CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Homomorphic Encryption: Computing on encrypted data

Master's thesis in Computer science and engineering

NALA YEHE

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Homomorphic Encryption: Computing on encrypted data

NALA YEHE

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Homomorphic Encryption: Computing on Encrypted Data

NALA YEHE

© NALA YEHE, 2023.

Supervisor: Uddipana Dowerah, CSE
Examiner: Elena Pagnin, CSE

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2023

Homomorphic Encryption: Computing on Encrypted Data
NALA YEHE
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Homomorphic encryption is the ability to compute mathematical functions on encrypted data without decryption. The output of the computations is itself in encrypted form which when decrypted is identical to the output had the computation been performed on the plaintexts directly. A homomorphic encryption scheme is said to be fully homomorphic if mathematical functions of arbitrary complexity can be computed on encrypted data. Fully homomorphic encryption (FHE) has several applications in security and privacy since computations can be outsourced to a third (untrusted) party while the data is still in encrypted form. For the usability of FHE in various applications, it is needed that these schemes are thoroughly studied and implemented to be able to use in practice. However, not all schemes in the literature are implemented yet. In this thesis, we implemented the basic encryption operation and homomorphic addition property of the FHE scheme, proposed by Dowerah et.al in her PhD dissertation. Further, we explored the possibility of extending the scheme to a multi-key setting for the basic encryption scheme. Multi-key homomorphic encryption (MK-HE) enables separate parties to utilize distinct keys for encryption, different from traditional homomorphic encryption scheme which assesses the arithmetic circuits of ciphertexts encrypted with the same key.

Keywords: fully homomorphic encryption, cryptography, learning with errors, multi-key FHE.

Acknowledgements

First and foremost, I am extremely grateful to my thesis supervisor, Uddipana Dowerah, for her invaluable guidance, support, and mentorship throughout this thesis journey. Her expertise and dedication have been instrumental in shaping the direction of this research and enhancing its quality. I am truly fortunate to have had such a knowledgeable and supportive supervisor.

I would also like to extend my sincere thanks to Elena Pagnin, my thesis advisor, for her constructive feedback. Her expertise in the field has been invaluable in shaping the development of this thesis work.

I am deeply appreciative of my family and my boyfriend for their unwavering support, love, encouragement and understanding throughout my studying pursuits.

Nala Yehe, Gothenburg, 2023

Contents

List of Figures

List of Tables

1
Introduction

Due to the exponential growth of digital data and communication channels, data security and privacy have become increasingly important challenges in modern life. The history of using encryption techniques to safeguard confidential data can be traced to ancient times, such as Morse code and Caesar cipher. Encrypting and decrypting messages are the studies of mathematical methods for secure communication when third parties or adversaries are present, which is known as cryptography[1]. It involves creating and analyzing algorithms and protocols that prevent unauthorized access to information and ensure confidentiality, data integrity, authentication, and non-repudiation [1]. The confidentiality, data integrity, and authenticity of information or data that is transmitted over communication networks or stored in storage devices are guaranteed by cryptographic algorithms and protocols.

Cryptography refers to the technique of encryption, decryption, and transforming computer information, which has two branches, one mainly studies the methods of transforming information to protect the messages' security and the other one called cryptanalysis mainly study how to decipher messages. We will focus on encryption schemes in this report. In an encryption system, an encryption key is used to encrypt the messages into ciphertext, and then decrypt the ciphertext back to the original messages through a decryption key at the other end. The general procedure of encryption and decryption is shown in Figure 1.1.

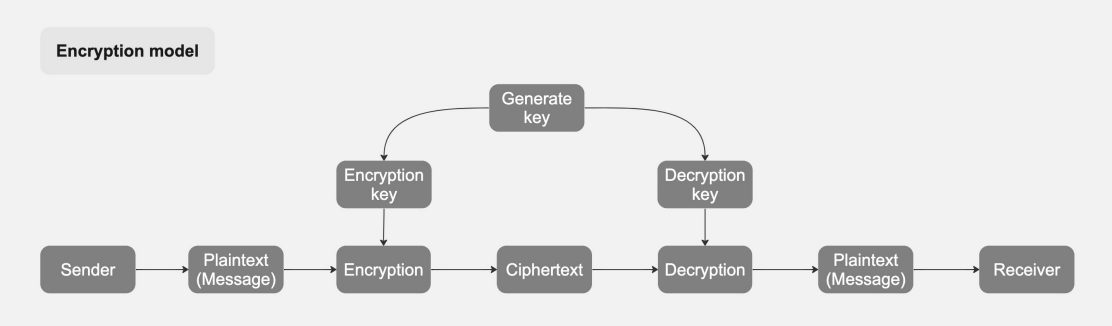

Figure 1.1: Encryption Model

When the encryption key is the same as the decryption key, it is a secret key encryption model. When the encryption key is the receiver's public key and the decryption key is the receiver's private key, it is a public key encryption model.

Encryption is the process of transforming plaintext data into ciphertext in order to protect it from unauthorized access, and can be decrypted using a secret key.

The data can only be accessed by those having the correct key, and can correctly decrypt it to get the original data. However, computing on encrypted data or any other operations other than decryption is extremely difficult since any additional operations could lead to incorrect decryption or failure, thereby posing a significant challenge in performing computations on encrypted data. To overcome this issue, Homomorphic Encryption (HE) schemes were proposed [2]. Homomorphic encryption allows the performing of mathematical operations on encrypted data without the need for decryption. The result of these computations remains encrypted but can be decrypted to obtain the same output as if the operations were performed on the original, unencrypted data. A homomorphic encryption scheme is considered to be fully homomorphic if mathematical functions of arbitrary complexity can be computed on encrypted data. Fully Homomorphic Encryption (FHE) schemes have a lot of potentials to improve data security and privacy in a variety of applications, such as healthcare and banking.

Fully Homomorphic Encryption (FHE) enables computations to be performed on encrypted data without the need for decryption. The idea of homomorphic encryption was first introduced in 1978 by Rivest, Adleman, and Dertouzos [3], which they called "privacy homomorphism". However, it was not until thirty years later that the first construction of fully homomorphic encryption was proposed in [2]. Before the construction of [2], several encryption schemes with partial and somewhat homomorphic capabilities were proposed. The RSA encryption algorithm, proposed by Rivest, Shamir, and Adleman in [4], is an example of a homomorphic encryption scheme that is homomorphic with respect to multiplication. Researchers have attempted to create homomorphic encryption systems using various algebraic structures, in addition to conventional approaches. Several proposals for homomorphic encryption employing lattices and linear codes have been made, including works by Armknecht et al. in [5], Rivest and Ronald L in [6], Melchor et al. in [7], Melchor et al. in [8], and Peikert et al. in [9]. The Polly Cracker [10] family of techniques uses multivariate polynomial algebra and is naturally homomorphic with respect to addition and multiplication. However, these schemes suffer from the problem that the size of the ciphertext expands exponentially after multiplication [11]. Researchers continue to explore new algebraic structures and techniques in their quest for fully homomorphic encryption.

In 2009, a significant breakthrough occurred in cryptography with the introduction of the first fully homomorphic encryption (FHE) scheme by Gentry [2], which enabled the evaluation of arbitrary circuits while preserving the security of the encrypted data. Gentry's work not only presented the FHE scheme but also introduced a method to construct a general FHE scheme that had limited but sufficient homomorphic evaluation capabilities. After that, there has been a growing interest in homomorphic encryption, leading to the development of new schemes such as BGV [12], BFV [13], [14], CKKS [15], etc. each contributing to the advancement of homomorphic encryption.

FHE schemes can be simply explained by the example given in [2]. Alice owns a jewellery store where she employs workers to make jewellery out of priceless materials

like gold and diamonds. But, Alice is concerned about the possibility of stealing by the workers during the crafting process. The question is whether there is a way for workers to produce the jewellery without direct access to the raw materials. One solution is that Alice locks the raw materials in a transparent closed box with gloves, which the workers can wear to handle the raw materials without having direct access to them. The boxes are locked so that workers cannot touch any material during processing. Once the workers finish crafting the jewellery, Alice retrieves the box, unlocks it, and retrieves the finished jewellery. This analogy story demonstrates the fundamental principles of the FHE schemes, which encrypt confidential information before processing it without the need for decryption. FHE schemes enable computation on encrypted data without revealing its original contents like the raw materials kept in a closed box with gloves. In summary, the box represents the encryption algorithm, the lock of the box is the key, the raw materials within a locked box represent encrypted data, the making process represents computational operations on the encrypted data, and opening the box to obtain the jewellery represents the decryption to get the results.

Current research in FHE is focused on creating more efficient schemes for conducting computations on encrypted data since the present FHE schemes are computationally costly. It might involve new cryptographic techniques and optimization approaches. Overall, FHE has the potential to enhance data security and privacy in daily life, especially for cloud applications, utilized when a user lacks the processing power needed while handling massive amounts of data. To overcome this limitation, the user can use cloud computing services to process the data and obtain the results. However, data security and privacy may be risked when data is directly shared on the cloud. To solve this problem, FHE schemes offer the possibility to process data while keeping it encrypted, protecting user privacy. A user can use FHE to encrypt their data before sending it to the cloud to be processed. The user can decrypt the data after it has been processed by the cloud and returned to them. Apart from the cloud applications, FHE has several other applications in security and privacy since computations can be outsourced to a third (untrusted) party while the data is still in encrypted form. Numerous applications for homomorphic encryption include delegated computation, private data processing, searching on encrypted data, privacy preserving machine learning and many more [16].

In 2021, Dowerah et al. [11] proposed an FHE scheme based on the multivariate polynomial evaluation. The security of the scheme depends on the learning with errors (LWE) problem. Homomorphic multiplication is performed by evaluating a bilinear map on the ciphertexts. A three-way tensor, which acts as the public evaluation key for multiplication, is used to represent the map. Contrary to Brakerski-Gentry-Vaikuntanathan (BGV) [12] scheme and Brakerski-Fan-Vercauteren (BFV) [13], [14] scheme, the scheme does not enlarge the size of the ciphertexts after multiplication. Consequently, relinearization is not necessary for their FHE scheme.

For the usability of FHE in various applications, it is needed that these schemes are thoroughly studied and implemented to be able to use in practice. There are various open-source implementations of the standard FHE schemes [12]–[14]. For

instance, the open-source software library HElib [17] implements the BGV [12] scheme. Similarly, Microsoft SEAL [18] is the library implementation of BFV [13], [14] scheme. However, Dowerah et al.'s scheme [11] has not been implemented yet, which leads to our goal in this thesis.

An extension of homomorphic encryption is its multi-key variant called Multi-key homomorphic encryption (MK-HE). MK-HE enables separate parties to utilize distinct keys for encryption. L'opez-Alt et al. [19] were the first to introduce the idea of multi-key homomorphic encryption and its particular application based on the NTRU scheme. Multi-key variations of the fully homomorphic encryption schemes, BFV [13], [14] and Cheon-Kim-Kim-Song (CKKS) [15] scheme called MK-BFV and MK-CKKS were proposed by Chen et al [20]. However, multi-key variant of Dowerah and Krishnaswamy does not exist. Hence, our further aim is to extend their scheme into the multi-key setting. Multi-key homomorphic encryption (MK-HE) enables separate parties to utilize distinct keys for encryption, different from traditional homomorphic encryption scheme which assesses the arithmetic circuits of ciphertexts encrypted with the same key.

1.1 Aim

As mentioned in the previous sections, FHE is of tremendous interest in the current research scenario and has several applications in privacy and security solutions. Therefore, our first goal is to develop a deep understanding of the basic cryptographic knowledge, encryption principles and how the scheme proposed in [11] works and acquire the theoretical knowledge required to further the research in this area. We achieve this by providing an overview of state-of-the-art literature on fully homomorphic encryption and by implementing the FHE scheme proposed in [11].

Various open-source libraries exist in the literature for different FHE schemes, but there are no implementations of [11] so far. Therefore, our goal is to give an implementation of their scheme in this thesis. Studying and implementing the scheme into a code is challenging. Further challenges involved in implementing [11] into practice are selecting parameters, choosing the programming language, constructing algorithms, and conducting experiments on them. Specifically, parameter selection is challenging since the parameters will decide the scheme's efficiency and security. Increasing the values or complexity of the parameters can improve the security of the FHE scheme, but the cost is increased in time and power consumed. To balance this trade-off, the parameters setting needs careful consideration. Regarding the programming language selection, many implementations of FHE schemes exist and can be studied, and they are using different programming languages. Different languages have different libraries or tools that can be used to implement but not all of them have these tools, so selecting a suitable programming language can simplify the implementation process and decrease the risk of errors. In addition to appropriate tools and libraries, the language needs to have high-performance abilities and efficient memory management to implement FHE schemes. The implementation of the FHE scheme has a challenge in the algorithm due to the complexity of the mathematical

computations involved in key generation. The FHE scheme complexity is based on the complexity of mathematical computations so it can make testing time-consuming and resource-intensive. Additionally, small details in the computations can lead to incorrect decryption, so identifying the reason that caused the wrong decryption is challenging. During the implementation process, the only way to determine whether it is successfully implemented is by the correct decryption. Hence, it may require checking every step and verifying the reasons that caused any wrong decryption, which is challenging.

Further, our aim is to explore richer cryptographic properties in existing FHE schemes. We aim to extend the FHE scheme of [11] to a multi-key setting.

In summary, our main goals are as follows:

1. Literature review of state-of-the-art FHE schemes
2. Implementation of the scheme proposed in [11]
3. Extending [11] to the multi-key setting

We address the following research questions based on the above goals:

1. What is the current state-of-the-art of fully homomorphic encryption? How does the LWE-based FHE schemes work?
2. What are the challenges in implementing an FHE scheme and how to address them?
3. Is it possible to extend a single-key FHE scheme to a multi-key scheme? What are the challenges involved in doing so?

1.2 Our Contribution

Our contributions in this thesis can be divided into three parts:

- A state of the art literature survey of LWE based FHE schemes.
- Implementation of Dowerah and Krishnaswamy's FHE scheme [11].
- Extending the scheme proposed in [11] to the multi-key setting.

1.3 Outline

This thesis is organized as follows. In Chapter 2, we give the necessary background on fully homomorphic encryption and lattice based cryptography. In Chapter 3, we give some literature review on LWE based FHE schemes. Chapter 4 presents our methodology and implementation details, and in Chapter 5, we present the possibility of the extension of the single-key scheme to a multi-key scheme. Finally, Chapter 6 concludes the thesis.

2

Preliminaries

This chapter covers the necessary theoretical knowledge as well as several key concepts in mathematics and cryptography. We discuss some basic concepts from Abstract Algebra and some basic definitions and computational problems on lattices. Furthermore, we discuss the main ideas related to fully homomorphic encryption.

2.1 Notations

The notations listed in Table 2.1 are used in this thesis.

