf{c}_1 \cdot y + \mathbf{c}_2 \cdot y^2$, and its "equivalent" linear ciphertext $\mathtt{ct}'(y) = \mathbf{c}'_0 + \mathbf{c}'_1 \cdot y$, we want to write

$$\left[\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s} + \mathbf{c}_2 \cdot \mathbf{s}^2\right]_q = \left[\mathbf{c}'_0 + \mathbf{c}'_1 \cdot \mathbf{s} + \mathbf{r}\right]_q \tag{3.2}$$

where $\mathbf{r}$ is a small error term from relinearizing. We need to keep in mind that relinearization is an operation that happens *homomorphically*, so we can only work with the public key. The technique used is to find a way to provide a masked copy of $\mathbf{s}^2$ in much the same way we mask $\mathbf{s}$ in the public key. We do this by generating an extra public key called a *relinearization key*

denoted $\texttt{rlk}$. If we use the same method as with the public key, the obvious way of choosing $\texttt{rlk}$ would be $\texttt{rlk} = (-(\mathbf{a}\cdot\mathbf{s}+\mathbf{e})+\mathbf{s}^2, \mathbf{a})$, with $\mathbf{a} \xleftarrow{\$} R_q, \mathbf{e} \leftarrow \chi$, and as we did with the public key in 2, we can compute $\texttt{rlk}[0] + \texttt{rlk}[1] \cdot \mathbf{s} = \mathbf{s}^2 - \mathbf{e}$. With this in mind, we can see the natural way to choose $\mathbf{c}_0'$ and $\mathbf{c}_1'$ is

$$\mathbf{c}_0' = \mathbf{c}_0 + \texttt{rlk}[0] \cdot \mathbf{c}_2 \tag{3.3}$$

$$\mathbf{c}_1' = \mathbf{c}_1 + \texttt{rlk}[1] \cdot \mathbf{c}_2. \tag{3.4}$$

If we apply the first step of the decryption algorithm, and use the fact that $\texttt{rlk}[0]+\texttt{rlk}[1]\cdot\mathbf{s} = \mathbf{c}^2 - \mathbf{e}$, we get

$$\mathbf{c}_0' + \mathbf{c}_1' \cdot \mathbf{s} = \mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s} + \mathbf{c}_2 \cdot \mathbf{s}^2 - \mathbf{c}_2 \cdot \mathbf{e}. \tag{3.5}$$

The authors of [FV12] acknowledge it is not as simple as providing a masked copy of $\mathbf{s}^2$ in the exact same way as the public key because the multiplication of the error term $\mathbf{e}$ with $\mathbf{c}_2$ could result in a massive increase in error due to $\mathbf{c}_2$ being a random element in R_q (i.e., having norm $q/2$). The first version of relinearization proposed [FV12] is to convert the coefficients of $\mathbf{c}_2$ into some other base T that has smaller norm. We emphasize that this new base T is unrelated to the plaintext value T and is not chosen with respect to t. In fact, as we will see below, it is chosen with respect to the ciphertext modulus q. To avoid confusion, some works use w to represent the base change instead. As the authors of [FV12], we stick with T in this thesis.

Example 3.2. Let $f(x) = 12 + 3x + 9x^2 + x^4$, and $T = 2$. Then we can decompose f as

$$f(x) = (x + x^2 + x^4) \cdot 2^0 + (x) \cdot 2^1 + (1) \cdot 2^2 + (1 + x^2) \cdot 2^3.$$

This idea of decomposing $\mathbf{c}_2$ into a smaller base is used in several schemes [FV12, BGV14, BV14, BEHZ17].

Remark 3.3. The literature often says we can write $\mathbf{c}_2 = \sum_{i=0}^{l} \mathbf{c}_2^{(i)} \cdot T^i$ with $l = \lfloor \log_T(q) \rfloor$. However, we must be careful with how we compute this. We cannot simply choose any representative modulo q or T, but we must remain consistent. That is, we either need to always remain centered modulo q or convert to the least positive residue. We present an algorithm to explicitly show how to perform this base conversion.

Algorithm 1: BaseDecomp($\mathbf{c}_m, T$)

Input: $\mathbf{c}_m$ the degree $m-1$ coefficient of a ciphertext $\texttt{ct} \in R_q[y]$ with $\deg \texttt{ct} >= 2$ and T a positive integer with $T < q$

Output: $(\mathbf{c}_m^{(0)}, \dots \mathbf{c}_m^{(l)})$, where $l = \lfloor \log_T(q) \rfloor$, and $\mathbf{c}_m^{(i)} \in R_T, 0 \leq i \leq l$ such that $\texttt{ct} = \sum_i^l \mathbf{c}_m^{(i)} \cdot T^i$

$l \leftarrow \log_T q$;

for $i = 0$ **to** l **do**

  `/* Operations are done coefficient-wise */`

  $\mathbf{c}_m^{(i)} \leftarrow [\mathbf{c}_m]_T$; `/* Centered reduction of` $\mathbf{c}_m \mod T$ `*/`

  $\mathbf{c}_m \leftarrow (\mathbf{c}_m - \mathbf{c}_m^{(i)})/T$;

end

return $(\mathbf{c}_m^{(0)}, \dots, \mathbf{c}_m^{(l)})$;

When we slice $\mathbf{c}_2$ up in this way, we can make a minor adjustment to the above proposed method of choosing `rlk` to get a new way to relinearize. The idea is to provide a relinearization key as above, but with l varying powers of T all scaling $\mathbf{s}^2$. This gives us a length l key of the form $(-(\mathbf{a}_i \cdot \mathbf{s} + \mathbf{e}_i) + T^i \cdot \mathbf{s}^2, \mathbf{a}_i)$ for $i \in [0, l]$. This now gives new ciphertexts

$$\mathbf{c}_0' = \left[\mathbf{c}_0 + \sum_{i=0}^{l} \texttt{rlk}[i][0] \mathbf{c}_2^{(i)} \right]_q \tag{3.6}$$

$$\mathbf{c}_1' = \left[\mathbf{c}_1 + \sum_{i=0}^{l} \texttt{rlk}[i][1] \mathbf{c}_2^{(i)} \right]_q . \tag{3.7}$$

Now it can easily be seen that this relinearizes exactly as we want. Furthermore, the error does not grow too much because our error terms are now multiplied by $\mathbf{c}_2^{(i)} \in R_T$ where T was chosen so all of the $\mathbf{c}_2^{(i)}$'s have small norm.

Just as we generalized multiplication and addition to ciphertexts of arbitrary lengths, so too can we generalize relinearization. The process is as follows:

- Let $\texttt{ct} = (\mathbf{c}_0, \dots, \mathbf{c}_m)$ be a degree m ciphertext in $R_q[y]$.
- Select a base T as above, with $l = \lfloor \log_T q \rfloor$.
- Generate $m-1$ relinearization keys $\texttt{rlk}_i = (-(\mathbf{a}_j \cdot \mathbf{s} + \mathbf{e}_j) + T^j \cdot \mathbf{s}^i, \mathbf{a}_j), j \in [0, l], i \in [2, m]$.

