HOMOMORPHIC ENCRYPTION BASED ON THE RING LEARNING WITH ERRORS (RLWE) PROBLEM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

İREM KESKİNKURT

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
CRYPTOGRAPHY

SEPTEMBER 2017

Approval of the thesis:

HOMOMORPHIC ENCRYPTION BASED ON THE RING LEARNING WITH ERRORS (RLWE) PROBLEM

submitted by **İREM KESKİNKURT** in partial fulfillment of the requirements for the degree of **Master of Science in Department of Cryptography, Middle East Technical University** by,

Prof. Dr. Bülent Karasözen
Director, Graduate School of **Applied Mathematics** _____

Prof. Dr. Ferruh Özbudak
Head of Department, **Cryptography** _____

Assoc. Prof. Dr. Murat Cenk
Supervisor, **Cryptography, METU** _____

Examining Committee Members:

Prof. Dr. Ferruh Özbudak
Mathematics, METU _____

Assoc. Prof. Dr. Murat Cenk
Cryptography, METU _____

Assist. Prof. Dr. Oğuz Yayla
Mathematics, Hacettepe University _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: İREM KESKİNKURT

Signature :

ABSTRACT

HOMOMORPHIC ENCRYPTION BASED ON THE RING LEARNING WITH ERRORS (RLWE) PROBLEM

Keskinkurt, İrem

M.S., Department of Cryptography

Supervisor : Assoc. Prof. Dr. Murat Cenk

September 2017, 41 pages

The encryption techniques used to ensure data secrecy have been evolving in compliance with the developments in technology and reforming according to need. Nowadays, the increase in the amount of data that should be stored in encrypted form, has led to the need for encryption schemes that provide both the safety and the efficient usability of data. Homomorphic encryption, which enables the ability to make computations on encrypted data, is seen as one of the solutions that can meet this need.

In this thesis, the definitions and the properties of homomorphic encryption, some possible practical applications of homomorphic encryption, the *Ring Learning with Errors* problem and a somewhat homomorphic encryption scheme based on this problem has been examined. The computational complexity and efficiency of the algorithm have been studied by adapting some techniques in the literature to the algorithm.

Keywords: homomorphic encryption, partially homomorphic, fully homomorphic, somewhat homomorphic, relinearization

ÖZ

HALKALARDA HATALARLA ÖĞRENME (HHÖ) PROBLEMİNE DAYALI HOMOMORFİK ŞİFRELEME

Keskinkurt, İrem
Yüksek Lisans, Kriptografi Bölümü
Tez Yöneticisi : Doç. Dr. Murat Cenk

Eylül 2017, 41 sayfa

Verilerin mahremiyetini sağlamak için kullanılan şifreleme teknikleri, teknolojideki gelişmelere uygun olarak değişmekte ve ihtiyaçlara göre şekillenmektedir. Günümüzde şifrelenerek depolanması gereken verilerin miktarının artması, verilerin hem güvenliğini ve hem de verimli bir şekilde kullanılabilirliğini sağlayan şifreleme tekniklerine ihtiyaç doğmasına sebep olmuştur. Şifreli veriler üzerinde hesaplamalar yapabilme imkanı sağlayan homomorfik şifeleme, bu ihtiyacı karşılayabilecek çözümlerden biri olarak görülmektedir.

Bu tezde, homomorfik şifreleme tanımları ve temel özellikleri, homomorfik şifrelemenin olası uygulama alanları ve şekilleri, *Halkalarda Hatalarla Öğrenme* problemi ve güvenliği bu probleme dayalı olan bir sınırlı homomorfik şifreleme algoritması incelenmiştir. Literatürdeki bazı teknikler algoritmaya uyarlanarak algoritmanın hesaplama karmaşıklığı ve verimliliği üzerine çalışılmıştır.

Anahtar Kelimeler: homomorfik şifreleme, kısmi homomorfik şifreleme, sınırlı homomorfik şifreleme, tam homomorfik şifreleme, tekrar doğrusallaştırma

Dedicated to my family

ACKNOWLEDGMENTS

I would like to thank my thesis supervisor Accos. Prof. Dr. Murat Cenk for his support and guidance.

I want to thank to my parents Sabiha Keskinkurt and İlker Keskinkurt for always believing in me.