Table 2.1: Mathematical Notations

Notations	**Meaning**
$\mathbb{Z}$	Set of Integers
$\mathbb{N}$	Set of Natural numbers
$\mathbb{R}$	Set of Real numbers
$\mathbb{Q}$	Set of Rational numbers
λ	Security parameter
$x \in \mathbb{R}, \lfloor x \rfloor, \lceil x \rceil, \lfloor x \rceil$	Rounding of x down, up, or to the nearest integer
$\boldsymbol{v}$	A vector $\boldsymbol{v}$
$\|\boldsymbol{v}\|, \|\boldsymbol{v}\|_\infty$	The Euclidean norm of $\boldsymbol{v}$, the infinity norm of $\boldsymbol{v}$
χ	A probability distribution on $\mathbb{Z}$
$\boldsymbol{M}$	A matrix $\boldsymbol{M}$
q	A prime number
$\mathbb{Z}_q$	The set of integers modulo q
n	Dimension n of polynomial vector space
$\mathbb{Z}_q^n$	The n-dimensional vector space over the finite field $\mathbb{Z}_q$
$\mathcal{I}$	An Ideal
$\mathcal{R}$	A Ring
$i \in [1, n]$	an integer i range from 1 to n, which $n \in \mathbb{Z}$

2.2 Rings and Fields

Definition 1 (Ring [21]) . *A ring is a set equipped with two binary operations called addition and multiplication, which satisfy certain axioms. In detail, a ring is a set $\mathcal{R}$ with two binary operations denoted by $'+'$ and $'\cdot\ '$ such that:*

1. *$(\mathcal{R}, +)$ is an abelian group, which means that addition is commutative, associative, has an identity element denoted by 0 such that $a + 0 = a$ and $0 + a = a$, and every element has an inverse such that $a + (-a) = (-a) + a = 0$.*

2. *$(\mathcal{R}, \cdot)$ is a monoid, which means that multiplication is associative such that*

$$(a \cdot b) \cdot c = a \cdot (b \cdot c)\ \forall a, b, c \in \mathcal{R} \tag{2.1}$$

and has an identity element denoted by 1. Multiplication is distributive over addition, which means that for all $a, b, c \in \mathcal{R}$,

$$a \cdot (b + c) = a \cdot b + a \cdot c\ \ (left\ \ distributivity) \tag{2.2}$$

and

$$(a + b) \cdot c = a \cdot c + b \cdot c\ \ (right\ \ distributivity). \tag{2.3}$$

Definition 2 (Ring Homomorphism [22]) . *A ring homomorphism is a function $f : \mathcal{R} \to \mathcal{P}$ between two rings $\mathcal{R}, \mathcal{P}$ such that*

$$f(a + b) = f(a) + f(b) \tag{2.4}$$

and

$$f(a \cdot b) = f(a) \cdot f(b) \tag{2.5}$$

for all a and b in $\mathcal{R}$.

Definition 3 (Subring [21]) . *In a ring $\mathcal{R}$, a subring is a subset that is both closed under addition $'+'$ and multiplication $'\cdot'$, and also forms a ring under these operations.*

Definition 4 (Ideal [21]) . *An ideal of a ring $\mathcal{R}$ is a subset $\mathcal{I}$ of $\mathcal{R}$ that is a subring of $\mathcal{R}$ and satisfies the property that for any element $a \in \mathcal{I}$ and $r \in \mathcal{R}$, the products ar and ra are both contained in $\mathcal{I}$.*

Definition 5 (Field [21]) . *A field is a set $\mathbb{F}$ that is equipped with two binary operations, which are addition and multiplication. The set $\mathbb{F}$ contains two special elements, 0 and e, where 0 and e are distinct from each other.*

In addition to these basic properties, $\mathbb{F}$ needs to satisfy the following conditions:

1. *First, the set $\mathbb{F}$ needs to be an abelian group with respect to addition, where 0 serves as the identity element.*

2. *Second, the elements of* $\mathbb{F}$ *that are not equal to* 0 *must form an abelian group with respect to multiplication, where* e *is the identity element.*

3. *Third, the operations of addition and multiplication satisfied the distributive law, which states that* $a \cdot (b + c) = a \cdot b + a \cdot c$ *for all* $a, b, c \in \mathbb{F}$.

4. *Finally, the second distributive law,* $(b+c) \cdot a = b \cdot a + c \cdot a$*, follows automatically from the commutativity of multiplication.*

2.3 Polynomials

To define polynomials, we first define monomials.

Definition 6 (Monomial [23]) . *A monomial in variables* $x_1, ..., x_n$ *is the product of one or more terms of the form*

$$x_i^{\alpha_i}, i \in [1, n],$$

where each exponent α_i *is a non-negative integer. The total degree of the monomial is the sum of all the exponents, that is* $\alpha_1 + ... + \alpha_n$*, where* α_i *is the exponent of variable* x_i.

Definition 7 (Polynomial [23]) . *A polynomial* f *in variables* $x_1, ..., x_n$ *with coefficients from a field* $\mathbb{F}$ *is a linear combination, with coefficients in* $\mathbb{F}$*, of monomials. It can be represented as*

$$f = \sum_{\alpha} a_\alpha x^\alpha, a_\alpha \in \mathbb{F},$$

where the sum is over a finite number of n*-tuples* $\alpha = (\alpha_1, ..., \alpha_n)$*. The set of all such polynomials is denoted by* $\mathbb{F}[x_1, ..., x_n]$.

The coefficient of the monomial x^α *is* a_α*. If the coefficient* a_α *is non-zero, then* $a_\alpha x^\alpha$ *is a term of* f*. The total degree of* f*, denoted* $deg(f)$*, is the maximum value of* $|\alpha|$ *such that the coefficient* a_α *is non-zero.*

2.4 Lattice Theory

2.4.1 Lattices

Definition 8 (Lattices [24]) . *In* n*-dimensional Euclidean space, a lattice* $L \in \mathbb{R}^n$ *is a discrete additive subgroup that consists of a set of points arranged in a regular, periodic pattern. This set of points is generated by a set of* k *linearly independent vectors* $\boldsymbol{b}_1, \boldsymbol{b}_2, ..., \boldsymbol{b}_k$ *with each vector being* n*-dimensional in* $\mathbb{R}^n$*. By taking integer linear combinations of the basis vectors, any point in the lattice can be generated. Thus, the lattice spanned by these vectors is defined as:*

$$L(\boldsymbol{b}_1, \boldsymbol{b}_2, ..., \boldsymbol{b}_k) = \sum_{i=1}^{k} c_i \boldsymbol{b}_i \ : c_i \in \mathbb{Z} \tag{2.6}$$

Let $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ be a set of n linearly independent vectors arranged as the columns of a matrix $\boldsymbol{B} \in \mathbb{R}^{n \times n}$, then the n-dimensional lattice generated by $\boldsymbol{B}$ is given by

$$L(B) = \{\boldsymbol{x} \cdot \boldsymbol{B} : \boldsymbol{x} \in \mathbb{Z}^n\}. \tag{2.7}$$

2.4.2 Lattice Problems

Lattice problems are a type of mathematical problem that involve finding short vectors and the closest vectors in a lattice. These problems form the foundation for many lattice-based cryptographic schemes and are considered to be computationally hard.

2.4.2.1 CVP and SVP

The Closest Vector Problem asks for a vector that is relatively close to a given target point. The Closest Vector Problem problem is NP-hard since the Closest Vector Problem is proven at least as hard as the Shortest Vector Problem [25].

Definition 9 (Closest Vector Problem (CVP) [25]) . *Given a basis $\boldsymbol{B} = (\boldsymbol{b}_1, ..., \boldsymbol{b}_n)$ of a lattice $\mathcal{L}$ and a target vector $\boldsymbol{t}$, find a vector $\boldsymbol{v}$ in $\mathcal{L}$ closest to $\boldsymbol{t}$ such that the distance $\|\boldsymbol{t} - \boldsymbol{v}\|$ is minimized.*

The Shortest Vector Problem asks for a nonzero vector whose Euclidean norm is minimum among all other nonzero lattice vectors and in 1996, Ajtai proved that the Shortest Vector Problem problem is NP-hard [26].

Definition 10 (Shortest Vector Problem (SVP) [25]) . *Given a basis $\boldsymbol{B} = (\boldsymbol{b}_1, ..., \boldsymbol{b}_n)$ of a lattice $\mathcal{L}$, find the shortest non-zero vector $\boldsymbol{s} \in \mathcal{L}$ such that $\|\boldsymbol{s}\| = \lambda_1(\mathcal{L})$, where $\lambda_1(\mathcal{L})$ is the length of the shortest lattice vector in $\mathcal{L}$.*

The SVP can be relaxed to an approximation algorithm to provide solutions that are only guaranteed to be within an approximation factor γ from the optimum. In the case of the γ-approximation SVP, the goal is to find a non-zero lattice vector at most $\gamma\lambda_1(\mathcal{L})$, where γ is a function of n and is greater than or equal to 1. When $\gamma = 1$, γ-approximation SVP is SVP [25].

Definition 11 (Approximate SVP (SVP_γ) [25]) . *Given a basis $\boldsymbol{B} = (\boldsymbol{b}_1, ..., \boldsymbol{b}_n)$ of a lattice $\mathcal{L}$ and an approximation factor $\gamma \geq 1$, find a non-zero vector $\boldsymbol{v}$ in $\mathcal{L}$ such that the distance $\|\boldsymbol{v}\| \leq \gamma\lambda_1(\mathcal{L})$ is minimized.*

The security of lattice-based cryptographic schemes depends on the hardness of particular lattice problems. However, proving the security of these schemes based on the search version of the SVP_γ is currently an open problem [27]. Instead, the security of these schemes is typically proven based on the decisional variant of the problem, known as the Decisional Approximate Shortest Vector Problem (GapSVP_γ).

Definition 12 (Decisional Approximate SVP (GapSVP_γ) [27]) . *Given a basis $\boldsymbol{B} = (\boldsymbol{b}_1, ..., \boldsymbol{b}_n)$ of an n-dimensional lattice $\mathcal{L} = \mathcal{L}(\boldsymbol{B})$, the GapSVP_γ problem is to decide whether $\lambda_1(\mathcal{L}) \leq 1$ or $\lambda_1(\mathcal{L}) > \gamma(n)$.*

2.4.2.2 Learning with Errors

Learning with Errors (LWE) is a computational problem that involves solving a system of linear equations with errors. It was introduced as an extension of the Learning Parity with Noise (LPN) problem to a higher modulus. The goal of LWE is to extract a secret vector from a set of equations that include both a secret vector and a noise term generated from a specific probability distribution. This noise term is added to make the problem computationally challenging. LWE has been widely studied in cryptography and has numerous applications [11]. The definition of the LWE problem is given as follows.

Definition 13 (Learning With Error (LWE) [11]) . *The Learning with Errors (LWE) problem, denoted by $LWE_{n,q,\mathcal{X}}$, involves a probability distribution $\mathcal{X}$ on $\mathbb{Z}$ and a secret vector $\boldsymbol{s}$ uniformly chosen from $\mathbb{Z}_q^n$ for some $n, q \in \mathbb{N}$. The distribution $\mathcal{A}_{\boldsymbol{s},\mathcal{X}}$ generates a pair $(\boldsymbol{a}, b = \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e \in \mathbb{Z}_q^n \times \mathbb{Z}_q)$ by randomly selecting a vector $\boldsymbol{a}$ from $\mathbb{Z}_q^n$ and an error e from $\mathcal{X}$. The LWE problem is to output the vector $\boldsymbol{s} \in \mathbb{Z}_q^n$ with overwhelming probability, given polynomially many samples from $\mathcal{A}_{\boldsymbol{s},\mathcal{X}}$.*

The decisional Learning With Errors (DLWE) problem, denoted as $DLWE_{n,q,\mathcal{X}}$, is to distinguish the distribution $A_{s,\mathcal{X}}$ from the uniform distribution over $Z_q^n \times Z_q$.

Overwhelming probability refers to the likelihood that an event will occur with a probability approach of 1. An event is said to occur with overwhelming probability if the probability of it not occurring is negligible [28].

2.4.2.2.1 Hardness of LWE The hardness of LWE is related to the difficulty of solving lattice problems. Specifically, if the probability distribution used in LWE is a discretized Gaussian distribution with a standard deviation of at least $2\sqrt{n}$, then it is statistically indistinguishable from a distribution that is bounded by some value B. This allows for a quantum reduction of LWE to approximate the decisional Shortest Vector Problem ($GapSVP_\gamma$), where γ is the approximation factor that depends on the ratio q/B. Classical reductions of LWE from worst-case lattice problems have also been shown to exist, with different moduli giving different levels of hardness. The best-known algorithms for these problems require superpolynomial time [11].

2.4.2.3 Ring Learning with Errors

The Ring Learning With Errors (RLWE) problem is a variant of the Learning With Errors (LWE) problem, which is a difficult computational problem. Here is the definition of the Decision Ring-LWE problem.

Definition 14 (Ring-Learning With Errors (RLWE) [29]) . *The ring $\mathcal{R} = \mathbb{Z}_q[x]/\langle x^n + 1 \rangle$ with n as a power of 2 and an error distribution $\mathcal{X}$ over $\mathcal{R}$ are given, a random element $\boldsymbol{w}$ is sampled uniformly from $\mathbb{R}_q$. The decision RLWE problem involves distinguishing between two probability distributions based on m independent samples $(\boldsymbol{a}_i, \boldsymbol{a}_i\boldsymbol{w} + \boldsymbol{e}_i) \in \mathbb{R}_q \times \mathbb{R}_q$, where $\boldsymbol{e}_i$ are polynomials in $\mathcal{R}$. Each sample is distributed according to either the distribution $A_{\boldsymbol{s},\boldsymbol{\mathcal{X}}}$ or the uniform distribution over*

$\mathcal{R}$.

The RLWE problem is the LWE problem over a ring $\mathcal{R}$, which is to distinguish the output $(\boldsymbol{a}_i, \boldsymbol{a}_i\boldsymbol{w} + \boldsymbol{e}_i)$ and the randomly generated secret polynomial from the samples, where the samples are corrupted by the noise $\boldsymbol{e}_i$. Compared with LWE, all elements are intuitively smaller than LWE since each part is a polynomial in RLWE instead of a matrix in the LWE problem which improves the efficiency of a scheme.

2.5 Homomorphic Encryption

Homomorphic encryption (HE) allows for performing computations on encrypted data without decrypting it. This enables computations to be performed on ciphertexts directly without revealing any information about the plaintexts.

An encryption scheme is said to be homomorphic if it satisfies the following condition: $\forall m_1, m_2 \in \mathcal{M}$,

$$Enc(m_1 \square m_2) = Enc(m_1) \triangle Enc(m_2),$$

where Enc is the encryption function, $\mathcal{M}$ is the plaintext space, and $\square$ and $\triangle$ are the operators in the plaintext and ciphertext spaces, respectively. The $=$ symbol means computations can be directly computed without the need for decryption [22].

Definition 15 (Homomorphic Encryption (HE) Scheme [11]) *. A homomorphic encryption scheme HE = (KeyGen, Enc, Eval, Dec) with message space $\mathcal{M}$ consists of the following four algorithms:*

1. *HE.KeyGen$(1^\lambda) \to (pk, sk, evk)$: This algorithm generates a public encryption key pk, a secret decryption key sk, and a public evaluation key evk using the security parameter λ.*

2. *HE.Enc$(pk, m) \to \boldsymbol{c}$: This algorithm encrypts a message $m \in \{0,1\}$ into the corresponding ciphertext $\boldsymbol{c}$ using the public key pk.*

3. *HE.Eval$(evk, \phi, \boldsymbol{c}_1, ..., \boldsymbol{c}_t) \to \boldsymbol{c}_{eval}$: Takes the evaluation key evk, a function $\phi = \{0,1\}^t \to \{0,1\}$, and ciphertexts $\boldsymbol{c}_1, ..., \boldsymbol{c}_t$ as input and outputs the new ciphertext $\boldsymbol{c}_{eval}$. The function ϕ is an arithmetic circuit over the binary field $GF(2)$ that uses only addition and multiplication operations.*

4. *HE.Dec$(sk, \boldsymbol{c}) \to m$: This algorithm decrypts a ciphertext $\boldsymbol{c}$ using the secret key sk and recovers the corresponding original message m.*

Indistinguishability under chosen plaintext attack (IND-CPA) security is a concept in cryptography that describes a property of encryption schemes, which means the attacker cannot learn anything about the plaintext from the ciphertext. This ensures that the encrypted messages are secure even if the attacker has access to the ciphertexts. A homomorphic encryption scheme is said to be secure if it is IND-CPA secure [30].

Definition 16 (IND-CPA Security [11]) . *A homomorphic encryption scheme is considered indistinguishable under chosen-plaintext attack (IND-CPA) secure if an adversary A, who has access to the public encryption key pk, public evaluation key evk, and encryption of a message m_β (given by the encryption algorithm of a HE scheme such as HE.Enc(m_β, pk)) for a random bit $\beta \in \{0,1\}$, cannot guess the value of bit β with probability greater than $\frac{1}{2}$ plus a negligible function, which is $\frac{1}{2} + negl(\lambda)$.*

The correctness property for a homomorphic encryption scheme HE with message space $\mathcal{M}$ states that when a homomorphic operation is performed on a ciphertext and then decrypted, the decryption result needs to be equal to the same operation performed on the plaintext message.