- Using all of the relinearization keys, and initializing $\mathbf{c}'_0 = \mathbf{c}_0$, and $\mathbf{c}'_1 = \mathbf{c}_1$, we compute

$$\mathbf{c}'_0 = \left[\mathbf{c}'_0 + \sum_{i=0}^{l} \mathtt{rlk}_j[i][0] \cdot \mathbf{c}_j^{(i)}\right]_q \tag{3.8}$$

$$\mathbf{c}'_1 = \left[\mathbf{c}'_1 + \sum_{i=0}^{l} \mathtt{rlk}_j[i][1] \cdot \mathbf{c}_j^{(i)}\right]_q \tag{3.9}$$

$$\tag{3.10}$$

  with $j \in [2, m]$.

The obvious drawback to supporting relinearization of ciphertexts of arbitrary size is that we would need to know *a priori* how large of ciphertexts we will encounter, and we need to handle more keys. At the time of this writing, this seems to be the best known way to handle relinearizing ciphertexts of arbitrary size.

For simplicity, we will work with a ciphertext of degree 2. If we apply the first step of decryption to the relinearized ciphertexts $\mathbf{c}'_0, \mathbf{c}'_1$

$$\mathbf{c}'_0 + \mathbf{c}'_1 \cdot \mathbf{s} = \mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s} + \mathbf{c} \cdot \mathbf{s}^2 - \sum_{i=0}^{l} \mathbf{c}_2^{(i)} \cdot \mathbf{e}_i \tag{3.11}$$

then we expose the noise added from relinearizing. We see that the newly added noise is $\sum_{i=0}^{l} \mathbf{c}_2 \cdot \mathbf{e}_i$. Therefore, the upper bound on the noise introduced by relinearizing is given by $(l+1) \cdot \frac{T}{2} \cdot B \cdot \delta_R$. First, we note that the noise introduced is additive and not at all dependent on the ciphertexts being relinearized. Second, if we want to generalize this to a ciphertext of arbitrary degree $m \geq 2$, we simply need to multiply the above bound by $m - 1$.

3.3 Multiplicative Depth

As we have seen, homomorphic addition increases noise additively, but homomorphic multiplication increases noise significantly more, even when not paired with relinearizing. According to Lemma 3.1, a single multiplication increases the noise by a factor of roughly[1] $2 \cdot t \cdot \delta_R^2 \cdot \|\mathbf{s}\|$. Choosing $\mathbf{s}$ of small norm helps keep this noise growth low, but the ring constant and our

[1]For brevity, we will only consider the dominant term here.

choice of plaintext modulus clearly play a significant role in noise growth. Due to the major role multiplication plays in noise growth compared to addition, the literature often refers to the *depth of a circuit* as the number of multiplications that may be performed before either decryption or a bootstrapping operation is required. We present the multiplicative depth for the classical FV scheme where multiplication only happens with linear ciphertexts in $R_q[y]$.

Theorem 3.4. *Let the notation be as above, along with χ a B-bounded probability distribution. The FV scheme has multiplicative depth L and decrypts properly provided*

$$4 \cdot \delta_R^L \cdot (\delta_R + 1.25)^{L+1} \cdot t^{L-1} < \lfloor q/B \rfloor.$$

Proof. See [FV12, thm. 1]. □

The authors of [FV12] also provide a way of bootstrapping the above scheme to be fully homomorphic. We do not discuss that here and refer the interested reader to their work for more details.

4 Optimizations

Microsoft's SEAL adds several optimizations to the classical BFV scheme we looked at. We will discuss some of those optimizations in this chapter.

4.1 Number Theoretic Transform

In this section we will discuss the number theoretic transform (NTT). This is used to speed up polynomial multiplication.

4.1.1 Discrete Fourier Transform

First we recall the discrete Fourier transform (DFT).

Definition 4.1. Given an input vector $\mathbf{x}_n = (x_0, \ldots, x_{n-1}) \in \mathbb{C}^n$, let $\omega_n = e^{\frac{-2\pi i}{n}}$ be a primitive nth root of unity. Then $\mathcal{F}(\mathbf{x}) = (X_0, \ldots, X_n) \in \mathbb{C}^n$ is the DFT of $\mathbf{x}$ where

$$X_k = \sum_{j=0}^{n-1} x_j \omega_n^k.$$

The inverse DFT is given by

$$x_k = \frac{1}{n} \sum_{j=0}^{n-1} X_j \omega_n^{-k}$$

The DFT can be computed naively in $\mathcal{O}(n^2)$ time. The *fast Fourier transform*(FFT) is a more efficient algorithm that runs in $\mathcal{O}(n \log n)$ time. Polynomial multiplication can performed naively in $\mathcal{O}(n^2)$ time. To speed this up, we take the FFT of a polynomial and perform coordinate-wise multiplication and follow that with an inverse-FFT to achieve polynomial multiplication in $\mathcal{O}(n \log n)$.

4.1.2 Finite Fields

The DFT can be generalized to work with fields besides $\mathbb{C}$ so long as ω_n is a primitive nth root of unity. If we choose our field to be $\mathbb{Z}/p\mathbb{Z}$ with p prime, we know there exists a primitive root $r \in \mathbb{Z}/p\mathbb{Z}$. To compute the DFT of a sequence of n integers, we require a primitive nth root of unity, ω_n. By Dirichlet's theorem on arithmetic progressions, there are infinitely many primes $p \equiv 1 \pmod{n}$, so we can always find p of the form $p = kn + 1$ for some $k \in \mathbb{Z}$. This allows us to set $\omega_n \equiv r^k \pmod{p}$. Now $\omega_n^n \equiv r^{kn} \equiv r^{p-1} \equiv 1 \pmod{p}$. We also have that $\omega_n^m \not\equiv 1 \pmod{p}$ for $m < n$, so ω_n is a primitive nth root of unity as required.

We call this version of the DFT the *number theoretic transform*(NTT). It has similar properties to the DFT. The main use of the NTT in various HE implementations is to speed up the multiplication of ciphertexts.

4.2 FV Residue Number System

The *residue number system* (RNS) variant of the FV scheme was first introduced in [BEHZ17]. Since the ciphertext modulus q does not need to be prime, this scheme uses the Chinese Remainder Theorem (CRT) representation of q. If q is chosen to be a product of "small", coprime moduli, the need for multiprecision arithmetic can be avoided. We can take "small" to mean a machine word. The FV scheme needs to be modified slightly to be compatible with these CRT representations of ciphertexts. The two functions that are not directly compatible are multiplication and decryption of ciphertexts. This is due to the non-positional nature of RNS. We first begin by defining several functions used throughout the scheme.