Thanks are also due to my friends for their frienship and support.

I would like to express my gratitude to my fiancee Sinan Paksoy. I am grateful to him for his patience, support and love.

This work is partially supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) under the grant no 115R289.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

DFT	Discrete Fourier Transform
FFT	Fast Fourier Transform
FHE	Fully Homomorphic Encryption
$\mathbb{N}$	Natural Numbers
PHE	Partially Homomorphic Encryption
$\mathbb{Q}$	Rational Numbers
RLWE	Ring Learning with Errors
SwHE	Somewhat Homomorphic Encryption
TMVP	Toeplitz Matrix Vector Product
$\mathbb{Z}$	Integers

CHAPTER 1

INTRODUCTION

Since long before the common era, cryptography has been the main tool for information security. Back then, people were using simple ciphers and techniques, such as shift cipher, substitution cipher and stenography [20], to conceal private information or communicate in secrecy. In time, the concept of cryptography has evolved. Today, we can define cryptography as an interdisciplinary science that deals with designing and developing secure cryptographic algorithms.

Modern cryptographic studies are being pursued in two main parts: the *symmetric key* cryptography and the *asymmetric(public)* key cryptography [28]. Symmetric key schemes use only one cryptographic key for both encryption and decryption, and they are very efficient compared to asymmetric key schemes. Public key schemes are based on hard mathematical problems, such as *integer factorization* and *discrete logarithm*, and they require two different keys one for encryption, one for decryption. In public key schemes, the key which must be kept secret is the decryption key, that we refer to as the *private key*.

In today's world people safely store or transmit their confidential data securely under favor of symmetric and public key cryptosystems. Technological developments bring about need for new cryptographic algorithms to provide more efficiency and security. As an example of this, the use of cloud storage gives rise to a need for cryptosystems that provide the ability to perform computations on encrypted data without having to decrypt it.

The search of encryption schemes that can compute *arbitrary* functions on encrypted data, namely *fully homomorphic*, started almost 40 years ago. Since then, lots of schemes have been presented but none of them can be used practically due to efficiency problems. The biggest problem is that computing arbitrary functions requires so many arithmetic operations on ciphertext space. A ciphertext space is usually a set with a large number of elements with which we can not make calculations easily. For example the scheme we examine in this thesis has a ciphertext space which consists of high degree polynomials with large coefficients. Operations on this set take lots of time and make it unpractical.

On the other hand, it is easier to use homomorphic encryption to evaluate the functions that can be computed with less arithmetic operations. The schemes that can support a

limited number of arithmetic operations are called *somewhat homomorphic*. With these schemes simple functions, such as functions that includes the sum of numbers and multiplication of a few numbers, can be computed efficiently while they are encrypted. But for more complicated functions, such as exponentiation and division, we need many homomorphic multiplications and this makes the homomorpic operations very slow. Therefore, any development on arithmetic operations would be a big contribution to this subject.

1.1 Historical Process and Literature Review

In 1978 the widely used cryptosystem *RSA* was presented as the first practical public key encryption scheme [32]. A classical *RSA* set up starts with generating two large primes p and q. The numbers $n = pq$ and $N = (p-1)(q-1)$ are calculated. A number d satisfying $gcd(d, N) = 1$ is chosen as the private key(decryption key). The public key(encryption key) e is the number satisfying $ed \equiv 1 \mod N$. The pair (n, e) is published. *RSA* encrypts a message m by computing $c = m^e \mod n$ and decrypts the ciphertext c by computing $m = c^d \mod n$. The matematical foundations of the algorithm can be found in [32].

Suppose two messages m_1 and m_2 are encrypted by *RSA* with the parameters above and ciphertexts c_1 and c_2 are obtained. Then $c_1 = m_1^e \mod n$ and $c_2 = m_2^e \mod n$. The multiplication of this ciphertexts $c_1c_2 = m_1^e m_2^e \mod n = (m_1m_2)^e \mod n$ yields the multiplication of the message m_1m_2 when decrypted. Because of this property *RSA* is said to be a *multiplicatively homomorphic* encryption scheme. Altough this seems very useful at first, *RSA* can not be used without padding as it is not semantically secure and the padded version *RSA-OAEP* (*RSA- Optimal Asymmetric Encryption Padding*) [2] does not have the homomorphic property.