Definition 17 (Correctness [11]) . *Given a set of functions Φ, generate (pk, sk, evk) by HE.KeyGen(1^λ), generate $\boldsymbol{c}_i$ by HE.Enc(pk, m_i) for all i, and generate $\boldsymbol{c}_{eval}$ by HE.Eval$(evk, \phi, \boldsymbol{c}_1, ..., \boldsymbol{c}_t)$, where $\phi \in \Phi$. A homomorphic encryption scheme correctly evaluates the set of functions Φ if for any $\phi \in \Phi$ and any messages $m_1, ..., m_t \in \{0,1\}$, we can decrypt $\boldsymbol{c}_{eval}$ using the secret key sk and obtain $\phi(m_1, ..., m_t)$ such that HE.Dec$(\boldsymbol{c}_{eval}, sk) = \phi(m_1, ..., m_t)$ holds.*

If a homomorphic encryption scheme satisfies the above property for any depth L arithmetic circuit over $GF(2)$ and for all $m_1, ..., m_t \in \{0,1\}$, then it is a L-homomorphic encryption scheme[11].

The compactness property of a homomorphic encryption scheme HE states that for any given security parameter λ, the size of the output of an evaluation on a ciphertext does not depend on the evaluation circuit.

Definition 18 (Compactness [11]) . *For a fixed polynomial bound $b = b(\lambda)$, a homomorphic encryption scheme is said to be compact if the size of the output ciphertext from HE.Eval is at most b bits, regardless of the function ϕ being evaluated.*

There are three main types of homomorphic encryption based on the complexity of the computations it can perform: Partially Homomorphic Encryption (PHE), Somewhat Homomorphic Encryption (SWHE) and Fully Homomorphic Encryption (FHE)[31].

2.5.1 Partially Homomorphic Encryption

Partially homomorphic encryption refers to a type of homomorphic encryption that allows only one type of computation to be performed on encrypted data. The two most common types of partial homomorphic encryption are additive homomorphism and multiplicative homomorphism. An example of additive homomorphic encryption is the ElGamal encryption algorithm [22]. Rivest-Shamir-Adleman (RSA) is an example of PHE as well, which was proposed by Rivest et al. [32] in 1978 and the multiplicative homomorphic property was shown in [3]. RSA is based on the difficulty of factoring large composite numbers into their prime factors, which are defined as follows.

1. $RSA.KeyGen(\lambda) \rightarrow (pk, sk)$: first, choose two large primes p and q, let $n = pq$ and $\phi = (p-1)(q-1)$. Then, choose an integer e such that $gcd(e, \phi) = 1$, and calculate $d = e - 1 \mod \phi$ since $ed = 1 \mod \phi$. Finally, set the public key pk as (e, n) and the secret key sk as (d, n). The $gcd(p, q)$ function is used to verify that p and q are relatively prime, which means they have no common factors other than 1.

2. $RSA.Enc(pk, m) \rightarrow c$: a message m such that $0 \leq m < n$, then compute ciphertext $c = Enc(m) = m^e(\mod n)$.

3. $RSA.Dec(sk, m) \rightarrow m$: recover the message from the ciphertext by computing $m = Dec(c) = c^d(\mod n)$.

4. $RSA.Mult(m_1, m_2)$: $\forall m_1, m_2 \in \mathcal{M}$, then $Enc(m_1) \cdot Enc(m_2) = (m_1^e(\mod n)) \cdot (m_2^e(\mod n)) = (m_1 \cdot m_2)^e(\mod n) = Enc(m_1 \cdot m_2)$.

RSA has the multiplicative homomorphic property that allows for the computation of $RSA.Enc(m_1 \cdot m_2)$ directly using $RSA.Enc(m_1)$ and $RSA.Enc(m_2)$ without decryption. However, RSA does not support the homomorphic addition of ciphertexts.

2.5.2 Somewhat Homomorphic Encryption

Partially homomorphic encryption schemes, such as RSA and ElGamal, can only support one operation, either addition or multiplication, on the encrypted data. This limitation makes it difficult to perform complex computations on encrypted data. Somewhat homomorphic encryption (SWHE) allows for both addition and multiplication to be computed simultaneously on the ciphertext. However, the number of these operations is limited, and the range of functions that can be computed is also restricted [14].

2.5.3 Fully Homomorphic Encryption

Fully Homomorphic Encryption (FHE) allows arbitrary computations to be performed on encrypted data, with no limitations on the types of operations that can be performed.

Definition 19 (Fully Homomorphic Encryption Scheme [11]) . *If the set of all efficiently computable functions is denoted as Φ, an encryption scheme is called fully homomorphic if it is compact and homomorphic with respect to the set of functions Φ.*

2.5.3.1 Leveled Fully Homomorphic Eencryption

Leveled Homomorphic Encryption (LHE) is a type of fully homomorphic encryption that allows computations on encrypted data of a limited complexity with an upper limit, denoted as L. If the function f to be computed is represented by a binary circuit C, then the depth and size of C must be within the range of L, that is $|C| \leq L$. This

means that LHE is capable of performing low-complexity homomorphic operations when L is relatively large. Leveled Fully Homomorphic Encryption (Leveled FHE) can perform computations at most L depths, in which the number of levels is fixed in advance in the parameter setting. In conclusion, the leveled FHE scheme may evaluate circuits with various gate types using a limited number of levels. The FHE scheme is capable of evaluating circuits with various gate types and an unlimited depth. The leveled FHE scheme is capable of evaluating circuits with a variety of gates and a fixed number of levels.

Gentry's [2] work showed that if the decryption circuit of a homomorphic encryption scheme has a lower depth, then a scheme designed for low depth circuits can be transformed into a scheme for polynomial depth circuits where the polynomial is predefined. This property can be used to construct an unbounded circular security counterexample, which means that it demonstrates a situation where the security of a homomorphic encryption scheme breaks down due to a circular dependency between encryption and decryption operations. However, the notion of bootstrappability does not directly imply fully homomorphic encryption since an FHE scheme needs to be able to evaluate a ciphertext on all polynomial depth circuits, not just on predefined polynomials.

Definition 20 (Bootstrappable Scheme [30]) . *An L-homomorphic encryption scheme that can correctly evaluate any depth L circuit, it says to be bootstrappable if the depth of its decryption circuit is less than L.*

If an encryption scheme is secure even if the adversary has access to the secret key's encrypted bits, it is considered to be circularly secure [11].

Definition 21 (Boostrapping [2]) . *If there exists an L-homomorphic encryption scheme and it is Boostrappable, then there exists a leveled fully homomorphic encryption scheme.*

If this scheme is circular secure, then there exists a fully homomorphic encryption scheme.

Circular security is a concept in the public key cryptosystem that focuses on the security of a scheme when an adversary has access to an encryption of the secret key. It ensures that the adversary cannot gain any additional information about the secret key. A scheme is considered circularly secure if it can resist such attacks[30].

2.5.3.2 Multi-key FHE

Traditional FHE schemes are single-key which means that the computations can be performed on ciphertexts encrypted under the same key. When all parties involved in the computation are trustworthy and all of them are willing to share the secret key, a single-key FHE scheme can be used. When there are multiple parties involved and in a scenario when they do not trust each other, we need a multi-key FHE scheme. In a multi-key FHE scheme, computations can be performed on ciphertexts encrypted under different keys. To decrypt the resulting ciphertext, one needs to use all the

involved secret keys.

The notion of multi-key homomorphic encryption scheme was first proposed by L'opez-Alt et al. [19]. They applied the key-switching (also known as relinearization) technology and modulus-switching technology to design an NTRU based multi-key fully homomorphic scheme. They introduced a new parameter N which is the number of keys. Compared with the single-key FHE scheme, they mentioned in the article [19] that the multi-key FHE scheme has two differences, which are the homomorphic evaluation algorithm takes many ciphertexts encrypted up to N keys in a polynomial manner and all involved keys need to be used to decrypt the resulting ciphertext.

Multi-key leveled fully homomorphic encryption (MKFHE) is a cryptographic system that allows for the evaluation of an arithmetic circuit on ciphertexts, even when those ciphertexts are encrypted under different keys.

Definition 22 (Multi-Key Fully Homomorphic Encryption (MKFHE) [33]) *. For a set of circuits $\mathcal{C}$, a leveled Multi-key fully homomorphic encryption (MKFHE) scheme ε = (Setup, KeyGen, Enc, Dec, Eval) is defined as follows:*

1. *ε.Setup$(1^\lambda, 1^K, 1^L) \rightarrow pp$: Taking the security parameter λ, the number of distinct users K, and the circuit depth L as input, output the public parameter pp.*
2. *ε.KeyGen$(pp) \rightarrow (sk_i, pk_i, evk_i)$: Taking the pp as input, output a public key pk_i, a secret key sk_i, and a evaluation key evk_i of party i, where $i \in \{1, 2, ..., K\}$.*
3. *ε.Enc$(m, pk_i) \rightarrow ct_i$: Taking pk_i as input and encrypting a plaintext m as input, outputs a ciphertext ct_i.*
4. *ε.Dec$((sk_{i_1}, ..., sk_{i_k}), ct_S) \rightarrow m$: Taking a ciphertext ct_S along with the corresponding set of users $S = \{i_1, i_2, ..., i_k\} \subseteq [K]$ and their secret keys $sk_S = \{sk_{i_1}, sk_{i_2}, ..., sk_{i_k}\}$ as input, output the plaintext m.*
5. *ε.Eval$(\mathcal{C}, (ct_{S_1}, pk_{S_1}, evk_{S_1}), ..., (ct_{S_t}, pk_{S_t}, evk_{S_t})) \rightarrow ct_S$: Taking a boolean circuit $\mathcal{C}$ with t tuples $(ct_{S_i}, pk_{S_i}, evk_{S_i})_{i=1,...,t}$, where each tuple comprises of a ciphertext ct_{S_i} corresponding to a user set S_i, a set of public keys $pk_{S_i} = \{pk_j, \forall j \in S_i\}$, and the evaluation keys evk_{S_i} as inputs, outputs a ciphertext ct_S corresponding to a set of secret keys indexed by $S = \cup_{i=1}^{t} S_i \subseteq [K]$.*

MKFHE schemes need to have two properties, which are correctness and compactness.

Definition 23 (Correctness of MKFHE scheme [33]) *. Given a circuit $\mathcal{C}$ of depth at most L and a set of tuples $\{(ct_{S_i}, pk_{S_i})\}_{i\in\{1,...,t\}}$, assume $m_i = Dec(sk_{S_i}, ct_{S_i})$ in which $sk_{S_i} = \{sk_j, \forall j \in S_i\}$, an MKFHE scheme ε is considered correct if it satisfies the property that $\varepsilon.Dec(sk_S, Eval(\mathcal{C}, (ct_{S_i}, pk_{S_i}, evk_{S_i})_{i\in[t]}))= \mathcal{C}(m_1, ..., m_t)]$.*

Definition 24 (Compactness of MKFHE [33]) *. If there exists a polynomial $poly(\cdot, \cdot, \cdot)$ such that the size of the ciphertext ct, denoted as $|ct|$, is bounded by*

poly(λ, K, L). *That is, the size of ct is independent of the circuit* $\mathcal{C}$*, but it depends on the security parameter* λ*, the number of distinct users* K*, and the circuit depth* L*.*

3

Literature Review

In this chapter, we give a general overview of a few state-of-the-art FHE schemes in the first part and offer a brief description of LWE based FHE schemes in the second part to answer the RQ1.

3.1 A Brief Overview of Fully Homomorphic Encryption

Figure 3.1 illustrates the timeline depicting the development and progression of major homomorphic encryption (HE) schemes over time.

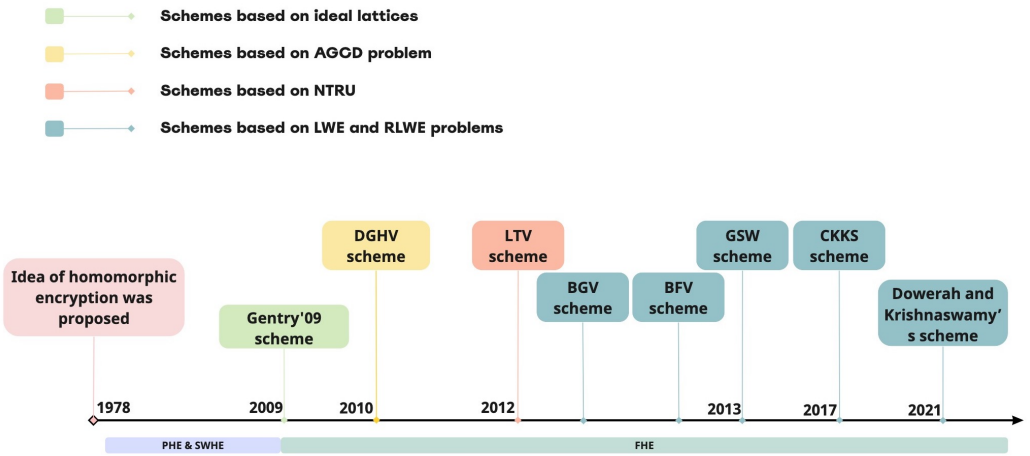

Figure 3.1: Timeline of Main HE Schemes

Fully Homomorphic Encryption (FHE) is considered the "holy grail" of cryptography. In 1978, Rivest, Adleman, and Dertouzos [3] first introduced the idea of homomorphic encryption which they called *privacy homomorphism* back then. Three decades later, Gentry [2] gave the first candidate construction of FHE based on ideal lattices in 2009, which designed a blueprint for creating the first FHE scheme. It involves three main steps: first creating a SWHE scheme that can perform computations on encrypted data using low-degree polynomials, then compressing the decryption circuit that can be expressed as a low-degree polynomial, and finally performing *bootstrapping* operations to create an FHE scheme. The first candidate construction of FHE was

proposed in [2], and this framework for building FHE schemes became a basis for consequent works for developing secure and efficient FHE schemes. The ciphertexts in Gentry's scheme contain noise and this noise grows as homomorphic computations are performed on the ciphertexts. In order to overcome this problem, Gentry introduced a technique called *bootstrapping*. Whenever the noise in a ciphertext becomes too large to be able to correctly decrypt, bootstrapping can be used to reduce the noise to a level when decryption is possible. However, bootstrapping is a very expensive step since the running time of one bootstrapping operation range from 30 seconds to 30 minutes [34], which makes the scheme inefficient. Even though Gentry's original FHE scheme [2] was impractical due to its high computing complexity, it is still a base stone for subsequent research and the creation of more efficient FHE schemes.

One year later, an FHE scheme called DGHV [35] based on Approximate-Greatest Common Divisor (AGCD) problem over integers was proposed by Dijk, Gentry, Halevi and Vaikuntanathan, in 2010. The AGCD problem is to recover a prime number p from a set of values $x_i = pq_i + r_i$, where q_i, r_i are small integers. It is a difficult computational problem when there are a large number of equations involved [36]. It simplifies the concept of Gentry's scheme and realizes FHE through simple integer addition and multiplication, which demonstrates the possibility of the FHE scheme being applied in practical scenarios. However, this simple way is achieved by taking a huge number of computations to ensure the security of the scheme, which leads to its inefficiency.

Later, the NTRU-based FHE scheme LTV [19] was proposed by Adriana Lopez-Alt, Eran Tromer, and Vinod Vaikuntanathan in 2012, based on the NTRU scheme. The N-th degree Truncated Polynomial Ring Unit (NTRU) scheme [37] is a type of public-key cryptosystem that uses polynomial algebra to provide encryption and decryption, which was proposed by Hoffstein, Pipher, and Silverman in 1996. It uses key switching to re-linearize the nonlinear part of the ciphertext product and modulus switching to perform reductions to achieve fully homomorphic operations. However, [38] shows that the NTRU FHE scheme is vulnerable to subdomain attacks, which can compromise the security of the encryption.