4.2.1 RNS Decryption

We recall that standard FV decryption consists of four steps:

1. Evaluate $[\mathtt{ct}(\mathbf{s})]_q$
2. Multiply by $\frac{t}{q}$.
3. Round each coefficient to the nearest integer, rounding up in the case of ambiguity.

4. Apply $[\cdot]_t$ (reduce coefficients to the range $[-t/2, t/2)$).

The first and third steps can be done in RNS, but the other two steps need modifications to work with RNS.

Definition 4.2. Let t, q, R, R_q as above. We define *Division and Rounding* in $R[Y]$ as

$$\texttt{DR}_i \ : \ \texttt{ct} = \sum_{j=0}^{i} \texttt{ct}[j] Y^j \mapsto \sum_{j=0}^{i} \left\lfloor \frac{t}{q} \texttt{ct}[j] \right\rceil Y^j.$$

Ciphertexts are in R_q while messages are in R_t. The fourth step listed above involves taking ciphertexts from R_q into R_t. It is easy to see that we cannot simply apply $[\cdot]_t$ to each component in RNS form. Instead, the idea is to use the Chinese Remainder Theorem to convert an element from $\prod_{i=1}^{n} R_{q_i}$ to an element in R_q, then reduce that element modulo t_i for each t_i in $t = t_1 t_2 \dots t_m$.

Definition 4.3. The *fast base conversion* is a map

$$\texttt{FastBconv} : R_q \to R_t$$

where $q = \prod_{i=1}^{n} q_i, t = \prod_{i=1}^{m} t_i$, and $z = (z_1 = z \mod q_1, \dots, z_n = z \mod q_n)$. The goal is to write $z = (z'_1 = z \mod t_1, \dots, z'_m = z \mod t_m)$. Given z, q, and t, the algorithm is

$$z'_j = \left| \underbrace{\sum_{i=1}^{n} \left| z_i \left(\frac{q}{q_i} \right)^{-1} \right|_{q_i} \times \frac{q}{q_i}}_{z + \alpha q} \right|_{t_j} \tag{4.1}$$

and is applied coefficient-wise.

It is worth noting that (4.1) does not give exactly z, but instead gives an extra multiple of q. This is because the q_is are typically chosen to be a word length, and so multiprecision division is required, which is computationally expensive. The authors work around this by dealing with this approximate result and adjusting it in a way that is faster than multiprecision reduction modulo q.

The division and rounding steps do not work correctly in RNS. To get around this, the authors of [BEHZ17] replace rounding with flooring. This obviously introduces an error term. Lemma 1 of [BEHZ17] shows how to combine this error term with the error term introduced from `FastBconv`, and lemma 2 of [BEHZ17] gives a method they call a γ-correction technique.

5 Ring Expansion Factor

The ring expansion factor δ_R was first introduced in [LM06]. When we reduce a ring element modulo an ideal generated by a polynomial f, it is possible for the infinity norm to increase. The maximum size it can increase by is captured by δ_R. For the ring we typically work with in ring-LWE, the maximum such growth is equal to the degree of f. As we have seen in the previous chapters, this factor is important when it comes to providing an upper bound on the noise of ciphertexts, and is also used to determine the maximum noise growth when multiplying ciphertexts. For certain polynomials, it is possible for this growth to be exponential, and in fact we will look at some examples where this is the case.

In this chapter, we will look at ring expansion factors for arbitrary polynomials. Furthermore, we will look at the growth of multiplying arbitrary polynomials and reducing them modulo arbitrary ideals as well as those of the form $x^n + 1$. We will provide some numerical experiments which suggest that $\delta_R = n$ for the quotient ring $\mathbb{Z}[x]/\langle x^n + 1\rangle$ is a significant overestimate of the actual growth of the norm after reduction. It remains an open question of how one might be able to exploit such average case improvements in cryptographic applications where guaranteed correctness of decryption is a typical requirement.

5.1 Norm Growth

In the standard homomorphic encryption schemes, we will often want to multiply elements together. Since we are working in a polynomial quotient ring, it is standard to reduce our products to their canonical representatives. The result of this reduction can cause the norm to grow substantially. As shown in lemma 2.1, when working with the quotient ring $\mathbb{Z}[x]/\langle x^n + 1\rangle$, the maximum a norm can grow by is equal to n. We recall that reducing the product is done by using the fact that $x^n \equiv -1 \mod x^n + 1$. It is easy to cook up an ideal where the norm can grow exponentially. A simple example is taking the ideal generated by $x^n - cx^{n-1}$

for some constant c. Then $x^n \equiv cx^{n-1}$ and it is easily seen that $x^{n+k} \equiv c^{k+1}x^{n-1}$ for any positive integer k. In [LM06], the authors provide bounds on the expansion factor for a given polynomial.

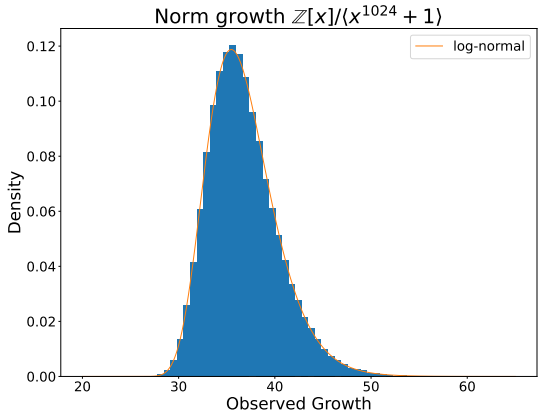
By sampling random elements in the ring $\mathbb{Z}[x]/\langle x^n + 1\rangle$ with n a power of two, we can try to see how elements grow from multiplication and reduction. The motivation behind this is to see how close to δ_R the norms of elements actually grow by. We see that the actual norm growth of these trials appears to be bounded as $\mathcal{O}(n^{0.58})$ on average.

If we can prove, that with great certainty, the likely growth of norms is much less than the worst-case estimate of $\delta_R = n$, then on average we can decrease all of the noise bounds we showed in chapter 3 by choosing a smaller value than n. This has the effect of increasing the number of levels of multiplication we can perform.

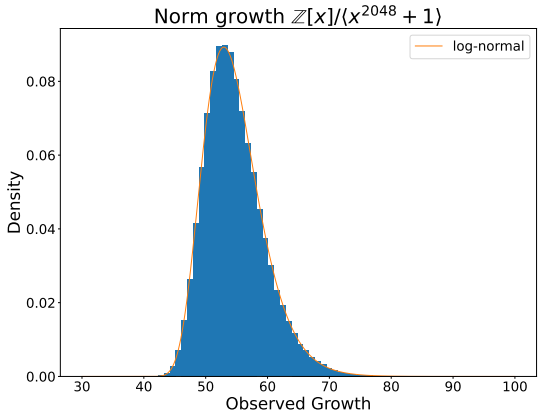
Histograms of the results of the above experiment are shown in Figure 5.1. When running 100,000 trials of multiplying polynomials and reducing them modulo $x^n + 1$ for various n, fitting various pdfs to the resultant norm growths using SciPy [VGO+20], selecting the best candidate via the KS test (see B.2), we see that the norm growth appears to exhibit behaviour similar to that of a log-normal distribution; for a definition of the log-normal distribution along with a description of the parameters, see appendix B.1. Using the Python library, SciPy [VGO+20], we computed what the expected maximum norm growth would be for the 99.9th percentile. As a function of n, we observe that 99.9th percentile corresponds to $\approx n^{0.58}$.