Consider any cryptosystem that has a homomorphic property similar to unpadded *RSA*. If this cryptosystem uses a one-to-one function as its encryption map, it would have the same semantic security problem as *RSA* no matter what the encryption function is. That means encryption maps should not be chosen as functions in order to make homomorphic operations useable. Instead one should use randomized maps to define encryption.

After noticing RSA's homomorphic property, Ronald R.Rivest, Len Adleman and Michael L. Dertouzos started the search of homomorphic encryption schemes or as they call it "*privacy homomorphism*" [31]. Other encryption schemes that are homomorphic with respect to just one operation as *RSA* were presented. *ElGamal* cryptosystem is another example of multiplicatively homomorphic encryption schemes [12]. We can give *Goldasser-Micali* [18], *Benaloh* [3] and *Paillier* [29] cryptosystems as examples of additive homomorphic encryption schemes. These kinds of encryption schemes are called *partially homomorphic*.

The search of encryption schemes that can have homomorphic property with respect to both operations, namely *fully homomorphic*, continued with no results for almost 30 years until Dan Boneh, Eu-Jin Goh and Kobbi Nissim presented the *BGN* cryptosys-

tem in 2005 [5]. This cryptosystem is the first significant development on this subject but it was not the exact solution of the problem. This scheme could support an unlimited number of homomorphic additions, but only a single multiplication. This scheme was neither partially homomorphic nor fully homomorphic. After that discovery, encryption schemes which support a limited number of homomorphic operations were regarded as *somewhat homomorphic*.

Finally, in 2009 Craig Gentry presented the first fully homomorphic encryption scheme [17]. To be more precise, he presented a somewhat homomorphic encryption scheme and a *bootstrapping* technique that transforms this scheme into a fully homomorphic encrypiton scheme. This lattice based scheme has both homomorphic properties, that is, it can compute both addition and multiplication on encrypted data without preliminary decryption. This means any function can be evaluated on encrypted data by using circuits. Gentry's work is the first credible development for a fully homomorphic encryption scheme. To explain Gentry's brilliant scheme briefly, we should start with the size-increasing effect of homomorphic operations on ciphertexts. Every homomorphic operation, especially multiplication produces a ciphertext with larger noise. Here we mean extra loadings that a randomized encryption map add to the ciphertexts to provide security. The bootstrapping technique applies decryption process homomorphically and produces a ciphertext with smaller noise. Therefore, it makes it possible to perform as many homomorphic operations as we want, theoretically. Although some improvements were made [15, 36, 37] still there is no practical implementation of the scheme due to efficiency problems. Gentry's work has been an inspiration and since then several fully homomorphic schemes that are simpler and efficient have been presented[14, 35]. But again, non of them is efficient enough to use in practical applications yet. These studies of *FHE* can be categoriced into four main classes: lattice-based [17], over the integers [39], the (Ring) Learning with Errors Problem based [7, 8] and NTRU-like [23].

In 2011, Zvika Brakerski and Vinod Vaikuntanathan presented a fully homomorphic encryption scheme [7]. Following Craig Gentry's work, they present a somewhat homomorphic encryption scheme and obtain a fully homomorphic encryption by using a technique called relinearization which is similar to the bootstrapping technique in [17]. The scheme's security is based on the *Ring Learning with Errors(RLWE)* problem [24]. Encryption and decryption algorithms of this scheme are both defined on a finite polynomial ring and use polynomial arithmetic. Multiplication of high degree polynomials with large coefficients is the most important factor that affects the efficiency of the algorithm. As with other schemes, this one cannot be implemented efficiently to real life applications.

We presented the important milestones of the historical development of homomorphic encryption. Other studies on homomorphic encrypiton schemes can be found in [8, 16, 19, 23, 25, 27, 33, 39, 41].

1.2 Outline

In this thesis, we work on a proposed *somewhat homomorphic encryption* scheme based on the *Ring Learning with Errors* problem [7]. We mostly concentrated on developing ideas for efficient implementation of the scheme. In Chapter 2, we give some basic definitions, properties and we explain some algorithms as a background information. In Chapter 3, we give a formal definition of homomorphic encryption and mention some possible real life applications of it. In Chapter 4, we introduce the Ring Learning with Errors problem and the somewhat homomorphic encryption scheme based on this problem. In Chapter 5, we explain two methods we use to reduce the computational complexity of the encryption scheme and make a comparison between these methods. Finally, we state our conclusion in Chapter 6, and share some ideas about future studies on homomorphic encryption schemes.