Based on a different lattice assumption called the Learning with Errors (LWE) problem, Brakerski, Gentry, and Vaikuntanathan presented a new FHE scheme called BGV in 2014 [12] where a (leveled) FHE scheme can be obtained without bootstrapping. Another scheme called the BFV scheme [13], [14] was subsequently proposed by Brakerski, Fan, and Vercauteren based on the (ring) LWE problem. In these schemes, the secret key and the ciphertext are both vectors of size $\mathcal{O}(n)$. Homomorphic multiplication is performed by taking the tensor product of two ciphertexts. As a result, the ciphertext's size grows to be $\mathcal{O}(n^2)$ after homomorphic multiplication. A technique called *relinearization* is used to reduce this explosion in ciphertext size in [12]–[14]. Ever since FHE has become a topic of great interest and substantial work has been done to improve its efficiency and practicality. Then, in 2013, Gentry, Sahai, and Waters proposed a new fully homomorphic encryption (FHE) scheme called GSW [39]. Similar to the BGV scheme, the GSW scheme has a finite series of fully homomorphic properties based on the simpler LWE problem and

can be fully homomorphic through bootstrapping.

After that, the Cheon-Kim-Kim-Song (CKKS) [15] scheme proposed by Jung Hee Cheon, Kyungmin Kim, Minkyeong Kim, and Dowon Song in 2017, uses a rescaling technique to manipulate ciphertexts during computation. In contrast to earlier encryption schemes, the decryption result is exactly the same as the plaintext, CKKS is intended for approximation computation and permits a little amount of error in the decryption results. Nevertheless, it is sufficient for the majority of operations since most computations take real numbers and only needs to retain a part of the efficient digits. Compared to existing approaches based on LWE/RLWE problems, this considerably improves the computational efficiency since the error is allowed and the accuracy limit is relaxed. There are many other FHE schemes that have been proposed as well, each with its own strengths and weaknesses.

Apart from new constructions, there has been a rapid increase in the development of various FHE schemes that are based on the BGV and GSW schemes, with a focus on optimizing and accelerating their operating efficiency. For instance, HElib [40] is an open-source fully homomorphic computing library based on the BGV scheme that includes several optimizations to improve its efficiency, the TFHE [41] library is a fast and efficient FHE scheme that uses the GSW scheme with optimizations to improve its performance.

3.2 FHE Schemes Based on the LWE Problem

In this part, we give a brief description of some FHE schemes based on the LWE problem.

3.2.1 BGV Scheme

The BGV scheme [12] is an FHE scheme based on the learning with errors (LWE) problem. It supports homomorphic operations over the integers. Specifically, the BGV scheme [12] uses a polynomial ring over the integers modulo q, where q is a prime integer, to represent messages and the secret key. The messages are encrypted as polynomials and then the noise is added to them. Decryption is accomplished by calculating the inner product of the secret key polynomial and the ciphertext polynomial. The inner product will also have some noise since it is introduced to the ciphertext polynomial during the encryption process. Then, a rounding operation is carried out on the inner product to get the nearest integer polynomial and get rid of the noise. As the security parameter λ limits the size of the noise, decryption correctly recovers the message.

3.2.1.1 Basic Scheme Description

The BGV scheme [12] consists of the following algorithms.

1. Setup($1^\lambda, 1^\mu, b$): The scheme takes the security parameter λ, a $\mu - bit$ modulus q such that $\mu = \theta(\log\lambda + \log L)$ where L is the depth of the circuit and the function θ is based on the level of security, the bit $b \in \{0, 1\}$ to decide whether setting parameters either for an LWE- based scheme (where $d = 1, \mathcal{R} = \mathbb{Z}$) or an RLWE-based scheme (where $n = 1, \mathcal{R} = \mathbb{Z}[x]/f(x), f(x) = x^d + 1, d = d(\lambda)$) as inputs, output other required parameters $n = n(\lambda, \mu, b), N = \lceil(2n + 1)\log q\rceil, \mathcal{X} = \mathcal{X}(\lambda, \mu, b)$. These parameters $params = (q, d, n, N, \mathcal{X})$ are selected to ensure the security of the scheme and the efficiency of the computations.

2. KeyGen($params$): The scheme generates two keys, which are the public key pk and the secret key sk. The secret key is a short secret vector that is used to generate the ciphertexts and decrypt the messages, which is $sk = \boldsymbol{s} = (1, \boldsymbol{s}')$, where $\boldsymbol{s}'$ is a randomly selected vector from the distribution $\mathcal{X}$. Then, randomly generated a matrix $\boldsymbol{B}$ from $\mathcal{R}_q^{N\times n}$ and $\boldsymbol{e}$ is an error vector sample from $\mathcal{X}^N$, let $\boldsymbol{b} = \boldsymbol{B}\boldsymbol{s}' + 2\boldsymbol{e}$, and set $\boldsymbol{A} = (\boldsymbol{b} \quad -\boldsymbol{B}) \in \mathcal{R}_q^{N\times(n+1)}$, then output $\boldsymbol{A}$ as the public key pk. (Observe, $\boldsymbol{A}\cdot\boldsymbol{s} = \boldsymbol{b}\cdot 1 - \boldsymbol{B}\cdot\boldsymbol{s'} = 2\boldsymbol{e}$).

3. Enc($params, pk, m$): Using public key pk to encrypt a message $m \in \mathcal{R}_2$. The ciphertext is $\boldsymbol{c} = \boldsymbol{m} + \boldsymbol{A}^\top\cdot\boldsymbol{r} = (c_0, c_1) = \boldsymbol{m} + 2(\boldsymbol{e_0}, \boldsymbol{e_1}) + \boldsymbol{r}\boldsymbol{A} \in \mathcal{R}_q^2$ where $\boldsymbol{m} = (m, 0, ..., 0) \in \mathcal{R}_q^2$, $\boldsymbol{r}, \boldsymbol{e_0}, \boldsymbol{e_1}$ are sampled from $\mathcal{X}$. $\mathcal{R}_2 = \mathcal{R}/2\mathcal{R}$ is the plaintext space. Then, $\boldsymbol{c} = (c_0, c_1) = (m + 2\boldsymbol{e}_0 + br, 2\boldsymbol{e}_1 - \boldsymbol{B}\boldsymbol{r})$.

4. Dec($params, sk, \boldsymbol{c}$): Taking the ciphertext $\boldsymbol{c}$, the secret key sk as input, output the decrypted messages
$$m = [\,[\,\langle\boldsymbol{c}, \boldsymbol{s}\rangle\,]_{\,q}\,]_{\,2}, \tag{3.1}$$
where $\langle\boldsymbol{c}, \boldsymbol{s}\rangle = c_0 + c_1\boldsymbol{s}$.

5. Add: Add two ciphertexts $\langle\boldsymbol{c}, \boldsymbol{s}\rangle = c_0 + c_1\boldsymbol{s}, \langle\boldsymbol{c'}, \boldsymbol{s}\rangle = c_0' + c_1'\boldsymbol{s}$, which are corresponding to the messages m, m', that is $\boldsymbol{c}_{add} = \langle\boldsymbol{c}, \boldsymbol{s}\rangle + \langle\boldsymbol{c'}, \boldsymbol{s}\rangle = (c_0 + c_0') + (c_1\boldsymbol{s} + c_1'\boldsymbol{s})$.

6. Mult: Multiply $\boldsymbol{c}$ and $\boldsymbol{c'}$, that is $\langle\boldsymbol{c}, \boldsymbol{s}\rangle\cdot\langle\boldsymbol{c'}, \boldsymbol{s}\rangle = (c_0 + c_1\boldsymbol{s})(c_0' + c_1'\boldsymbol{s}) = c_0c_0' + (c_0c_1' + c_1c_0') + c_1c_1'\boldsymbol{s}^2 = d_0 + d_1 + d_2\boldsymbol{s}^2$ is a ciphertext. Then, the extended ciphertext (d_0, d_1, d_2) can be decrypted using an extended secret key $(1, \boldsymbol{s}, \boldsymbol{s}^2)$ [42].

The BGV scheme uses the key switching technique to decrease the dimension expansion of the ciphertext vector when manipulating multiplication operations to the original ciphertext dimension. Besides, the BGV scheme replaces the bootstrapping process in Gentry's scheme with modulus switching to control noise growth without complex decryption circuits. When performing multiplication operations, the ciphertext dimension is reduced by the key switching technique and the noise is reduced by the modulus switching technique for the following operations. The main advantage of the BGV scheme is that it does not require bootstrapping, which is a computationally expensive operation in FHE schemes. However, the BGV scheme has some limitations, for instance, slow decryption since it uses a larger ciphertext size and the rounding operation.

3.2.2 BFV Scheme

The BFV [13], [14] scheme is an FHE scheme that is based on the hardness of the LWE problem. This scheme was proposed in [14] that discusses a fully homomorphic encryption scheme [13] based on the ring-LWE problem. They introduce optimized versions of relinearization or key switching that result in smaller keys and faster computations and provide a detailed analysis of homomorphic operations including multiplication, relinearization, and bootstrapping, with tight worst-case bounds on noise. Unlike BGV, BFV does not have to use the modulus switching technique to control ciphertext noise, but it still uses the key switching technique to solve the issue of the ciphertext dimension expansion problem caused by multiplication operations.

3.2.2.1 Basic Scheme Description

The BFV scheme [13], [14] consists of the following steps.

1. Setup: The BFV scheme involves choosing several parameters that are similar to the BGV scheme, including a security parameter λ, a modulus q, and a polynomial ring with degree n. Let $\mathcal{R}$ be a ring defined as $\mathcal{R} = \mathbb{Z}[x]/\langle x^d + 1\rangle$, where d is a power of 2. The plaintext space is denoted as $\mathcal{R}_t$, where t is an integer greater than 1. The result of dividing q by t and rounding down to the nearest integer denoted as $\Delta = \lfloor q/t \rfloor$, the remainder when q is divided by t denoted as $r_t(q) = q \mod t$, then $q = \Delta \cdot t + r_t(q)$. t and q do not need to be prime, and they do not have to be coprime.

2. KeyGen(1^λ): Sample $\boldsymbol{s}$ from a distribution $\mathcal{X}$ as the secret key sk, then the public key is given by $pk = ([-(\boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e})]_q, \boldsymbol{a})$, where $\boldsymbol{a}$ is sampled from $\mathcal{R}_q$ and $\boldsymbol{e}$ is sampled from a distribution $\mathcal{X}$.

3. Enc($pk, \boldsymbol{m}$): Encrypts a random message from $\mathcal{R}_t$ using the public key pk, let $\boldsymbol{p}_0 = pk[0], \boldsymbol{p}_1 = pk[1]$ and sample $\boldsymbol{u}, \boldsymbol{e_1}, \boldsymbol{e_2}$ from the distribution $\mathcal{X}$, and then the ciphertext is

$$\boldsymbol{c} = ([\boldsymbol{p}_0 \cdot \boldsymbol{u} + \boldsymbol{e}_1 + \Delta \cdot \boldsymbol{m}]_q, [\boldsymbol{p}_1 \cdot \boldsymbol{u} + \boldsymbol{e}_2]_q)\,.$$

4. Dec($sk, \boldsymbol{c}$): Decrypting the ciphertext $\boldsymbol{c}$ using the secret key sk, and let $\boldsymbol{z}_0 = c[0], \boldsymbol{z}_1 = c[1]$ to recover the message

$$\boldsymbol{m} = \left[\left\lfloor \frac{t \cdot [\boldsymbol{z}_0 + \boldsymbol{z}_1 \cdot \boldsymbol{s}]_q}{q} \right\rceil \right]_t .$$

5. Add: Let $\boldsymbol{c}_1, \boldsymbol{c}_2$ denote the two ciphertexts, then the addition property can be represented by

$$\boldsymbol{c}_{add} = \boldsymbol{c}_1 + \boldsymbol{c}_2 = ([\boldsymbol{z}_1[0] + \boldsymbol{z}_2[0]]_q, [\boldsymbol{z}_1[1] + \boldsymbol{z}_2[1]]_q).$$

6. Mult: The multiplication has two steps, first, the ciphertexts $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ are multiplied and scaled by t/q, and second, the relinearization technique is applied since the resultant ciphertext is composed of three ring elements,

rather than two. In the relinearization process, a relinearization key $\boldsymbol{rlk} = ([-(\mathbf{a} \cdot \mathbf{s} + \mathbf{e}) + p \cdot \mathbf{s}^2]_{p \cdot q}, \mathbf{a})$ is generated, where $\boldsymbol{a} \in \mathcal{R}_{p \cdot q}$, $\boldsymbol{e}$ sampled from the variance of $\mathcal{X}'$, $\boldsymbol{s}$ sampled from $\mathcal{R}_2$, q is the modulus and p is an integer.

Then, if $\boldsymbol{c}_m = \boldsymbol{c}_1 \otimes \boldsymbol{c}_2 = [\boldsymbol{z}_0, \boldsymbol{z}_1, \boldsymbol{z}_2]$, the relinearization process computes

$$(\mathbf{z}_{2,0}, \boldsymbol{z}_{2,1}) = \left(\left[\left\lfloor \frac{\mathbf{z}_2 \cdot \mathbf{rlk}[0]}{p} \right\rceil \right]_q, \left[\left\lfloor \frac{\mathbf{z}_2 \cdot \mathbf{rlk}[1]}{p} \right\rceil \right]_q \right)$$

and then $\boldsymbol{c}_m = ([\mathbf{z}_0 + \mathbf{z}_{2,0}]_q, [\mathbf{z}_1 + \mathbf{z}_{2,1}]_q)$.

For an FHE scheme, it is difficult to perform operations many times since the noise in the ciphertext grows with each homomorphic operation and then the ciphertexts become too noisy to decrypt accurately. The BFV scheme addresses this issue with relinearization technique, which makes the ciphertext rescaled to reduce the noise.

3.2.3 GSW Scheme

The GSW scheme [39] was proposed by Craig Gentry, Amit Sahai and Brent Waters in 2013. It is an FHE scheme based on the learning with errors (LWE) problem, which introduces an approximate eigenvector method to make homomorphic addition and multiplication simpler and faster. Compared with the former schemes like BFV and BGV scheme, GSW has no need for an evaluation key to perform homomorphic operations.

3.2.3.1 Basic Scheme Description

The basic understanding of the encryption scheme [39] is described as follows:

1. Setup($1^\lambda, 1^L$): There are some parameters that need to be chosen including a modulus q with $\kappa = \kappa(\lambda, L)$ bits, the lattice dimension parameter is $n = n(\lambda, L)$, and an error distribution $\mathcal{X} = \mathcal{X}(\lambda, L)$, a parameter $m = m(\lambda, L) = O(n \log q)$. Then, the parameter is $params = (n, q, \mathcal{X}, m)$.Then, $l = \lfloor \log q \rfloor + 1$ and $N = (n+1) \cdot l$.

2. KeyGen($params$): Randomly sample a $\boldsymbol{t} \leftarrow \mathbb{Z}_q^n$, then the secret key is $sk = \boldsymbol{s} \leftarrow (1, -t_1, ..., -t_n) \in \mathbb{Z}_q^{n+1}$. Then, randomly generate a matrix $\boldsymbol{B} \leftarrow \mathbb{Z}_q^{m \times n}$, a vector $\boldsymbol{e} \leftarrow \mathcal{X}^m$, then vector $\boldsymbol{b} = \boldsymbol{B} \cdot \boldsymbol{t} + \boldsymbol{e}$. Set $\boldsymbol{A} = (\boldsymbol{B}, \boldsymbol{b})$ and then $\boldsymbol{A} \cdot \boldsymbol{s} = \boldsymbol{e}$. Then, let the public key is $pk = \boldsymbol{A}$.

3. Enc($params, pk, m$): Encrypting a message $m \in \mathbb{Z}_q$, randomly sample a matrix $\boldsymbol{R} \in \{0,1\}^{N \times n}$ and the ciphertext $\boldsymbol{C}$ is

$$\boldsymbol{C} = Flatten(m \cdot \boldsymbol{I}_N + BitDecomp(\boldsymbol{R} \cdot \boldsymbol{A})) \in \mathbb{Z}_q^{N \times N}$$

where function $Flatten(\boldsymbol{C}) = (Flatten(\boldsymbol{C}_{row1}), ..., Flatten(\boldsymbol{C}_{rowN}))^\top$ is used for applying operations to each row of the matrix, $\boldsymbol{I}_N$ is the identity matrix of N dimension, function $BitDemop(\boldsymbol{a}) = (a_{1,0}, ..., a_{1,l-1}, ..., a_{k,0}, ..., a_{k,l-1}) \in$

$\{0,1\}^N(\boldsymbol{a} = (a_1, ..., a_k) \in \mathbb{Z}_q^k)$ where $a_{i,j}$ is the j-th bit in a_i's binary representation. Then, $\boldsymbol{C} \cdot \boldsymbol{v} = m \cdot \boldsymbol{v} + \boldsymbol{R} \cdot \boldsymbol{A} \cdot \boldsymbol{s} = m \cdot \boldsymbol{v} + \boldsymbol{R} \cdot \boldsymbol{e}$, where $\boldsymbol{v} = PowerSof2(\boldsymbol{s}) = (s_1, 2s_1, ..., 2^{l-1}s_1, ..., s_{n+1}, 2s_{n+1}, ..., 2^{l-1}s_{n+1})$.