As seen in table 5.1, in most instances the observed norm growth in $\mathbb{Z}[x]/\langle x^n + 1\rangle$ appears to exhibit log-normal behaviour. This is seen by comparing the values in the D-stat column to the critical values listed at the top. The null hypothesis H_0 is that the distribution is a log-normal distribution. We reject H_0 when the D-stat is greater than the critical value for a given confidence level. We observe that the hypothesis H_0 is sometimes rejected, implying the norm growth does not come from a log-normal distribution. More experimentation may be needed to say with confidence this comes from a log-normal distribution. For example, one could try running these experiments using different software or libraries, comparing against a different set of distributions, using different statistical tests, or using a different range for sampling coefficients.

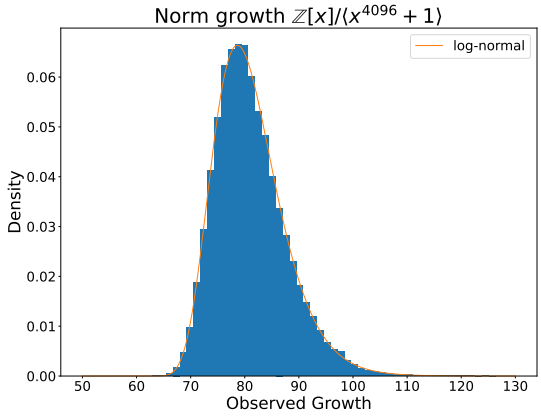
Remark 5.1. The outliers in the plots seem to occur for small values of n, even after re-



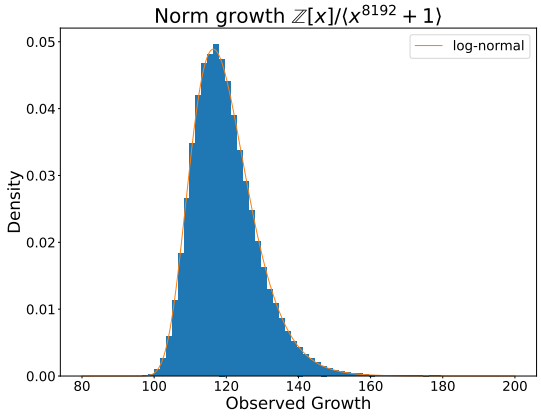
(a) $\mathbb{Z}[x]/\left\langle x^{1024}+1\right\rangle$



(b) $\mathbb{Z}[x]/\left\langle x^{2048}+1\right\rangle$



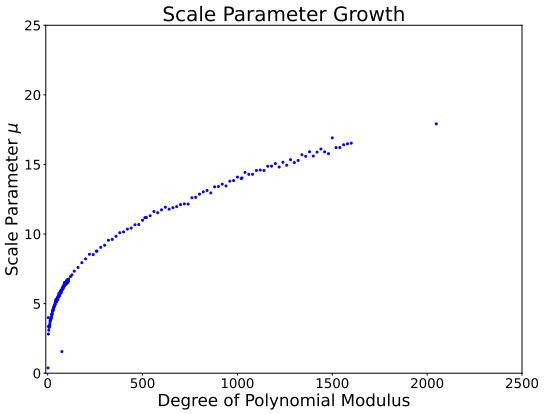
(c) $\mathbb{Z}[x]/\left\langle x^{4096}+1\right\rangle$



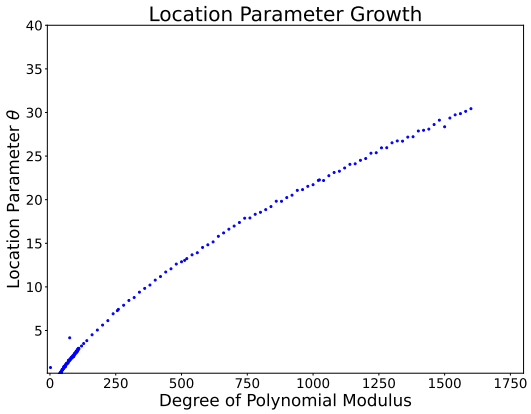
(d) $\mathbb{Z}[x]/\left\langle x^{8192}+1\right\rangle$

Figure 5.1: Observed norm growth for power-two cyclotomic polynomials

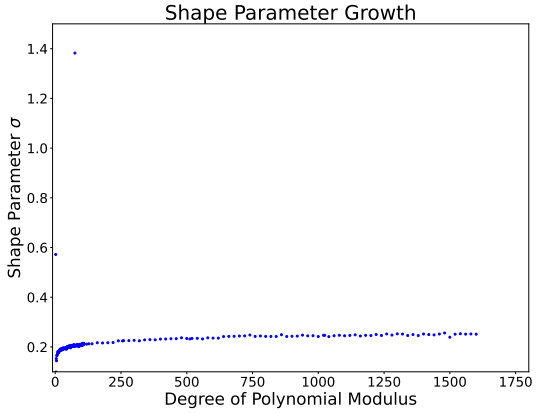
running the trials several times. This could be due the range used for sampling coefficients, or that this distribution observation starts to break down at smaller values of n. The outlier in figure 5.3 could be for the fact that we are measuring the inherently noisy part of the distribution.



(a) Scale growth



(b) Location growth



(c) Shape growth

Figure 5.2: Growth of estimated log-normal parameters

5.2 FV Noise Growth

In this section we will look at the noise growth from multiplication of arbitrary ciphertexts in the classical FV scheme.

If we have some ciphertext $\mathtt{ct} = (\mathbf{c}_0, \ldots, \mathbf{c}_n)$ encrypting some plaintext $\mathbf{m} = [\mathbf{m}_1 \ldots \mathbf{m}_n]_t$

Table 5.1: KS test statistics for norm growth in polynomial quotient rings of the form $\mathbb{Z}[x]/\langle x^n + 1\rangle$ under a log-normal pdf assumption for 100,000 trials.