CHAPTER 2

BACKGROUND

In this chapter, we give some background information for a better understanding on the subjects that will come up later on. In Chapter 3, we explain the construction of the functions that can be evaluated homomorphically and we use *arithmetic circuits* to do that. In Chapter 4, a proposed somewhat homomorphic encryption scheme is examined with the focus of efficient implementation. To develop techniques for efficient implementation, we use n-th *cyclotomic polynomial*, which we define in this chapter. We also use *Fast Fourier Transform* and *Teoplitz Matrix Vector Product* in Chapter 5 to speed up polynomial multiplication in the finite polynomial ring of the encryption scheme. We use the *Big-Oh*$(\mathcal{O})$ notation to compare the computational complexity of the algorithms.

2.1 Definitions and Properties

As we will see in Chapter 3, we need to construct *arithmetic circuits* to evaluate functions homomorphically. A simple definition and some basic properties would be sufficient for our purpose.

Definition 2.1. *(Arithmetic circuits)*[34]

Let R be a ring and $\mathcal{X}$ be a set of variables. An arithmetic circuit $\mathscr{C}$ over $\mathcal{X}$ and R, is a directed acyclic graph, whose vertices are called gates and which satisfies the following properties:

i. every input gate of $\mathscr{C}$ (in-degree 0 gate) is either labeled by a variable in $\mathcal{X}$ or an element in R,

ii. every other gate of $\mathscr{C}$ (in-degree 2 gate) is labeled by either one of the arithmetic operations of R.

An arithmetic circuit is the standard model of computing polynomials by using the operations $+$ and $\times$ with variables $x_1, x_2, \ldots, x_n \in \mathcal{X}$ and coefficients from R. The output of an arithmetic circuit is simply a polyomial in $R[x_1, x_2, \ldots, x_n]$. The number of edges in $\mathscr{C}$ is called the *size* of $\mathscr{C}$ and the length of the longest path in $\mathscr{C}$ is called

the *depth* of $\mathscr{C}$. The depth of an arithmetic circuit is our main concern because it is directly related to the number of homomorphic operations that a somewhat homomorphic encryption scheme supports.

Before the definition of cyclotomic polynomial, we need to explain what a primitive n-th root of unity is. For a positive integer n, the number ζ is called a *n-th root of unity* if it satisfies the equation $\zeta^n = 1$. The smallest integer k satisfying $\zeta^k = 1$ is called the *order* of ζ. If a n-th root of unity has order n then, it is called the *primitive n-th root of unity*.

Definition 2.2. *(Cyclotomic Polynomial)*

The monic polynomial $\Phi_n(x) \in \mathbb{Z}[x]$ having the primitive n-th roots of unity as its roots is called the *n-th cyclotomic polynomial.*

A primitive n-th root of unity is a generator of the set of all n-th roots of unity. More precisely if ζ is primitive a n-th root unity, then the set S of all n-th roots of unity is

$$S = \langle \zeta \rangle = \{1, \zeta, \zeta^2, \ldots, \zeta^{n-1}\}.$$

If $gcd(k, n) = 1$ for the integer $k < n$ then, ζ^k is also a primitive n-th root of unity. Therefore, there are exactly $\varphi(n)$ primitive n-th roots of unity, where φ is the *Euler's totient* function.

Proposition 2.1. *(Some Properties of Cyclotomic Polynomial)*

1. *Since the roots of the n-th cyclotomic polynomial $\Phi_n(x)$ are the n-th roots of unity, the following equalities hold:*

$$\Phi_n(x) = \prod_{\substack{i=0,\ldots,\varphi(n) \\ \zeta_i^n = 1 \\ ord(\zeta_i) = n}} (x - \zeta_i) = \prod_{\substack{1 \le k < n \\ gcd(k,n) = 1}} (x - \zeta^k).$$

2. *Since there are $\varphi(n)$ primitive n-th roots of unity, the degree of $\Phi_n(x)$ is $\varphi(n)$ where $\varphi(n)$ denotes the Euler's totient function of n.*