4. Dec($params, sk, c$): For the l coefficients of vectors $\boldsymbol{v}$ are $1, 2, ..., 2^{l-1}$, let $v_i = 2^i \in (q/4, q/2]$, $\boldsymbol{C}_{rowi}$ is the i-th row of $\boldsymbol{C}$. Then, the plaintext is $m' = \lfloor x_i/v_i \rceil$ where $x_i \leftarrow \langle \boldsymbol{C}_{rowi}, \boldsymbol{v} \rangle$.

5. Add: m_1, m_2 are two messages with corresponding ciphertexts are $\boldsymbol{C}_1, \boldsymbol{C}_2 \in \mathbb{Z}_q^{N \times N}$. Using $Add(\boldsymbol{C}_1, \boldsymbol{C}_2)$ algorithm to add ciphertexts $\boldsymbol{C}_1, \boldsymbol{C}_2$, and the output is $\boldsymbol{C}_1 + \boldsymbol{C}_2 = (m_1 + m_2) \cdot \boldsymbol{v} + (\boldsymbol{e}_1 + \boldsymbol{e}_2)$.

6. Mult: Using $Mult(\boldsymbol{C}_1, \boldsymbol{C}_2)$ algorithm to multiply $\boldsymbol{C}_1, \boldsymbol{C}_2 \in \mathbb{Z}_q^{N \times N}$, then

$$\begin{aligned} Mult(\boldsymbol{C}_1, \boldsymbol{C}_2) \cdot \boldsymbol{v} &= \boldsymbol{C}_1 \cdot \boldsymbol{C}_2 \cdot \boldsymbol{v} \\ &= \boldsymbol{C}_1 \cdot (m_2 \cdot \boldsymbol{v} + \boldsymbol{e_2}) + m_2 \cdot (m_1 \boldsymbol{v} + \boldsymbol{e_1}) + \boldsymbol{C}_1 \cdot \boldsymbol{e_2} \\ &= m_1 \cdot m_2 \cdot \boldsymbol{v} + (m_2 \cdot \boldsymbol{e_1} + \boldsymbol{C}_1 \cdot \boldsymbol{e_2}). \end{aligned} \tag{3.2}$$

To reduce the new error of $m_2 \cdot \boldsymbol{e_1} + \boldsymbol{C}_1 \cdot \boldsymbol{e_2}$, the message space can be made smaller by setting it to $\{0, 1\}$. This can be achieved using NAND gates in the GSW scheme. By restricting the message space to $\{0, 1\}$, the error term can be reduced, thereby improving the accuracy and efficiency of the encryption scheme. The noise increase of homomorphic multiplication is much larger than homomorphic addition in the FHE scheme, GSW can more efficiently control the noise growth of homomorphic multiplication compared with its previous scheme, but with a larger matrix form ciphertext.

3.2.4 Dowerah and Krishnaswamy's Scheme

The FHE scheme [11] is proposed by Dowerah et.al in 2021, which can evaluate polynomial functions on encrypted data. It uses the technique based on multivariate polynomials and relies on the hardness of the Learning with Errors (LWE) problem for security. For multiplication operations, the scheme uses a polynomial based technique that does not require relineralization or key switching techniques as BGV or BFV schemes. This means that the noise associated with the ciphertext only increases linearly with each multiplication operation, which improves the FHE scheme's efficiency.

3.2.4.1 Basic Scheme Description

Let $\mathcal{R} = \mathbb{Z}_q[x_1, ..., x_v]/\langle x_1^q - x_1, ..., x_v^q - x_v \rangle$ be a polynomial ring, where $q = q(\lambda)$ is a prime number. Consider an ideal $\mathcal{I}$ of the polynomial ring $\mathcal{R}$. Let $r \in \mathbb{N}$, $\mathcal{R}_{\leq r}$ forms a vector space of dimension $N = \binom{v+r}{r}$ over $\mathbb{Z}_q$. Let $\mathcal{I}_{\leq r}$ denote the set of polynomials in $\mathcal{I}$ with degree less than or equal to r. It can be obsolve that $\mathcal{I}_{\leq r}$ is a subspace of $\mathcal{R}_{\leq r}$. Let n be the dimension of $\mathcal{I}_{\leq r}$. It is evident that evaluating

a polynomial $f \in \mathcal{R}_{\leq r}$ at all points of $\mathbb{Z}_q^v$ generates a vector in $\mathbb{Z}_q^{q^v}$. The set of such vectors obtained by evaluating all polynomials in $\mathcal{R}_{\leq r}$ forms an N-dimensional subspace of $\mathbb{Z}_q^{q^v}$. Likewise, evaluating polynomials in $\mathcal{I}_{\leq r}$ yields an n-dimensional subspace of $\mathbb{Z}_q^{q^v}$. Let $\{\boldsymbol{z}_1, ..., \boldsymbol{z}_l\}$ be l distinct points in $\mathbb{Z}_q^v$ for some $l \in \mathbb{N}$ such that $n < l \leq N$. It is always possible to select the l points such that at least n of them are linearly independent. Evaluating polynomials in $\mathcal{I}_{\leq r}$ at $(\boldsymbol{z}_1, ..., \boldsymbol{z}_l)$ yields an n-dimensional subspace $\mathcal{S}_{\mathcal{I}_{\leq r}}$ of $\mathbb{Z}_q^l$ [11].

The plaintext space is $\{0,1\}^{l-n}$ and the ciphertexts are vectors in $\mathbb{Z}_q^l$. Additionally, the scheme is restricted to scenarios where $\mathcal{I}$ is a principal ideal. The scheme [11] description is as follows:

1. Setup$(1^\lambda, 1^L)$: Takes the security parameter λ and the depth of circuits L as input, output parameters $\pi = (n, l, q, \mathcal{X})$, where $n = n(\lambda, L)$, $l = l(\lambda, L)$, modulus $q = q(\lambda, L)$, and noise distribution $\mathcal{X} = \mathcal{X}(\lambda, L)$.

2. KeyGen(π): Selects two positive integers v and r' such that $n = \binom{v + r'}{r'}$. Chooses integers r and r_g such that $r - r_g = r'$. Choose l points, $\boldsymbol{z}_1, ..., \boldsymbol{z}_l$ from $\mathbb{Z}_q^v$ such that every vector in $\mathbb{Z}_q^l$ can be obtained by evaluating a polynomial in $\mathbb{Z}_q[x_1, ..., x_v]_{\leq r}$ at $(\boldsymbol{z}_1, ..., \boldsymbol{z}_l)$ and every vector in $\mathbb{Z}_q^n$ can be obtained by evaluating a polynomial in $\mathcal{I}_{\leq r}$ at $(\boldsymbol{z}_1, ..., \boldsymbol{z}_l)$. A random polynomial $g(x_1, x_2, ..., x_v)$ with degree r_g in v variables such that $g(\boldsymbol{z}_1), ..., g(\boldsymbol{z}_l)$ are all non zero, as a generator of an ideal $\mathcal{I}$. A basis $\mathcal{B}_{\mathcal{I}_{\leq r}} = (gh_1, gh_2, ..., gh_n)$ of $\mathcal{I}_{\leq r}$ is generated, where $h_1, h_2, ..., h_n$ are linearly independent polynomials with degrees not exceeding r'. Construct a basis for the subspace $\mathcal{S}_{\mathcal{I}_{\leq r}}$ by evaluating the polynomials in the basis $\mathcal{B}_{\mathcal{I}_{\leq r}}$ at the given points $(\boldsymbol{z}_1, ..., \boldsymbol{z}_l)$. Construct a basis $\{\tilde{\boldsymbol{s}}_{n+1}, \tilde{\boldsymbol{s}}_{n+2}, ..., \tilde{\boldsymbol{s}}_l\} \in \mathbb{Z}_q^l$ for $(\mathcal{S}_{\mathcal{I}_{\leq r}})^\perp$ which represent by the following form

$$\begin{cases} \tilde{\boldsymbol{s}}_{n+1} & = [s_{n+1}^1 \ \cdots \ s_{n+1}^n \ 1 \ 0 \ \cdots \ 0] \\ \tilde{\boldsymbol{s}}_{n+2} & = [s_{n+2}^1 \ \cdots \ s_{n+2}^n \ 0 \ 1 \ \cdots \ 0] \\ & \vdots \\ \tilde{\boldsymbol{s}}_l & = [s_l^1 \ \cdots \ s_l^n \ 0 \ 0 \ \cdots \ 1] \end{cases} \tag{3.3}$$

and it can be written as $[\boldsymbol{S} \quad \boldsymbol{I}_{l-n}]$, then the matrix $\boldsymbol{S}$ is

$$\boldsymbol{S} = \begin{bmatrix} s_{n+1}^1 & s_{n+1}^2 & \cdots & s_{n+1}^n \\ s_{n+2}^1 & s_{n+2}^2 & \cdots & s_{n+2}^n \\ \vdots & \vdots & \ddots & \vdots \\ s_l^1 & s_l^2 & \cdots & s_l^n \end{bmatrix} \tag{3.4}$$

Then, choose a matrix $\boldsymbol{R} \in \mathbb{Z}_q^{l \times l}$ in the form of

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{R}_1^{n \times n} & \boldsymbol{0}^{n \times (l-n)} \\ \boldsymbol{R}_2^{(l-n) \times n} & \boldsymbol{I}_{l-n} \end{bmatrix} \tag{3.5}$$

where $\boldsymbol{R}_1$ is a randomly chosen full rank matrix from $\mathbb{Z}_q^{n \times n}$ and $\boldsymbol{R}_2$ is randomly chosen from $\mathbb{Z}_q$, $\boldsymbol{I}_{l-n}$ is the identity matrix of size $l - n$. Hence, the secret key is $sk = (\boldsymbol{S}, \boldsymbol{R}_1, \boldsymbol{R}_2)$.

3. Enc$(\pi, sk, \boldsymbol{m})$: To encrypt a message $\boldsymbol{m} \in \{0,1\}^{l-n}$, randomly sample a vector $\boldsymbol{y}$ from $\mathbb{Z}_q^n$, a vector $\boldsymbol{e}$ from $\mathbb{Z}_q^l$ such that $\boldsymbol{e} = (\boldsymbol{0}, e_{n+1}, ..., e_l)$ in which every e_j $(j = n+1, ..., l)$ is randomly selected from the distribution $\mathcal{X}$ and vector $\boldsymbol{0}$ denotes a zero vector of order n, a vector $\boldsymbol{p}$ is given by $\boldsymbol{p} = (\boldsymbol{0}, \boldsymbol{m}) = (\boldsymbol{0}, m_{n+1}, ..., m_l)$ within $\mathbb{Z}_q^l$, then the ciphertext $\boldsymbol{c}$ is

$$\boldsymbol{c} = \left(\boldsymbol{p} \cdot \left\lfloor \frac{q}{2} \right\rfloor + \boldsymbol{y} \cdot \boldsymbol{S}_{enc} + \boldsymbol{e}\right) \boldsymbol{R} \mod q \tag{3.6}$$

within $\mathbb{Z}_q^{n \times n}$, where $\boldsymbol{S}_{enc} = [\boldsymbol{I}_n \quad -\boldsymbol{S}^\top] \in \mathbb{Z}_q^{n \times l}$.

4. Dec$(\pi, sk, \boldsymbol{c})$: Given the ciphertext $\boldsymbol{c}$, the secret key sk as input, output the plaintext $\boldsymbol{m}$ is

$$\boldsymbol{m} = \left\lfloor \frac{1}{\lfloor \frac{q}{2} \rfloor} (\boldsymbol{c} \cdot \boldsymbol{S}_{dec} \mod q) \right\rceil \mod 2 \tag{3.7}$$

where $\boldsymbol{S}_{dec} = \boldsymbol{R}^{-1}[\boldsymbol{S} \quad \boldsymbol{I}_{l-n}]^\top \in \mathbb{Z}_q^{l \times (l-n)}$.

5. Add: For $\boldsymbol{y}_1, \boldsymbol{y}_2 \in \mathbb{Z}_q^n$, assume two ciphertexts $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ are

$$\boldsymbol{c}_1 = \left(\boldsymbol{p}_1 \cdot \lfloor \frac{q}{2} \rfloor + \boldsymbol{y}_1 \cdot \boldsymbol{S}_{enc} + \boldsymbol{e}_1\right) \boldsymbol{R} \mod q \tag{3.8}$$

and

$$\boldsymbol{c}_2 = \left(\boldsymbol{p}_2 \cdot \lfloor \frac{q}{2} \rfloor + \boldsymbol{y}_2 \cdot \boldsymbol{S}_{enc} + \boldsymbol{e}_2\right) \boldsymbol{R} \mod q \tag{3.9}$$

respectively. Then, the addictive result $\boldsymbol{c}_{add}$ is

$$\begin{aligned} \boldsymbol{c}_{add} &= \boldsymbol{c}_1 + \boldsymbol{c}_2 (mod\, q) \\ &= \left((\boldsymbol{p}_1 + \boldsymbol{p}_2) \cdot \lfloor \frac{q}{2} \rfloor + (\boldsymbol{y}_1 + \boldsymbol{y}_2) \cdot \boldsymbol{S}_{enc} + (\boldsymbol{e}_1 + \boldsymbol{e}_2)\right) \boldsymbol{R} (mod\, q). \end{aligned} \tag{3.10}$$

6. Mult: To perform homomorphic multiplication, a bilinear map on ciphertexts $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ is used, represented by a 3-way tensor $\mathcal{M}$. Choose $(n_1 - n)$ additional points, denoted by $\boldsymbol{z}_{l+1}, \ldots, \boldsymbol{z}_t$, such that every vector in $\mathbb{Z}_q^{n_1}$ can be obtained by evaluating a polynomial in $\mathcal{I} \leq 2r$ on the set of points $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n, \boldsymbol{z}_{l+1}, \ldots, \boldsymbol{z}_t$. By evaluating the polynomials in $\mathcal{I}_{\leq r}$ on the points $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_t$, we obtain an n_1-dimensional subspace of $\mathbb{Z}_q^t$. Process the homomorphic multiplication by transforming the ciphertexts $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ to vectors with $K_{1,j}, K_{2,j} \in \mathbb{Z}$, performing the bilinear map on these vectors to obtain a new vector $\tilde{\boldsymbol{c}}_{mult}$, and then transforming $\tilde{\boldsymbol{c}}_{mult}$ back into a ciphertext $\boldsymbol{c}_{mult}$ by adding noise and multiplying by the matrix $\boldsymbol{R}$. Then, the multiplicative result $\boldsymbol{c}_{mult}$ is

$$\boldsymbol{c}_{mult} = \left((\boldsymbol{p}_1 \odot \boldsymbol{p}_2) \cdot \lfloor \frac{q}{2} \rfloor + \boldsymbol{y}_{mult} \cdot \boldsymbol{S}_{enc} + \boldsymbol{e}_{mult}\right) \boldsymbol{R} (mod\, q) \in \mathbb{Z}_q^l. \tag{3.11}$$

4

Implementation

In this chapter, a thorough explanation is given regarding the implementation of the project. This includes the choice of programming language and libraries, the parameters that were established, the primary functions in the implemented code, and the outcomes of the testing.

4.1 Implementation Method

As previously mentioned, we choose to implement the leveled FHE scheme proposed by Dowerah and Krishnaswamy in 2021 [11]. The scheme is based on the LWE problem and uses a multiplication technique without the expensive relinearization procedure. In this scheme, the noise related to the ciphertext raises only linearly with every multiplication. This is a newer scheme that has not been implemented before.