α	D_{crit}
0.10	0.00386
0.05	0.00429
0.01	0.00514

Modulus	D-Stat	p-value	99.9th Percentile	Modulus	D-Stat	p-value	99.9th Percentile
x^2+1	0.03819	< 0.001	2.97	$x^{940}+1$	0.00384	0.104	49.9
x^4+1	0.00782	< 0.001	3.13	$x^{960}+1$	0.00405	0.074	50.5
x^8+1	0.00319	0.258	4.28	$x^{980}+1$	0.00302	0.319	51.0
$x^{16}+1$	0.00141	0.988	5.92	$x^{1024}+1$	0.00431	0.047	52.3
$x^{32}+1$	0.00269	0.463	8.33	$x^{1040}+1$	0.00257	0.521	52.6
$x^{64}+1$	0.00183	0.887	11.9	$x^{1100}+1$	0.00369	0.129	54.3
$x^{128}+1$	0.00224	< 0.001	17.1	$x^{1180}+1$	0.00435	0.044	56.4
$x^{256}+1$	0.00245	< 0.001	24.8	$x^{1240}+1$	0.00508	0.011	57.8
$x^{320}+1$	0.00280	0.411	27.9	$x^{1280}+1$	0.00294	0.350	58.9
$x^{360}+1$	0.00400	0.080	29.8	$x^{1360}+1$	0.00383	0.105	60.8
$x^{400}+1$	0.00381	0.109	31.5	$x^{1400}+1$	0.00519	0.009	61.8
$x^{480}+1$	0.00409	0.069	34.8	$x^{1440}+1$	0.00340	0.196	62.7
$x^{500}+1$	0.00264	0.488	35.5	$x^{1480}+1$	0.00502	0.012	63.9
$x^{512}+1$	0.00357	0.154	35.9	$x^{1500}+1$	0.00487	0.017	63.7
$x^{560}+1$	0.00459	0.029	37.7	$x^{1520}+1$	0.00393	0.089	64.6
$x^{640}+1$	0.00366	0.137	40.6	$x^{1580}+1$	0.00302	0.320	66.0
$x^{700}+1$	0.00323	0.246	42.7	$x^{1600}+1$	0.00335	0.209	66.3
$x^{780}+1$	0.00332	0.218	45.2	$x^{2048}+1$	0.00404	0.075	75.9
$x^{860}+1$	0.00287	0.381	47.5	$x^{4096}+1$	0.00456	0.031	110.1
$x^{900}+1$	0.00333	0.215	48.6	$x^{8192}+1$	0.00364	0.139	159.6

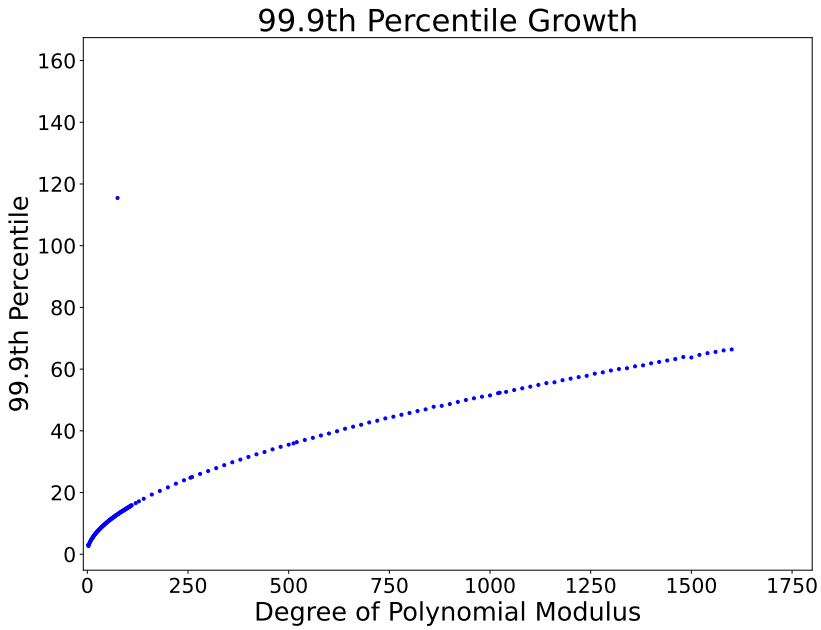


Figure 5.3: Observed 99.9th percentile of the norm growth for 100,000 trials

and the secret key $\mathbf{s}$ needed to decrypt $\mathtt{ct}$, then it turns out we can easily measure the noise associated with that particular ciphertext. Recall

$$\mathtt{ct}(\mathbf{s}) = \left[\sum_{i=0}^{n} \mathbf{c}_i \cdot \mathbf{s}^i\right]_q = \left\lfloor \frac{q}{t} \right\rfloor \cdot \mathbf{m} + \mathbf{v},$$

then we can isolate the noise simply by writing the above as

$$\left[\sum_{i=0}^{n} \mathbf{c}_i \cdot \mathbf{s}^i - \left\lfloor \frac{q}{t} \right\rfloor \cdot \mathbf{m}\right]_q = \mathbf{v}. \tag{5.1}$$

For a Sage implementation of measuring noise using the above method, along with a toy implementation of the FV scheme, see appendix A.

We can design an experiment similar to that in 5.1 by performing multiplications of arbitrary ciphertexts and measuring the growth of their noise by equation (5.1). The detailed method used is as follows:

1. Select the degree n of the polynomial modulus to work with.

2. Fix the error distribution χ with deviation $\sigma = 3.19$ so that it is B-bounded with $B \approx 19.14$.

Table 5.2: Noise growth statistics and bounds for a single multiplication without relinearization in toy FV implementation under the BKZ sieve cost model

n	$\log_2 q$	$\log_2 t$	Maximal growth (bits)	Observed growth (bits)
1024	27	1	42	11
2048	54	3	49	16
4096	109	25	80	37
8192	218	54	138	67

3. Select a bit-length for q according to the suggested parameters in [ACC+18] for n under the classical BKZ sieve cost model using the uniform ternary distribution for key selection.

4. Select a plaintext modulus t such that the noise growth from a single multiplication will still allow correct decryption.[1]

5. Generate random plaintext elements, encrypt them, perform a multiplication, and record the noise growth *relative to the ciphertext with the largest norm.*

We are interested in how large the noise grows relative to the norm of a ciphertext. Figure 5.4 shows histograms for the above experiment with $n = 1000$ trials for each quotient ring, and 5.2 shows the maximal noise growth observed under the given parameters along with the maximal possible noise growth given in lemma 3.1. Examining the results in table 5.2, we see that the maximal noise growth is 2-4 times that of the maximal observed noise growth under a fixed set of parameters.

Using the experimental results in section 5.1, we can replace the ring constant $\delta_R = n$ with the value $\gamma_R = n^{0.58}$, and can recompute the worst-case bounds for multiplication seen in chapter 3. We emphasize here that working with γ_R is based only on experimental results, and that we have not proven such a value is even justified. Table 5.3 shows the multiplicative depth of the classical FV scheme using both δ_R and γ_R with the suggested parameters in [ACC+18]

[1]There is no recommended way to select this parameter. It is dependent on the individual use case.