3. *Let $\mathbb{F}_{p^m} = \mathbb{F}_q$ be a finite field such that p is a prime and m is a positive integer. $\Phi_n(x)$ is defined over $\mathbb{F}_q$ if and only if n is not divisible by p.*

4. *Let k be a positive integer and $n = 2^k$. Then, the n-th cyclotomic polynomial $\Phi_n(x)$ can be determined easily as follows:*

$$\Phi_n(x) = x^{n/2} + 1 = x^{2^{k-1}} + 1.$$

5. *The n-th cyclotomic polynomial $\Phi_n(x)$ is irreducible over $\mathbb{Q}$ for every positive integer n, but not all are irreducible when considered as a polynomial over a finite field.*

The following theorem gives more detailed information about the factorization of the n-th cyclotomic polynomial $\Phi_n(x)$ over the finite fields for which $\Phi_n(x)$ exists. We use this fact to apply *Fast Fourier Transform*, which we explain next, to polynomial multiplication algorithm in the ring of the encryption scheme we study.

Theorem 2.2 ([22, p. 65])**.** *Let $\mathbb{F}_{p^m} = \mathbb{F}_q$ be a finite field where p is a prime number and m is a positive integer. Suppose the positive integer n is not divisible by p. Then, the $n-th$ cyclotomic polynomial $\Phi_n(x)$ factors into $\varphi(n)/d$ distinct irreducible polynomials of the same degree d over $\mathbb{F}_q$ where d is the order of q modulo n, that is d is the smallest positive integer satisfying $q^d \equiv 1 \mod n$.*

Now we explain the algorithms of *Fast Fourier Transform* and *Toeplitz Matrix Vector Product* and we need a measurement to analyze and compare the performance of the algorithms. Implementing and running the algorithms with several inputs to measure their performances according to their running times is not easy and requires lots of time. In *Computational Complexity Theory*, instead of this approach the technique called *asymptotic analysis* is used [40]. The idea is evaluating an algorithm's computational complexity in terms of the input size and express it asymptotically by using three notations Θ, $\mathcal{O}$(Big-Oh) and Ω. We use $\mathcal{O}$ notation to analyze the algorithms.

Definition 2.3. Let $M(n)$ be the function defining the computational complexity of the algorithm $\mathscr{A}$ with input size n. If there exists a function $f(n)$, a real number k and a positive integer b such that $M(n) \leq kf(n)$ for all $n \geq b$ then, we say the algorithm $\mathscr{A}$ runs in time $\mathcal{O}(f(n))$.

2.2 Discrete Fourier Transform

In this section we explain the *Discrete Fourier Transform(DFT)* and the *Inverse Discrete Fourier Transfrorm(IDFT or DFT^{-1})*. Before that we need to mention two ways of representing a polynomial which are essential for understanding *DFT* and *IDFT*. The polynomials that have degree less than a positive integer n are referred as the polynomials with degree bound n.

Definition 2.4. *(Coefficient representation of a polynomial)*

Let $a(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1}$ be a polynomial with degree bound n. The coefficient representation of the polynomial $a(x)$ is the vector A such that

$$A = (a_0, a_1, \ldots, a_{n-1}).$$

The coefficient representation of a polynomial is unique. Obtaining a polynomial from its coefficient representation is obvious and it does not require any arithmetic operations.

Definition 2.5. *(Point value representation of a polynomial)*

Let $a(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1}$ be a polynomial with degree bound n. A point value representation of the polynomial $a(x)$ is the set of n pairs

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\},$$

where x_i are distinct and $y_i = a(x_i)$ for $i = 0, \ldots, n-1$.

Unlike the coefficient representation, there are many different point value representations of a polynomial. Computing a point value representation of a polynomial requires n point evaluations which can be done in $\mathcal{O}(n^2)$ by using *Horner's method* [10]. Given a point value representation, obtaining a polynomial that fits this representation is called *polynomial interpolation*. The following theorem guarantees the existence and the uniqueness of a polynomial when a point value representation is given.