We implemented the scheme by following these steps:

1. We conducted a literature study to gain a comprehensive understanding of the scheme's theoretical principles.

2. In the design phase, we selected appropriate programming languages and libraries and familiarized ourselves with the necessary components.

3. The encoding stage involved setting the parameters, implementing the encryption and decryption algorithms, and addition operations.

4.2 Programming Language and Libraries

Python has been chosen as the implementation language due to the availability of libraries that can be imported into Python. SageMath was selected for the implementation plan, as it can be used as a Python package. SageMath [43] is a free open-source mathematics software system that provides support for a wide range of mathematical applications. It includes algebra, calculus, linear algebra, number theory, and cryptography, making it a suitable choice for implementing an encryption scheme.

NumPy [44] was also selected as another library, as it is commonly used for n-

dimensional arrays and matrices, which are required for the implementation. SymPy [45] was also chosen as a Python library for working with mathematical expressions. It provides tools for manipulating mathematical expressions and performing symbolic computations.

4.3 Parameters

When implementing a cryptographic scheme into practice, the parameter setup phase is crucial since it determines the security and efficiency of the scheme. Properly selected parameters are necessary for key generation, encryption, decryption, and other operations.

In cryptography, the key size refers to the length of the key used in the encryption scheme, which is usually measured in bits and it is the number of possible keys that can be used in the scheme. For example, a key size value equal to 84 bits means that there are 2^{84} possible keys that can be used in an encryption scheme. In the scheme [11], the key size equals the value of security parameter λ, which is used to determine the strength of cryptographic algorithms and systems, that is, the value of λ determines the security level of the system. A higher value of λ corresponds to a higher level of security, as it makes it more difficult for an attacker to break the encryption. However, a higher value of λ also typically requires more computational resources and can make the system slower or less efficient. To ensure adequate security in our implementation, we can select a security parameter λ value greater than 84 since the smallest estimated security parameter is 84 [46]. According to [47], for a symmetric cryptosystem with $56 + b$ bits of the key length and if it does not have any known vulnerabilities, it can provide sufficient security until the year $1982 + y$ where the value of b satisfies the condition $3b \geq 2y$. In other words, until the year 2023, y is equal to 41 and then b needs to be greater than or equal to 28. Therefore, the minimum bit length of the key is equal to 84 to ensure the security of the cryptosystem. Hence, any value of λ greater than or equal to 84 can be considered a successful implementation. Hence, We have decided to set the parameter $\lambda \geq 128$ to ensure that it is sufficiently secure.

Based on the value of λ, the parameter n is set equal to the value of λ since $n = \mathcal{O}(\lambda)$. The parameter l is set equal to $n + 10$ since $l - n = \mathcal{O}(1)$ which means that l is slightly greater than n. The parameter L is set equal to $\log_2 n$ since $L = \mathcal{O}(\log_2 n)$. The modulus q is set equal to $2^{L \log_2 n}$ since q is the bit size $\mathcal{O}(L \log_2 n)$. Then, set two integers v and r' such that $n = \binom{v + r'}{r'}$. The parameter r_g is an integer that we set equal to 1 and then the parameter r is set equal to $r' + r_g$ since $r - r_g = r'$. The parameter N is set equal to $N = \binom{v + r}{r}$. The maximum value of the bound B of the distribution $\mathcal{X}$ is set to $q/(4L)$ for L additions and the minimum value of B needs to satisfy the condition $B \geq \omega(\log n) \cdot \sqrt{n}$, where $\omega(\log n) = 1$ since it represents the logarithmic factor can assume to 1, to ensure the existence of an

efficiently sampleable B-bounded distribution $\mathcal{X}$ [11].

The parameter settings are shown in detail in table 4.1.

Table 4.1: Chosen Parameters and Example Values.

Parameters	Selection method	Example chosen value
λ	80,128,256,...	128
n	$n = \lambda$	128
l	$n + 10$	138
L	$\log_2 n$	7
$\log_2 q$	$L \log_2 n$	49
v, r'	$n = \binom{v + r'}{r'}$	$v = 127, r' = 1$
r_g	an integer	1
r	$r = r' + r_g$	2
N	$N = \binom{v + r}{r}$	8256
B	$\omega(\log n) \cdot \sqrt{n} \leq B \leq \frac{q}{(4L)}$	12

4.4 Main Functions

In this part, we demonstrate the implementation details from parameter setting, key generation, encryption, decryption, and homomorphic additive operations.

4.4.1 Parameters Setting

```
def generate_params(lambda_val):
    n = lambda_val
    l = n + 10
    L = int(math.log2(n))
    q = pow(2,(L * int(math.log2(n))))
    q = sp.nextprime(q)
    number_of_messages = L
    #randomly selected two messages to add
    index1 = random.randint(1,number_of_messages)
    index2 = random.randint(1,number_of_messages)

    print("n =", n)
    print("l =", l)
    print("L =", L)
    print("q =", q)
```

```
    print("number_of_messages =", number_of_messages)
    print(f"message {index1} & {index2} test addition operation")
    return n,l,L,q,number_of_messages,index1,index2

def generate_B_bounded(n):
    B = math.ceil(math.sqrt(n))
    print("B:", B)
    return B
```

The function $generate\_params(lambda_val)$ takes λ as input to generate other parameters, including $n, l, L, q, numer\_of\_messages$. Then, taking n as input in the function $generate\_B\_bounded(n)$ to generate the value of B such that the distribution $|\mathcal{X}| \leq B$ as mentioned in the 4.3.

4.4.2 Key Generation

```
def generate_v_r_apostrophe(n,l):
    for v in range(1, n):
        for r_apostrophe in range(1, n):
            result = int(math.factorial(v + r_apostrophe) /
            (math.factorial(v) * math.factorial(r_apostrophe)))
            if result == n:
                break
    r_g = 1
    r = r_apostrophe + r_g
    N = int(math.factorial(v + r) / (math.factorial(v) *
    math.factorial(r)))
    return v,r_apostrophe,r_g,r,N
```

The function $generate\_v\_r\_apostrophe(n, l)$ takes n, l value from the previous function as input to generate v, r', r_g, r, N.

```
def generate_g(v,r_g):
    R = PolynomialRing(ZZ, ['x{}'.format(i) for i in range(1, v+1)])
    g = [R.random_element(r_g)]
    I = R.ideal(g)
    return g

def gen_h(n,r_apostrophe):
    x = symbols('x:{}'.format(n+1))
    monomials = []
    for i in range(r_apostrophe+1):
        for indices in combinations_with_replacement(range(1,n),i):
```

```
            monomial = 1
            for i in indices:
                monomial *= x[i]
            monomials.append(monomial)
    return monomials
```

The function $generate\_g(v, r_g)$ is used to generate a $g = (x_1, x_2, ...x_v)$ that is a random polynomial with degree r_g. The first line is used to create a random polynomial ring with v variables $x_1, x_2, ..., x_v$ and the third line is used to generate an ideal $\mathcal{I}$ takes g as its generator. Then, the function $gen\_h(n, r\_apostrophe)$ is used to generate linearly independent monomials with the maximum degree r'.

```
#generating matrix S
#generate the l points z_1,z_2,...,z_l
def distinctvector(l,v,q):
    Zn=[]
    for i in range(l):
        z=[]
        for j in range(v):
            z.append(randint(-floor(q/2),floor(q/2)))
        Zn.append(z)
    return Zn

#create a basis for vector space
def basis_S(Zn,l,v,q,ba,n):
    Vspace_basis = []
    for i in ba:
        vec=[]
        for j in Zn:
            values = {f'x{k+1}': j[k] for k in range(v)}
            # create a dictionary of variable values
            vec.append(i(**values))
        Vspace_basis.append(vec)
    M = Matrix(ZZ,Vspace_basis)
    B = [[0 for x in range(l)] for y in range(n)]
    for i in range(0,n):
        for j in range(0,l):
            B[i][j] = modular(M[i][j],q)
    B = matrix(ZZ,B).transpose()
    return B

def split_B1(B,q,n):
    B1 = Matrix(GF(q),B[:n,:n])
    return B1
```

```
def split_B2(B,q,n):
    B2 = -Matrix(GF(q),B[n:,:])
    return B2

#use for generating matrix S
def generate_eye_matrix(l,n):
    vs = []
    V = Matrix.identity(l-n)
    vs.append(V)
    matrix = V
    list_of_lists = [list(row) for row in matrix]
    return list_of_lists

#generate S1,S2,...
def generate_S(Zn,l,n,v,q,ba):
    list_of_lists = generate_eye_matrix(l,n)
    B = basis_S(Zn,l,v,q,ba,n)
    B1 = split_B1(B,q,n)
    B2 = split_B2(B,q,n)
    BB = Matrix(GF(q),B)
    X = []
    j = 1
    m = 1
    for row in list_of_lists:
        S_j = []
        sage_vec = vector(ZZ,list(B1.solve_left(vector(ZZ,row)*B2))
        + list(row))
        S_j = vector(ZZ,[modular(sage_vec[k],q) for k in range(l)])
        if S_j in kernel(BB):
            Bol = True
            #print(f"S_{j} is in Kernel(Matrix(GF(q),B))?",Bol)
            X.append(S_j)
        else:
            Bol = False
            #print(f"S_{j} is in Kernel(Matrix(GF(q),B))?",Bol)
        j = j + 1
    lst = X
    new_list = [[x] for x in lst]
    z = len(new_list)
    my_matrix = Matrix([[j for j in i[0]] for i in new_list])
    S = my_matrix[:z,:(l-z)]
    return S
```

These functions generate the matrix $\boldsymbol{S}$ to generate the secret key sk in the following function.

```
def generate_key(n,q,l,S):
    M = MatrixSpace(GF(q), n, n)
    R1 = M.random_element()
    while R1.rank() != n:
        R1 = M.random_element()
    R2 = Matrix(GF(q), l-n, n, lambda i,j: randint(0, q-1))
    I = identity_matrix(GF(q), l-n)
    R = block_matrix([[R1, 0], [R2, I]])
    sk = (S, R1, R2)
    _, R1, R2 = sk
    return sk, S, R
```

4.4.3 Encryption

```
#generate y1,y2,...
def generate_y_vectors(number_of_messages,n,q):
    y_vectors = []
    for i in range(number_of_messages):
        y = [random.randint(0, q-1) for _ in range(n)]
        y_vectors.append(y)
    return y_vectors

#generate m1,m2,...
def generate_m_vectors(number_of_messages,l,n):
    m_vectors = []
    for i in range(number_of_messages):
        m = [random.randint(0, 1) for _ in range(l - n)]
        m_vectors.append(m)
    return m_vectors

#generate e1,e2,...
#generate zero
def zero(n):
    listofzeros = [0] * n
    return listofzeros

def generate_e_vector(n, l, B_bounded):
    e = [0] * n + [int(random.randint(0, B_bounded)) for _ in
    range(n+1, l+1)]
    return e

def generate_mult_e_vectors(number_of_messages,n,l,B_bounded):
```

```
    e_vectors = []
    for i in range(number_of_messages):
        e_vectors.append(generate_e_vector(n, l, B_bounded))
    return e_vectors

#genereate p1,p2,...
def generate_p_vectors(zero, m_vectors, n):
    zero = zero(n)
    p_vectors = []
    for m in m_vectors:
        p = zero + m
        p_vectors.append(p)
    return p_vectors

#generate S_enc
def neg_S_Transpose(S):
    neg_S_T = - matrix(ZZ,S).transpose()
    return neg_S_T

def generate_S_enc(S,n):
    neg_S_T = - matrix(ZZ,S).transpose()
    I_n = generate_eye_matrix(2*n,n)
    #concatenate I_n and neg_S_T to get S_enc
    S_enc = np.hstack((I_n, neg_S_T))
    return S_enc

#got R
def matrix_R(R):
    R = np.array(R)
    return R

#generate cipertexts c1,c2,...
def generate_ciphertext(pt,y,e,R,S_enc,q,l):
    y = vector(y)
    S_enc = matrix(S_enc)
    result = y * S_enc
    c1 = vector(np.dot(pt,floor(q/2)) +
    modular_vector(result,l,q) + e)
    R = matrix(R)
    re = c1 * R
    c = modular_vector(re,l,q)
    return c

def generate_ciphertexts(p_vectors, y_vectors, e_vectors,
number_of_messages,S_enc,R,q,l):
    ciphertexts = []
```

```
    for i in range(number_of_messages):
        c = generate_ciphertext(p_vectors[i], y_vectors[i],
        e_vectors[i],R,S_enc,q,l)
        ciphertexts.append(c)
    return ciphertexts
```

These functions generate vectors $\boldsymbol{y}, \boldsymbol{e}, \boldsymbol{p}$ and the encryption key $\boldsymbol{S}_{enc}$ to encrypt messages into ciphertexts.

4.4.4 Decryption

```
#generate S_dec
def generate_S_dec(l,n,S,q,R):
    I = identity_matrix(GF(q), l-n)
    #concatenate S and I vertically
    S_ext = np.hstack((S,I))
    S_dec = Matrix(GF(q),np.dot(Matrix(R).inverse(), S_ext.T))
    return S_dec

def round_vector(v,q):
    return [round(float(x)/int(floor(q/2))) for x in v]

def decrypt_ciphertexts(ciphertexts, S_dec,l,n,q):
    m_decrypt_list = []
    for c in ciphertexts:
        c = vector(c)
        S_dec = matrix(S_dec)
        result = c * S_dec
        m_decrypt = vector(ZZ,
        round_vector(modular_vector(result, l - n, q), q))
        m_decrypt = [x % 2 for x in m_decrypt]
        m_decrypt_list.append(m_decrypt)
    return m_decrypt_list
```

The function $generate\_S\_dec(l, n, S, q, R)$ is used to generate the decryption key $\boldsymbol{S}_{dec}$ to decrypt ciphertexts into plaintexts.

4.4.5 Homomorphic addition operation

```
#save every message in a dictionary
def list_to_messages(m_vectors):
```

```
    messages = {}
    for i in range(len(m_vectors)):
        key = 'm{}'.format(i+1)
        messages[key] = m_vectors[i]
    return messages

#add two randomly selected messages as m_add
def add_messages(m_vectors, index1, index2):
    m1 = m_vectors[index1-1]
    m2 = m_vectors[index2-1]
    m_add = []
    for i in range(len(m1)):
        m_add.append(m1[i] + m2[i])
    return m_add

#add all messages as m_add_all
def add_messages_all(m_vectors):
    m_add_all = list(map(sum, zip(*m_vectors)))
    return m_add_all
```

The above functions are used to calculate message addition results, including two randomly selected messages and all messages.

```
#c_add as the ciphertexts of two messages adding results
def generate_c_add(ciphertexts,index1,index2,l,q):
    c_add = []
    c1 = ciphertexts[index1 - 1]
    c2 = ciphertexts[index2 - 1]
    for i in range(len(c1)):
        c_add.append(c1[i] + c2[i])
    c_add = modular_vector(c_add,l,q)
    return c_add

#add all ciphertexts
def generate_c_add_all(ciphertexts):
    c_add_all = list(map(sum, zip(*ciphertexts)))
    return c_add_all
```

The results of adding ciphertexts are used to decrypt data directly, which demonstrates the addition homomorphic properties.

Apart from the main functions, the full version has been uploaded to GitHub, the git repository address is https://github.com/lalalaaaaaaaaaa/git-thesis-ptoject.

4.5 Testing and Results

4.5.1 Testing Functions

```
#test for decryption and addition
def test_dec(m_decrypt_list,m_vectors):
    if m_decrypt_list == m_vectors:
        print('\n')
        print("Dec_each_messages_Correct:",'\n',"messages:",'\n',
        m_vectors,'\n',"decrypt ciphertext:",'\n',m_decrypt_list)
    return

def test_add(m_add_decrypt,m_add_mod2,index1,index2):
    if m_add_decrypt == [m_add_mod2]:
        print("ADD_two_messages_Correct:",'\n',f"two_messages_add
        (m {index1} & m{index2}):",'\n',[m_add_mod2],
          '\n',"two_ciphertexts_add_decrypt:",'\n',
          m_add_decrypt)
    return

def test_add_all(m_add_all_decrypt,m_add_all_mod2):
    if m_add_all_decrypt == [m_add_all_mod2]:
        print("ADD_all_messages_Correct:",'\n',[m_add_all_mod2],
          '\n',"all_ciphertexts_add_decrypt:",'\n',
          m_add_all_decrypt)
    return
```

These functions are designed to verify the accuracy of the decryption process by checking if the decrypted plaintexts match the original messages as mentioned in 17. If the decrypted plaintexts are identical to the corresponding messages, it indicates that the encryption and decryption processes have been implemented correctly.