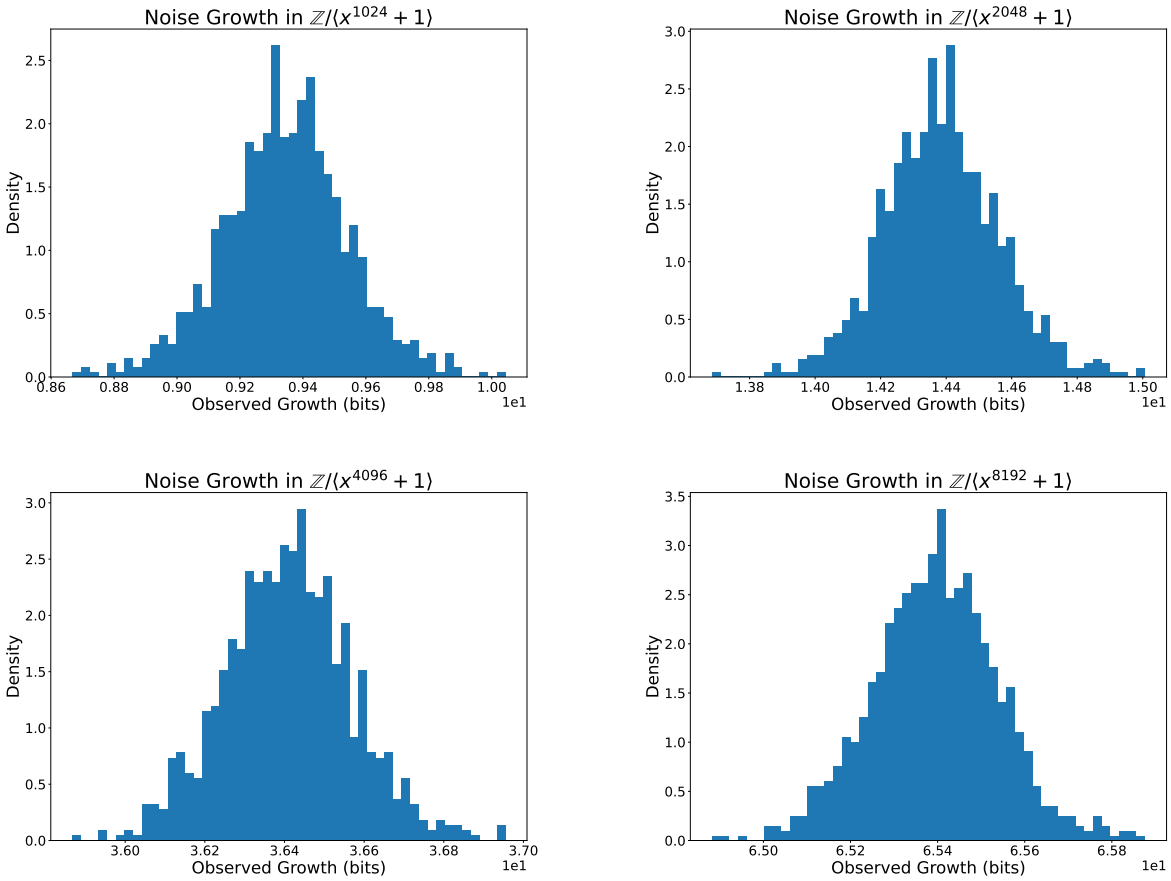


Figure 5.4: Observed noise growth of 1000 trials of a single multiplication without relinearization in toy FV implementation. See Table 5.2 for maximal growth values with fixed parameters observed here.

Table 5.3: Maximum multiplicative depth for classical FV with δ_R and γ_R using recommended parameters for 128 bit security in the classical case

n	$\log_2 q$	δ_R	γ_R	δ_R-depth	γ_R-depth
1024	27	1024	55.72	0	1
2048	54	2048	83.29	1	3
4096	109	4096	124.50	3	6
8192	218	8192	186.11	7	13
16384	438	16384	278.20	14	26
32768	881	32768	415.87	28	49

for 128-bit security in the classical model. We recall from theorem 3.4 that the plaintext modulus t has a role in determining the multiplicative depth, but we have dropped it from these calculations since it is a user-defined parameter not impacting security. The parameter t was left in table 5.2 because it was a required parameter in the toy FV implementation. We see in table 5.3 that as long as t is chosen conservatively so as not to contribute too much noise, we can approximately double the multiplicative depth of a classical FV circuit under the γ_R assumption.

6 Summary and Future Work

Since Gentry's thesis in 2009, there have been significant contributions to the study of homomorphic encryption cryptosystems. The hardness results are primarily based on the LWE scheme and general, random lattices. As mentioned in chapter 1, the popular HE libraries are based on the ring-LWE problem. Currently, the best-known attacks on ring-LWE-based cryptosystems are not significantly better than attacks on general LWE-based ones [ACC+18]. In section 2.3 we provided background for general cyclotomic fields, but in this thesis we have primarily looked at power-two cyclotomic fields. Additionally, this is what the state-of-the-art implementations use. While many of the results generalize to the general cyclotomic case, it is not currently recommended to use non-power-two cyclotomics [ACC+18].

Some of the main ring-LWE schemes [BGV14, FV12, BLLN13] are reliant on the ring expansion factor δ_R for determining circuit depth. As seen in the experiments outlined in chapter 5, the average-case growth of multiplication of ring elements appears to exhibit behaviour much better than the worst-case of $\delta_R = n$. More work is needed to better understand the noise growth of ring elements in power-two cyclotomics. It would be desirable to exploit this average-case growth to increase circuit depth. We once again remark that circuit depth is also dependent on the plaintext modulus t although security is not affected. In [Pla18], they remark about the automatic parameter selection of SEAL without giving any concrete method for choosing t. However, in issue #232 on the SEAL GitHub repository, one of the lead developers states that the automatic parameter selection was removed due to bugs.

In section 4.2 we introduced the RNS variant of the FV scheme [BEHZ17] used in SEAL. At the time of this writing, there is no bootstrapping operation that is compatible with this variant. The bootstrapping operation of the classical FV scheme appears to be incompatible with the RNS variant. Microsoft seems to have chosen this variant because of its increased

performance, but it would be beneficial if it could be made into a fully homomorphic scheme while still retaining the performance benefits.

References

[ACC+18] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

[Ajt96] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 99–108, New York, NY, USA, 1996. Association for Computing Machinery.

[Ajt98] Miklós Ajtai. The shortest vector problem in l2 is np-hard for randomized reductions. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 10–19, 1998.

[APS15] Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[BEHZ17] Jean-Claude Bajard, Julien Eynard, M. Anwar Hasan, and Vincent Zucca. A full rns variant of fv like somewhat homomorphic encryption schemes. In Roberto Avanzi and Howard Heys, editors, *Selected Areas in Cryptography – SAC 2016*, pages 423–442, Cham, 2017. Springer International Publishing.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 6(3):Art. 13, 36, 2014.

[BLLN13] Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *Cryptography and coding*, volume 8308 of *Lecture Notes in Comput. Sci.*, pages 45–64. Springer, Heidelberg, 2013.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, page 575–584, New York, NY, USA, 2013. Association for Computing Machinery.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 868–886, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM J. Comput.*, 43(2):831–871, 2014.

[CGGI19] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 2019.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[Coh13] Henri Cohen. *A course in computational algebraic number theory*, volume 138. Springer Science & Business Media, 2013.

[DF04] David Steven Dummit and Richard M Foote. *Abstract Algebra*, volume 3. Wiley Hoboken, 2004.

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

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[HPS08] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. *An introduction to mathematical cryptography*, volume 1. Springer, 2008.

[Lai17] Kim Laine. Simple encrypted arithmetic library 2.3. 1. *Microsoft Research https://www. microsoft. com/en-us/research/uploads/prod/2017/11/sealmanual-2-3-1. pdf*, 2017.

[LM06] Vadim Lyubashevsky and Daniele Micciancio. Generalized compact knapsacks are collision resistant. In *International Colloquium on Automata, Languages, and Programming*, pages 144–155. Springer, 2006.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 1–23, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[MJ51] Frank J Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[Neu13] Jürgen Neukirch. *Algebraic number theory*, volume 322. Springer Science & Business Media, 2013.

[Pla18] Rachel Player. *Parameter selection in lattice-based cryptography*. PhD thesis, Royal Holloway, University of London, 2018.

[RAD+78] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC'05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 84–93. ACM, New York, 2005.

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

[VGO+20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[Was97] Lawrence C Washington. *Introduction to cyclotomic fields*, volume 83. Springer Science & Business Media, 1997.

Appendix A

Toy FV Sage Implementation

```
def inf_norm(f):
    return max(abs(max(f)),abs(min(f)))

def reduce_and_centre(f, coeff_mod):
    coeff_min, coeff_max = ceil(-coeff_mod/2), (coeff_mod-1)//2
    g = 0
    for i in range(f.lift().degree() + 1):
        c = f[i] % coeff_mod
        if c > coeff_max:
            c -= coeff_mod
        g += c*x^i
    return g

def least_positive_residue(f, coeff_mod):
    return sum((f[i] % coeff_mod)*x^i for i in range(f.lift().degree
       () + 1))

class Ciphertext(object):
    def __init__(self,coeff_modulus, pt_modulus, ring, *elements):
        if len(elements) == 1:
            raise ValueError("Ciphertexts cannot consist of only 1
               element")
        self.elements = list(elements)
        self.coeff_modulus = coeff_modulus
        self.pt_modulus = pt_modulus
        self.R = ring

    def __add__(self, c2):
        if len(self) != len(c2):
            raise ValueError("Ciphertexts need to be the same size")
        temp = [self.R(reduce_and_centre(sum(x),self.coeff_modulus))
            for x in zip(self.elements, c2.elements)]
        return Ciphertext(self.coeff_modulus, self.pt_modulus, self.
           R, *temp)

    def __sub__(self, c2):
        if len(self) != len(c2):
            raise ValueError("Ciphertexts need to be the same size")
        temp = [self.R(reduce_and_centre(x[0] - x[1]),self.
           coeff_modulus) for x in zip(self.elements, c2.elements)]
        return Ciphertext(self.coeff_modulus, self.pt_modulus, self.
```

```
            R, *temp)

    def __mul__(self, c2):
        q = self.coeff_modulus
        t = self.pt_modulus
        # this allows us to multiply by an integer plaintext, since
           multiplication
        # by an integer works the usual way.
        if isinstance(c2, Integer):
            temp = [reduce_and_centre(c2*c, self.coeff_modulus) for
               c in self.elements]
            return Ciphertext(q, t, self.R, *[c2*self[i] for i in
               range(len(self))])
        elif isinstance(c2, Ciphertext):
            temp = [0]*(len(self) + len(c2) - 1)
            for i in range(len(self)):
                for j in range(len(c2)):
                    temp[i + j] += self[i]*c2[j]
            c1c2 = []
            for c in temp:
                c1c2.append(self.R(
                    reduce_and_centre(
                        self.R(sum(round(t/q*c[i])*x^i for i in
                           range(c.lift().degree() + 1))), q)))
            return Ciphertext(q, t, self.R, *c1c2)
        else:
            return TypeError("c2 is not of type Ciphertext or
               Integer")

    def __rmul__(self, c2):
        if isinstance(c2, Integer) or isinstance(c2, Ciphertext):
            return self*c2
        else:
            raise TypeError("c2 is not of type Ciphertext or Integer
               ")

    def __getitem__(self, i):
        return self.elements[i]

    def __len__(self):
        return len(self.elements)

class Plaintext(object):
    def __init__(self,coeff_modulus,ring,msg):
        self.coeff_modulus = coeff_modulus
        self.R = ring
        self.msg = self.R(msg)
```

```
    def __add__(self, f):
        temp = reduce_and_centre(self.msg + f.msg,self.coeff_modulus
           )
        return Plaintext(self.coeff_modulus, self.R, temp)

    def __sub__(self, f):
        temp = reduce_and_centre(self.msg - f.msg, self.
           coeff_modulus)
        return Plaintext(self.coeff_modulus, self.R, temp)

    def __mul__(self, f):
        if isinstance(f, Integer):
            tmp = reduce_and_centre(self.msg*f, self.coeff_modulus)
            return Plaintext(self.coeff_modulus, self.R, tmp)
        elif isinstance(f, Plaintext):
            temp = reduce_and_centre(self.msg*f.msg, self.
               coeff_modulus)
            return Plaintext(self.coeff_modulus, self.R, temp)
        else:
            raise TypeError("f is not of type Plaintext or Integer")

    def __rmul__(self, f):
        if isinstance(f, Integer) or isinstance(f, Plaintext):
            return self*f
        else:
            raise TypeError("f is not of type Plaintext or Integer")

    def __getitem__(self, i):
        return self.msg[i]

    def __eq__(self, f):
        return self.msg == f.msg

    def lift(self):
        return self.msg.lift()

class FV(object):

    # parms = [poly_modulus_degree, coeff_modulus, plain_modulus]
    # in FV, these are [n, q, t].
    def __init__(self, parms):
        self.n, self.q, self.t, self.T = parms
        R.<x> = PolynomialRing(ZZ)
        self.R = R.quotient(x^self.n + 1)
        self.l = floor(log(self.q, self.T))

        # The following parameters are defaults used in SEAL
        self.std_dev = 3.19
```

```
        self.B = 6*self.std_dev
        self.chi = RealDistribution('gaussian', self.std_dev)

    def gen_secret_key(self):
        s = sum(randint(-1,1)*x^i for i in range(self.n))
        return self.R(s)

    def gen_pub_key(self, s):
        a = self._sample_uniform()
        e = self._sample_chi()
        return [self.R(reduce_and_centre(-a*s + e, self.q)), a] # we
            need to reduce that first part modulo q and centre it

    def gen_relin_key_v1(self, s):
        rlk = []
        s_squared = s*s
        for i in range(self.l + 1):
            a_i = self._sample_uniform()
            e_i = self._sample_chi()
            rlk.append((self.R(reduce_and_centre((-(a_i*s + e_i) + (
                self.T^i)*s_squared),self.q)), a_i))
        return rlk

    def encrypt(self, pk, m):
        u, e1, e2 = self._sample_chi(), self._sample_chi(), self.
            _sample_chi()
        p0, p1 = pk
        c0 = self.R(reduce_and_centre(p0*u + e1 + floor(q/t)*m.msg,
            self.q))
        c1 = self.R(reduce_and_centre(p1*u + e2, self.q))
        return Ciphertext(self.q, self.t, self.R, c0, c1)

    def decrypt(self, s, ct):
        m_noisy = (t/q)*self.R(reduce_and_centre(sum(ct[i]*s^i for i
            in range(len(ct))), self.q))
        # To get rid of the noise, we round each coefficient.
        m_bar = self.R([round(coeff) for coeff in m_noisy])
        return Plaintext(self.t, self.R, reduce_and_centre(m_bar,
            self.t))

    def relinearize(self, rlk, ct):
        if len(ct) != 3:
            raise ValueError("Can only relinearize ciphertexts with
                3 elements")
        c0, c1, c2 = ct
        c2_sliced = self._base_convert(c2)
        if len(c2_sliced) != len(rlk):
            raise ValueError("c2_sliced is not the same length as
```

```
                rlk for some reason.")

        c0_prime = self.R(reduce_and_centre(c0 + sum(rlk[i][0]*
            c2_sliced[i] for i in range(len(rlk))), self.q))
        c1_prime = self.R(reduce_and_centre(c1 + sum(rlk[i][1]*
            c2_sliced[i] for i in range(len(rlk))), self.q))
        return Ciphertext(self.q, self.t, self.R, c0_prime, c1_prime
            )

    # Used for creating a specific plaintext message. This should
       probably go in the Plaintext class,
    # but I'd need a nicer way to pass our quotient ring to the
       Plaintext class.
    def plain(self, msg):
        return Plaintext(self.t, self.R, msg)

    def random_plain(self):
        coeff_min, coeff_max = ceil(-self.t/2), (self.t-1)//2
        msg = self.R(sum(randint(coeff_min, coeff_max)*x^i for i in
            range(self.n)))
        return Plaintext(self.t, self.R, msg)

    def _sample_uniform(self):
        coeff_min, coeff_max = ceil(-self.q/2), (self.q-1)//2
        return self.R(sum(randint(coeff_min,coeff_max)*x^i for i in
            range(self.n)))

    def _sample_chi(self):
        return self.R(sum(round(self.chi.get_random_element())*x^i
            for i in range(self.n)))

    # Decomposes a polynomial in R_q to one in R_T.
    def _base_convert(self, f):
        rebased = []
        f = self.R(least_positive_residue(f, self.q))
        T = self.T
        for i in range(self.l + 1):
            tmp = self.R(reduce_and_centre(self.R(sum(((f[j])%T)*x^j
                for j in range(self.n))), self.T))
            rebased.append(tmp)
            f = (f - tmp)*(1/T)
        return rebased

# Takes as input the secret key s, plaintext pt, and its
   corresponding ciphertext ct.
def noise(s, pt, ct, R):
    q, t = ct.coeff_modulus, pt.coeff_modulus
    n = len(ct)
```

```
Delta = floor(q/t)
cs = R(sum(ct[i]*s^i for i in range(n)))
v = R(reduce_and_centre(cs - R(Delta*(reduce_and_centre(pt.msg,
   t))), q))
return inf_norm(v)
```