4.5.2 Results

We give an example taking $\lambda = 128$. Then, the results are as follows:

```
n = 128
l = 138
L = 7
q = 562949953421381
number_of_messages = 7
message 6 & 7 test addition operation
v, r_apostrophe: 127 1
```

```
r_g, r: 1 2
N: 8256
g: [-5*x83 + 2*x111 - x114]
h: [1, x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11, x12, x13,
x14, x15, x16, x17, x18, x19, x20, x21, x22, x23, x24, x25, x26,
x27, x28, x29, x30, x31, x32, x33, x34, x35, x36, x37, x38, x39,
x40, x41, x42, x43, x44, x45, x46, x47, x48, x49, x50, x51, x52,
x53, x54, x55, x56, x57, x58, x59, x60, x61, x62, x63, x64, x65,
x66, x67, x68, x69, x70, x71, x72, x73, x74, x75, x76, x77, x78,
x79, x80, x81, x82, x83, x84, x85, x86, x87, x88, x89, x90, x91,
x92, x93, x94, x95, x96, x97, x98, x99, x100, x101, x102, x103,
x104, x105, x106, x107, x108, x109, x110, x111, x112, x113,
x114, x115, x116, x117, x118, x119, x120, x121, x122, x123,
x124, x125, x126, x127]
```

The first part of the output displays the parameter configurations outlined in 4.3. Additionally, two specific messages, namely 6 and 7, have been randomly selected to verify the addition homomorphic property.

The second part of the output displays the error vectors generated for the encryption process. These errors are random numbers of the B-bounded value added during the encryption process to ensure security. The decryption function removes these errors from the ciphertext to recover the original message. If the decryption function fails to remove the errors correctly, the decrypted message will differ from the original message, resulting incorrect decryption. The error values displayed in the output help in debugging the implementation of the encryption and decryption functions.

```
B: 12
errors: [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5,
5, 7, 9, 6, 9, 6, 0, 1, 8],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 11, 7, 7,
12, 4, 2, 3, 6, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 11, 1, 8,
10, 5, 0, 10, 0, 5],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 3, 9, 11,
10, 8, 8, 12, 3, 12],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 2, 4, 7,
1, 7, 11, 2, 12, 4],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 12, 10, 10,
4, 8, 0, 3, 6, 9],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 10, 7, 7,
12, 2, 0, 2, 9, 8]]
```

The third part includes the following information:

1. The 7 original messages and their corresponding decrypted ciphertexts. For example, the message $m_1 = [1, 0, 1, 1, 0, 1, 1, 1, 1, 0]$ and the corresponding decrypted ciphertext is $[1, 0, 1, 1, 0, 1, 1, 1, 1, 0]$, which means it is correctly decryption.

2. The addition of messages $m\_add(m_6, m_7) = [1, 2, 2, 1, 1, 1, 1, 1, 0, 1]$ and the corresponding decrypted ciphertexts is $m\_add\_decrypt = m\_add(m_6, m_7)($ mod 2$) = [1, 0, 0, 1, 1, 1, 1, 1, 0, 1]$, which means it is correctly decryption for the addition operation.

3. The addition of all 7 messages is $m\_add\_all = [4, 5, 6, 6, 3, 4, 5, 3, 2, 2]$, and the corresponding decrypted ciphertexts is $m\_add\_all\_decrypt = m\_add\_all($ mod 2$) = [0, 1, 0, 0, 1, 0, 1, 1, 0, 0]$, which shows the addition homomorphic property.

The final part is the testing results to verify the correctness of this implementation. It checks whether the decrypted plaintexts are the same as the corresponding original messages. If they match, it indicates that the basic encryption, decryption operations and the addition homomorphic property are correctly implemented. This is an important step to ensure the homomorphic encryption scheme is implemented.

```
messages: {'m1': [1, 0, 1, 1, 0, 1, 1, 1, 1, 0], 'm2': [0, 0, 0,
1, 0, 1, 1, 1, 0, 0], 'm3': [0, 1, 1, 1, 0, 1, 1, 0, 0, 1],
'm4': [1, 1, 1, 1, 1, 0, 0, 0, 1, 0], 'm5': [1, 1, 1, 1, 1, 0,
1, 0, 0, 0], 'm6': [1, 1, 1, 1, 1, 1, 0, 0, 0, 0], 'm7': [0, 1,
1, 0, 0, 0, 1, 1, 0, 1]}
Decrypted ciphertexts: [[1, 0, 1, 1, 0, 1, 1, 1, 1, 0], [0, 0,
0, 1, 0, 1, 1, 1, 0, 0], [0, 1, 1, 1, 0, 1, 1, 0, 0, 1], [1, 1,
1, 1, 1, 0, 0, 0, 1, 0], [1, 1, 1, 1, 1, 0, 1, 0, 0, 0], [1, 1,
1, 1, 1, 1, 0, 0, 0, 0], [0, 1, 1, 0, 0, 0, 1, 1, 0, 1]]
m_add(m6 & m7): [1, 2, 2, 1, 1, 1, 1, 1, 0, 1]
m_add_decrypt: [[1, 0, 0, 1, 1, 1, 1, 1, 0, 1]]
m_add_all: [4, 5, 6, 6, 3, 4, 5, 3, 2, 2]
m_add_all_decrypt: [[0, 1, 0, 0, 1, 0, 1, 1, 0, 0]]

Dec_each_messages_Correct:
 messages:
 [[1, 0, 1, 1, 0, 1, 1, 1, 1, 0], [0, 0, 0, 1, 0, 1, 1, 1, 0,
 0], [0, 1, 1, 1, 0, 1, 1, 0, 0, 1], [1, 1, 1, 1, 1, 0, 0, 0, 1,
 0], [1, 1, 1, 1, 1, 0, 1, 0, 0, 0], [1, 1, 1, 1, 1, 1, 0, 0, 0,
 0], [0, 1, 1, 0, 0, 0, 1, 1, 0, 1]]
 decrypt ciphertext:
 [[1, 0, 1, 1, 0, 1, 1, 1, 1, 0], [0, 0, 0, 1, 0, 1, 1, 1, 0,
 0], [0, 1, 1, 1, 0, 1, 1, 0, 0, 1], [1, 1, 1, 1, 1, 0, 0, 0, 1,
 0], [1, 1, 1, 1, 1, 0, 1, 0, 0, 0], [1, 1, 1, 1, 1, 1, 0, 0, 0,
 0], [0, 1, 1, 0, 0, 0, 1, 1, 0, 1]]
ADD_two_messages_Correct:
 two_messages_add(m 6 & m7):
 [[1, 0, 0, 1, 1, 1, 1, 1, 0, 1]]
 two_ciphertexts_add_decrypt:
 [[1, 0, 0, 1, 1, 1, 1, 1, 0, 1]]
ADD_all_messages_Correct:
 [[0, 1, 0, 0, 1, 0, 1, 1, 0, 0]]
 all_ciphertexts_add_decrypt:
 [[0, 1, 0, 0, 1, 0, 1, 1, 0, 0]]
```

4.5.3 Runtimes

In this part, we provide benchmarks for the various functions of the implemented scheme. The experiments were conducted on a standard laptop equipped with a 2.9 GHz Dual-Core Intel Core i5 processor, Intel Iris Graphics 6100 with 1536 MB, and

8 GB of 1867 MHz DDR3 memory.

In table 4.2, we present the results of the experiments conducted five times for $\lambda = 128$. Each experiment measures the execution time of the implemented scheme. Table 4.3 displays the average execution time obtained from the five experiments conducted for $\lambda = 128$. This average provides a representative measure of the overall performance of the scheme at this security parameter.

Similarly, in table 4.4, we present the results of the experiments conducted five times for $\lambda = 256$. Each experiment measures the execution time of the implemented scheme at this higher security parameter. Then, table 4.5 shows the average execution time obtained from the five experiments conducted for $\lambda = 256$. This average serves as a comprehensive measure of the scheme's performance at this increased security level.

Table 4.2: Runtime of the Implementation($\lambda = 128$).

λ	$\log_2 q$	L	KeyGen	Encryption	Decryption	Addition
128	49	7	56354 ms	822 ms	3350 ms	4 ms
128	49	7	46052 ms	591 ms	2434 ms	8 ms
128	49	7	41084 ms	621 ms	2336 ms	3 ms
128	49	7	42016 ms	665 ms	2739 ms	10 ms
128	49	7	47053 ms	782 ms	2933 ms	5 ms

Table 4.3: Average Runtime of the Implementation($\lambda = 128$).

λ	$\log_2 q$	L	KeyGen	Encryption	Decryption	Addition
128	49	7	46512 ms	696 ms	2758 ms	6 ms

Table 4.4: Runtime of the Implementation($\lambda = 256$).

λ	$\log_2 q$	L	KeyGen	Encryption	Decryption	Addition
256	64	8	365066 ms	2390 ms	19242 ms	7 ms
256	64	8	452126 ms	2441 ms	21692 ms	7 ms
256	64	8	422624 ms	2743 ms	22882 ms	6 ms
256	64	8	362227 ms	2407 ms	21485 ms	6 ms
256	64	8	359012 ms	2395 ms	18733 ms	6 ms

4.6 Discussion of Implementation

In this part, the reasons that arise incorrect decryption during our implementation process are discussed to answer the research question RQ2.

Table 4.5: Average Runtime of the Implementation($\lambda = 256$).

λ	$\log_2 q$	L	KeyGen	Encryption	Decryption	Addition
256	64	8	392211 ms	2475 ms	20807 ms	6 ms

The primary challenge in implementing the FHE scheme is the need for a deep understanding of cryptography and complex mathematical concepts such as ideals, lattices, and polynomial rings. The process of applying these concepts to code can be challenging since each FHE scheme have different parameter settings and mathematical computations, therefore, even though many schemes are implemented, we cannot directly borrow ideas from their implementation. Hence, literature review and the study of relevant basic knowledge are crucial for understanding the principles of the scheme that needs to be implemented.

The second challenge we met in implementing the scheme is selecting a proper data type with the appropriate libraries since different libraries with different data types, it can be difficult to perform computations between them. For instance, NumPy is commonly used to create arrays, vectors, or matrices, while SageMath is used to generate polynomials and ideals. To perform computations between results from these two libraries, the data type needs to be converted to the same format for the calculations. Using only one library that supports the required data types and computations can be a better solution to this problem. It can make data type conversion unnecessary and guarantees code consistency.

Besides, the NumPy library has limitations when using it to implement an FHE scheme since the FHE schemes usually require enormous calculations on parameters. When the security parameter λ is set to 84 which is the minimum value that can be acceptable for the implementation of this scheme [11], then the value of $q = 68719476767$ which is an 11 digits number. Therefore, when applying functions like np.dot() to calculate the inner product of the generated vector $\boldsymbol{y} \in \mathbb{Z}_q^n$ and the encryption key $\boldsymbol{S}_{enc} \in \mathbb{Z}_q^{n \times l}$, it results in data overflow. Actually, when the q value exceeds 8 digits, it leads to the wrong results of elements in the vector $\boldsymbol{y} \cdot \boldsymbol{S}_{enc}$. The same problem occurs when calculating $\boldsymbol{c} \cdot \boldsymbol{S}_{dec}$ in decryption. But SageMath can solve this problem and it can perform correct calculations even if we set $\lambda = 256$, which means that SageMath is more suitable for implementing the FHE scheme due to the data limitations problems with NumPy.

Moreover, the major challenge in implementing an FHE scheme is identifying the reasons that may cause incorrect decryption. This is because the success of the entire implementation depends on whether the ciphertext can be decrypted correctly. However, it can be difficult to pinpoint the exact cause of wrong decryption, as they may occur at any part of the parameter setting, key generation, or encryption phases. To address this challenge, we set the security parameter to a small value during the initial implementing phase, so that errors can be easily identified and corrected. Besides, we create test functions to compare the decrypted data results to the original messages to quickly identify whether any errors exist and avoid checking mistakes during each step of the implementation. After the implementation has been tested

and verified with a small security parameter, the value of the security parameter is set to gradually increase to ensure the FHE scheme is functioning correctly.

The multiplication property is not implemented in this project since we meet some challenges. Implementing multiplication in an FHE scheme presents several challenges that make it more difficult compared to addition and basic encryption operations. Firstly, multiplication involves complex operations such as polynomial multiplications or matrix operations. These computations require significant computational resources and time, making it hard to implement. We have generated additional $t-l$ points and the subspace $\mathcal{I}_{\leq 2r}$ for preparing the multiplication operation. Secondly, multiplication amplifies the noise level in the FHE scheme. Noise accumulates with each operation, and multiplication tends to increase the noise further. Managing and mitigating the noise growth during multiplication becomes crucial for a FHE scheme but adds complexity to the implementation. Performing multiple matrix operations adds to the complexity and resource requirements of implementing multiplication. Besides, multiplication in FHE requires larger ciphertexts and increased memory usage compared to addition operations. This puts a strain on memory management, especially when dealing with limited resources or large-scale computations. The larger ciphertext size and increased memory requirements make the implementation of multiplication more difficult.

5

Multi-key Extension

This chapter is dedicated to extending the encryption scheme introduced in [11] to the multi-key setting. The aim is to explore the feasibility of extending Dowerah and Krishnaswamy's single-key FHE scheme into a multi-key scheme. The chapter addresses the RQ3 and demonstrates the potential for expanding the original scheme to support multiple users.

5.1 Multi-key Additively HE scheme

We extend the single key scheme in [11] to a multi key additively homomorphic encryption scheme in the symmetric key setting. The proposed scheme can be described in terms of the following algorithms.

1. MK.Setup$(1^\lambda, 1^L) \to \pi$: Given the security parameter λ and the circuit depth L, output the pubic parameters $\pi = (n, v, r, r', r_g, l, q, \mathcal{X})$ such that $r - r_g = r'$ and $n = \binom{v + r'}{r'}$. Chooses l distinct points, $\boldsymbol{z}_1, ..., \boldsymbol{z}_l$ from $\mathbb{Z}_q^v$. Set $msk = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_l)$. $\mathcal{X}$ is a noise distribution.

2. MK.KeyGen$(\pi, msk) \to sk_j$: This algorithm generates a secret key sk_j for user j for $j = 1, 2, ..., l - n$. Generate a random polynomial $g_j(x_1, x_2, ..., x_v)$ with degree r_g in v variables such that $\mathcal{I}_j = (g_j)$ is an ideal. Generate a basis $\mathcal{B}_j = (g_j h_{j,1}, g_j h_{j,2}, ..., g_j h_{j,n})$ for $\mathcal{I}_{j_{\leq r}}$, where $h_{j,1}, h_{j,2}, ..., h_{j,n}$ are linearly independent polynomials with degrees not exceeding r'. Generate a subspace $\mathcal{S}_j \in \mathbb{Z}_q^l$ by evaluating the basis $\mathcal{B}_j$ at the points $(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_l)$. Generate a vector $\tilde{\boldsymbol{s}}_j$ in $\mathcal{S}_j^\perp$ such that $\tilde{\boldsymbol{s}}_j = (s_j^1, \ldots, s_j^n, 0, \ldots, 0, 1, 0, \ldots, 0)$ where the first n entries are random elements in $\mathbb{Z}_q$ and $n + j$-th entry is 1 with the rest of the entries equal to zero. Let $\boldsymbol{s}_j = (s_j^1, s_j^2, ..., s_j^n) \in \mathbb{Z}_q^n$. Then output the secret key $sk_j = (-\boldsymbol{s}_j, 1)^\top$.