Appendix B

Statistics

This appendix serves as a refresher for some basic results and techniques from statistics. Information on the log-normal distribution may be found in most undergraduate textbooks on statistics. We will also use the Kolmogorov-Smirnov test [MJ51]. The methods below are implemented in most statistics libraries, and in this thesis we use the ones implemented in SciPy [VGO+20].

B.1 Log-normal Distribution

Let Y be a random variable that is normally distributed with mean μ and standard deviation σ. If X is a random variable such that $X = \exp(Y)$, then we say X is a *log-normal distribution* parameterized by μ and σ.

With this parameterization for X, we can find its probability distribution function (pdf) $f_X(x)$. If we let $F_X(x)$ be the cumulative distribution function (cdf) of X, then we recall

$$F_X(x) = \Pr(X \leq x) = \int_{-\infty}^{x} f_X(t)\, dt.$$

We note that X only takes positive values. Then we have

$$\begin{aligned} F_X(x) &= \Pr(X \leq x) \\ &= \Pr(\ln X \leq \ln x) \\ &= \Pr(Y \leq \ln x) \\ &= \int_{-\infty}^{\ln x} f_Y(y)\, dy \end{aligned}$$

where $f_Y(y)$ is the pdf of the normal distribution on Y given by

$$f_Y(y) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right).$$

Using the Fundamental Theorem of Calculus to take the derivative of $F_X(x)$ gives

$$f_X(x) = \frac{1}{\sigma x\sqrt{2\pi}} \exp\left(-\frac{(\ln x-\mu)^2}{2\sigma^2}\right).$$

It is easily seen that the median and standard deviation of X are e^{μ} and e^{σ} respectively. It is important to not confuse the mean with the median, as the mean of X is actually given by $e^{\mu+\frac{\sigma^2}{2}}$.

An alternative parameterization is to take $s = e^{\mu}$ to be the shape parameter, σ to be the scale parameter, and λ to be the location parameter.

When one wants to compute percentiles of a log-normal distribution, a naïve and often common approach is to compute the percentile of its corresponding normal distribution, and then back-transform the result via exponentiation; this is the implementation in [VGO+20].

B.2 Kolmogorov-Smirnov Test

The Kolmogorov-Smirnov test (KS test) is a goodness-of-fit test used to determine if a sample is from a particular continuous probability distribution. The KS test is a *non-parametric test*. In other words, it makes no assumption about the underlying distribution. Non-parametric tests are often used when we do not know what distribution the samples are coming from.

The KS test measures the differences between the empirical cdf $F_n(x)$ and the cdf $F(x)$ of the reference distribution to be tested. The KS test returns a test statistic

$$D = \sup_x |F_n(x) - F(x)| .$$

For a given significance level α, we can find critical values for D. In practice, these critical values are found in software and so we do not present them here. If D is less than the critical value for α, we fail to reject the null hypothesis H_0 that the samples are from the distribution, and we reject H_0 if D exceeds the critical value.