3. MK.Enc$(\pi, sk_j, m) \to ct = \{\boldsymbol{c}, S\}$: Given the secret key sk_j of the j-th user and a message m, select a vector $\boldsymbol{y}_j = (y_{j_1}, y_{j_2}, ..., y_{j_n})$ uniformly at random from $\mathbb{Z}_q^n$ and samples e from the distribution $\mathcal{X}$ and generate a corresponding ciphertext

$$\boldsymbol{c} = (\boldsymbol{c}^0, c^1) = (\boldsymbol{y}_j, m \left\lfloor \frac{q}{2} \right\rfloor + \boldsymbol{y}_j \cdot \boldsymbol{s}_j^\top + e)$$

Let S be an ordered set that contains all the indexes of users corresponding to the ciphertext. We assume that the indexes in S are arranged in ascending

order, and S does not contain any duplicate elements. Then, a ciphertext is represented by a tuple $ct = \{\boldsymbol{c}, S\}$. Here $S = \{j\}$.

4. MK.Dec$(sk_S, ct = (\boldsymbol{c}, S)) \rightarrow m$: Takes a ciphertext $ct = (\boldsymbol{c}, S)$ as input, where $S = \{j_1, \ldots, j_k\}$ and its corresponding secret keys are $sk_{j_1}, ..., sk_{j_k}$. Let $sk_S = (-\boldsymbol{s}_{j_1}, -\boldsymbol{s}_{j_2}, ..., -\boldsymbol{s}_{j_k}, 1)^\top$, then output the message

$$m = \left\lfloor \frac{1}{\lfloor \frac{q}{2} \rfloor} (\langle \boldsymbol{c}, sk_S \rangle \mod q) \right\rceil \mod 2 \tag{5.1}$$

where $\boldsymbol{c} = (\boldsymbol{c}^0_{j_1}, \boldsymbol{c}^0_{j_2}, ..., \boldsymbol{c}^0_{j_k}, c^1_S) \in \mathbb{Z}_q^{k+1}$. Note that $\boldsymbol{c}^0_{j_1} = \boldsymbol{y}_{j_1}, \boldsymbol{c}^0_{j_2} = \boldsymbol{y}_{j_2}, ..., \boldsymbol{c}^0_{j_k} = \boldsymbol{y}_{j_k}$, $c^1_S = m\lfloor \frac{q}{2} \rfloor + \boldsymbol{y}_{j_1}\boldsymbol{s}^\top_{j_1} + \boldsymbol{y}_{j_2}\boldsymbol{s}^\top_{j_2} + ... + \boldsymbol{y}_{j_k}\boldsymbol{s}^\top_{j_k} + e$.

5. MK.Add$(\mathcal{C}, (ct_1, ..., ct_t)) \rightarrow \boldsymbol{c}_{add}$: For $j \in [t]$, parse ct_j as $(\boldsymbol{c}_j, S_j)$, where $|S_j| = k_j$, let $S = \cup_{j=1}^t S_j = \{j_1, ..., j_k\}$, and thus $\boldsymbol{c}_j \in \mathbb{Z}_q^{k_j+1}$. The evaluation of the Boolean circuit $\mathcal{C}$ can be outlined as follows:

   (a) For $j \in [t]$, compute MK.Ext$(\boldsymbol{c}_j, S) = \hat{\boldsymbol{c}}_j$ to obtain extended $k+1$ dimensional ciphertexts and these ciphertexts encrypt the same message under the key $sk_S = (sk_{j_1}, \ldots, sk_{j_k})$.

   (b) Perform homomorphic addition operations on the extended ciphertexts to evaluate each gate of circuit $\mathcal{C}$. This involves adding the extended ciphertexts $\hat{\boldsymbol{c}}_i$ and $\hat{\boldsymbol{c}}_j$ to obtain $\boldsymbol{c}_{add} = \hat{\boldsymbol{c}}_i + \hat{\boldsymbol{c}}_j \ (\mod q)$.

5.1.1 Ciphertext Extension

For ciphertext extension, we use the same technique from [33] which is described as follows.

MK.Ext$(ct = (\boldsymbol{c}, S')) \rightarrow ct'$: Taking a ciphertext tuple $ct = \{\boldsymbol{c} \in \mathbb{Z}_q^{k+1}, S = \{i_1, ..., i_k\}\}$ corresponding to k parties and another user set $S' = \{j_1, ..., j_{k'}\}$ for $S \in S'$ as input, output an extended tuple $ct' = \{\hat{\boldsymbol{c}} \in \mathbb{Z}_q^{k'+1}, S' = \{j_1, ..., j_{k'}\}\}$. The steps of the extending algorithm are as follows:

1. The ciphertext $\boldsymbol{c}$ is divided into $k+1$ sequential sub-vectors indexed by $S = \{i_1, ..., i_k\}$ (except for the last sub-vector). This division is done as follows: $\boldsymbol{c} = (c^0_{i_1} | c^0_{i_2} | ... | c^0_{i_k} | c^1_S)$, where the corresponding secret key is $sk_S = (-\boldsymbol{s}_{i_1}, -\boldsymbol{s}_{i_2}, ..., -\boldsymbol{s}_{i_k}, 1)^\top$.

2. The extended ciphertext $\hat{\boldsymbol{c}}$ consists of $k'+1$ sequential sub-vectors, which can be indexed by $S' = \{j_1, ..., j_{k'}\}$. It is defined as $\hat{\boldsymbol{c}} = (c'^0_{j_1} | c'^0_{j_2} | ... | c'^0_{j_{k'}} | c'^1_{S'})$. Set $c'^1_{S'} = c^1_S$. If an index j in S' is also included in S, we set $c'^0_j = c^0_j$, otherwise, we set $c'^0_j = 0$. The corresponding secret key for decryption is $sk_{S'} = (-\boldsymbol{s}_{j_1}, -\boldsymbol{s}_{j_2}, ..., -\boldsymbol{s}_{j_{k'}}, 1)^\top$.

5.2 Discussion

The multi-key scheme 5.1 generates multiple private keys for different users participating in the encryption scheme. Each user has their own private key for encryption and decryption.

One of the challenges in extending the single key scheme to a multi-key scheme lies in generating appropriate keys for different users. We address this challenge by generating distinct keys using separate ideals. However, this approach introduces additional computational overhead compared to the original single-key FHE scheme. This is because each user's secret key needs to be generated independently, resulting in the generation of $l - n$ ideals for $l - n$ users. Therefore, encryption, computation, and decryption operations require more time and resources due to the involvement of multiple users and their respective keys.

The homomorphic property of multiplication in the single-key scheme 3.2.4 needs a polynomial f_i in an ideal $\mathcal{I}_{\leq r}$ for constructing a ciphertext $\boldsymbol{c}_i$. However, our multi-key approach is not directly applicable to the homomorphic property of multiplication. In the multi-key setting, multiplication requires complex operations like polynomial multiplications and matrix operations, rather than simple multiplication of ciphertexts. Besides, different users need different polynomials generated from different ideals. Therefore, achieving the homomorphic property of multiplication in a multi-key scheme requires exploring alternative approaches and ideas.

Another challenge revolves around the treatment of the matrix $\boldsymbol{R}$ in the original single-key encryption algorithm, which is a component of the secret key. We ignore this matrix in the construction of the multi-key scheme for simplicity but it can be easily incorporated using similar ideas. Another potential future direction is to make the scheme public key. We can use similar ideas from previous work [33] to convert it into a public key scheme.

6
Conclusion

In this chapter, we provide a summary of the work conducted in our thesis and we discuss potential directions for future work.

6.1 Summary of the Results

In this thesis, we provided a brief survey of fully homomorphic encryption (FHE) schemes, focusing on LWE-based FHE schemes. We implemented Dowerah and Krishnaswamy's FHE scheme, which includes basic encryption and decryption operations as well as the homomorphic property of addition. We further extended the scheme to the multi-key setting, enabling additively homomorphic encryption with multiple keys.

In Chapter 3, we provided an overview of various FHE schemes, highlighting those based on the Learning With Errors (LWE) and Ring Learning With Errors (RLWE) problems, such as BGV, BFV, GSW, and Dowerah and Krishnaswamy's FHE schemes.

Chapter 4 presented the detailed implementation process, including parameter selection, library choice, main functions, and testing results. We demonstrated the functionality of the implementation by showcasing message encryption and corresponding decrypted plaintexts. We also discussed encountered challenges during the implementation and proposed potential solutions. Our implementation ensured the accurate decryption of basic encryption and decryption operations, as well as the addition homomorphic property. The insights gained from tackling implementation issues may be applicable not only to this specific scheme but also to the broader context of FHE implementations.

In Chapter 5, we successfully extended Dowerah and Krishnaswamy's FHE scheme to support the multi-key setting for homomorphic addition. We presented a description of the multi-key additively homomorphic encryption scheme. Additionally, we discussed the challenges faced when extending a single-key scheme to a multi-key scheme.

6.2 Scope of Future Work

The work presented in this thesis opens up possibilities for further work including two potential directions for future work are outlined below:

1. Homomorphic Property of Multiplication: In our implementation work, we focused on the basic encryption and decryption operations, as well as the homomorphic property of addition of Dowerah and Krishnaswamy's FHE scheme. However, an important extension would be to implement the homomorphic property of multiplication. This involves performing matrix operations and polynomial calculations, which can be complex and may require a deeper understanding of cryptography and some mathematical knowledge.

2. Multi-Key Multiplicative HE Scheme: Another direction for future work is extending the existing single-key FHE scheme to support a multi-key multiplicative homomorphic encryption (HE) scheme. This would involve finding innovative approaches to enable computations on encrypted data using multiple keys while preserving the multiplicative homomorphic property. It may require careful consideration of security, efficiency, and usability aspects.

Bibliography

[1] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 2018.

[2] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 169–178.

[3] R. L. Rivest, L. Adleman, M. L. Dertouzos, *et al.*, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[4] R. L. Rivest, "A method for obtaining digital signature and public-key cryptosystems," *ACM*, vol. 21, p. 2, 1987.

[5] F. Armknecht, D. Augot, L. Perret, and A.-R. Sadeghi, "On constructing homomorphic encryption schemes from coding theory," in *Cryptography and Coding: 13th IMA International Conference, IMACC 2011, Oxford, UK, December 12-15, 2011. Proceedings 13*, Springer, 2011, pp. 23–40.

[6] F. Armknecht and A.-R. Sadeghi, "A new approach for algebraically homomorphic encryption," *Cryptology ePrint Archive*, 2008.

[7] C. A. Melchor, G. Castagnos, and P. Gaborit, "Lattice-based homomorphic encryption of vector spaces," in *2008 IEEE international symposium on information theory*, IEEE, 2008, pp. 1858–1862.

[8] C. A. Melchor, P. Gaborit, and J. Herranz, "Additively homomorphic encryption with d-operand multiplications," in *Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings 30*, Springer, 2010, pp. 138–154.

[9] C. Peikert and B. Waters, "Lossy trapdoor functions and their applications," in *Proceedings of the fortieth annual ACM symposium on Theory of computing*, 2008, pp. 187–196.

[10] M. Fellows and N. Koblitz, "Combinatorial cryptosystems galore!" *Contemporary Mathematics*, vol. 168, pp. 51–51, 1994.

[11] U. Dowerah, "On lattice based cryptographic algorithms," Ph.D. dissertation, 2021.

[12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.

[13] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Annual Cryptology Conference*, Springer, 2012, pp. 868–886.

[14] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.

[15] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, Springer, 2017, pp. 409–437.

[16] J. Ma, S.-A. Naas, S. Sigg, and X. Lyu, "Privacy-preserving federated learning based on multi-key homomorphic encryption," *International Journal of Intelligent Systems*, 2022.

[17] S. Halevi and V. Shoup, "Design and implementation of helib: A homomorphic encryption library," *Cryptology ePrint Archive*, 2020.

[18] A. C. Mert, E. Öztürk, and E. Savaş, "Design and implementation of encryption/decryption architectures for bfv homomorphic encryption scheme," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 353–362, 2019.

[19] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, 2012, pp. 1219–1234.

[20] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 395–412.

[21] R. Lidl and H. Niederreiter, *Finite fields*. Cambridge university press, 1997.

[22] X. Yi, R. Paulet, E. Bertino, X. Yi, R. Paulet, and E. Bertino, *Homomorphic encryption*. Springer, 2014.

[23] D. Cox, J. Little, D. O'Shea, and M. Sweedler, "Ideals, varieties, and algorithms," *American Mathematical Monthly*, vol. 101, no. 6, pp. 582–586, 1994.

[24] S. Garg, C. Gentry, and S. Halevi, "Candidate multilinear maps from ideal lattices," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2013, pp. 1–17.

[25] M. Yasuda, "A survey of solving svp algorithms and recent strategies for solving the svp challenge," in *International Symposium on Mathematics, Quantum Theory, and Cryptography: Proceedings of MQC 2019*, Springer Singapore, 2021, pp. 189–207.

[26] M. Ajtai, "Generating hard instances of lattice problems," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 99–108.

[27] C. Peikert *et al.*, "A decade of lattice cryptography," *Foundations and Trends® in Theoretical Computer Science*, vol. 10, no. 4, pp. 283–424, 2016.

[28] Y. Lindell, "Foundations of cryptography 89-856," *Electronic document (April 2006)*, 2010.

[29] L. Ducas and A. Durmus, "Ring-lwe in polynomial rings," in *Public Key Cryptography–PKC 2012: 15th International Conference on Practice and The-*

ory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings 15, Springer, 2012, pp. 34–51.

[30] R. Goyal, V. Koppula, and B. Waters, "Separating ind-cpa and circular security for unbounded length key cycles," in *Public-Key Cryptography–PKC 2017: 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part I*, Springer, 2017, pp. 232–246.

[31] P. Chaudhary, R. Gupta, A. Singh, and P. Majumder, "Analysis and comparison of various fully homomorphic encryption techniques," in *2019 International Conference on Computing, Power and Communication Technologies (GUCON)*, IEEE, 2019, pp. 58–62.

[32] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[33] N. Li, T. Zhou, X. Yang, Y. Han, W. Liu, and G. Tu, "Efficient multi-key fhe with short extended ciphertexts and directed decryption protocol," *IEEE Access*, vol. 7, pp. 56 724–56 732, 2019.

[34] C. Gentry and S. Halevi, "Implementing gentry's fully-homomorphic encryption scheme," in *Advances in Cryptology–EUROCRYPT 2011: 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings 30*, Springer, 2011, pp. 129–148.

[35] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29*, Springer, 2010, pp. 24–43.

[36] S. D. Galbraith, S. W. Gebregiyorgis, and S. Murphy, "Algorithms for the approximate common divisor problem," *LMS Journal of Computation and Mathematics*, vol. 19, no. A, pp. 58–72, 2016.

[37] A. Kamal, K. Ahmad, R. Hassan, and K. Khalim, "Ntru algorithm: Nth degree truncated polynomial ring units," in *Functional Encryption*, Springer, 2021, pp. 103–115.

[38] M. Albrecht, S. Bai, and L. Ducas, "A subfield lattice attack on overstretched ntru assumptions: Cryptanalysis of some fhe and graded encoding schemes," in *Advances in Cryptology–CRYPTO 2016: 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, Springer, 2016, pp. 153–178.

[39] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, Springer, 2013, pp. 75–92.

[40] *HElib description*, `https://github.com/homenc/HElib`, Accessed: 2023-04-17.

[41] *TFHE: Fast Fully Homomorphic Encryption Library over the Torus description*, `https://github.com/tfhe/tfhe`, Accessed: 2023-04-17.

[42] C. Marcolla, V. Sucasas, M. Manzano, R. Bassoli, F. H. Fitzek, and N. Aaraj, "Survey on fully homomorphic encryption, theory, and applications," *Proceedings of the IEEE*, vol. 110, no. 10, pp. 1572–1609, 2022.

[43] *SageMath description*, `https://www.sagemath.org/index.html`, Accessed: 2023-02-28.

[44] *NumPy description*, `https://numpy.org/doc/stable/index.html`, Accessed: 2023-02-28.

[45] *SymPy description*, `https://www.sympy.org/en/index.html`, Accessed: 2023-04-04.

[46] *Cryptographic key length recommendation description*, `https://www.keylength.com/en/compare/`, Accessed: 2023-02-28.

[47] A. K. Lenstra, "Key lengths," Wiley, Tech. Rep., 2006.