Theorem 2.3 ([38, p. 38])**.** *For given n pairs $\{(x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$ such that x_i are distinct for $i = 0, \ldots, n-1$, there exists a unique polynomial $a(x) \in \mathscr{P}_n$ with $a(x_i) = y_i$ for $i = 0, \ldots, n-1$, where $\mathscr{P}_n$ is the set of all real or complex polynomials with a degree bound n.*

There are several methods in the literature for interpolating a polynomial. Further detailed information about the methods can be found in [21]. Now we are ready to give the definition of *Discrete Fourier Transform*.

Definition 2.6. *(Discrete Fourier Transform)*

Let $A = (a_0, a_1, \ldots, a_{n-1})$ be the coefficient representation of a polynomial $a(x)$ and $S = \{\zeta_0, \zeta_1, \ldots, \zeta_{n-1}\}$ be the set of all n-th roots of unity. The vector

$$y = (y_0, y_1, \ldots, y_{n-1})$$

such that $y_i = a(\zeta_i)$ for $i = 0, \ldots, n-1$ is called the *Discrete Fourier Transform* of the coefficient vector $A = (a_0, a_1, \ldots, a_{n-1})$ and it is denoted by $DFT_n(A) = y$.

In short, we can define *DFT* as computing a point value representation of a polynomial at the n-th roots of unity. By Theorem 2.3 we know that we can obtain the polynomial when the evaluations at n-th roots of unity are given. This process is called the *Inverse Discrete Fourier Transform* and we denote it by $IDFT_n(y) = A$ or $DFT_n^{-1}(y) = A$ where n, y, A are defined as in definition 2.6.

In the next section we see how to go back and forth between the coefficient representation and the point value representation of a polynomial in $\mathcal{O}(n \log n)$ by taking advantage of the properties of roots of unity. Note that we mean base two logarithm of n by $\log n$. We use some properties of the special matrix called *Vandermonde Matrix* to compute DFT^{-1} efficiently.

Definition 2.7. [30] A $n \times n$ matrix of the form

$$V = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ 1 & x_3 & x_3^2 & \dots & x_3^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} \end{pmatrix},$$

which is determined by arbitrary and distinct n elements $x_1, x_2, \dots, x_n$ is called a *Vandermonde matrix* of order n . We use $V(x_1, \dots, x_n)$ notation to denote the matrix above for simplicity. The determinant of a *Vandermonde matrix* $V(x_1, \dots, x_n)$ is defined by

$$det(V(x_1, \dots, x_n)) = \prod_{1 \le i < j \le n} (x_i - x_j).$$

2.3 Fast Fourier Transform(FFT)

In this section, we explain how *Fast Fourier Transform(FFT)* algorithm works and how it computes DFT_n and DFT_n^{-1} in $\mathcal{O}(n \log n)$ by benefiting from the properties of n-th roots of unity. Let

$$a(x) = a_0 + a_1 x + a_2 x^2 + a_2 x^3 + \dots + a_{n-1} x^{n-1}$$

be a polynomial with degree bound n. For simplicity we take n to be a power of 2, say $n = 2^k$ for a positive integer k. We can always add zero coefficients (of higher order) to provide this condition. Let us define two polynomials

$$a_{even}(x) = a_0 + a_2 x + a_4 x^2 + \dots + a_{n-2} x^{n/2-1},$$
$$a_{odd}(x) = a_1 + a_3 x + a_5 x^2 + \dots + a_{n-1} x^{n/2-1}.$$

Then, we can easily write $a(x)$ and $a(-x)$ in terms of these polynomials as follows:

$$a(x) = a_{even}(x^2) + x a_{odd}(x^2) \quad \text{and} \quad a(-x) = a_{even}(x^2) - x a_{odd}(x^2).$$

Therefore, once we compute $a_{even}(x^2)$ and $x a_{odd}(x^2)$ for $a(x)$, just an extra addition of half size polynomials would be enough to compute $a(-x)$. We use this fact to calculate $DFT_n(A)$ efficiently where A is the coefficient representation of the polynomial $a(x)$. Let ζ be a primitive n-th root of unity, then we may denote the set of all n-th roots of unity by $S_0 = \langle \zeta \rangle = \{1, \zeta, \zeta^2, \dots, \zeta^{n-1}\}$. We want to evaluate the polynomial $a(x)$ at

the points in S_0. Using the property above we have

$$\begin{aligned}
a(1) &= a_{even}(1) + a_{odd}(1),\\
a(\zeta) &= a_{even}(\zeta^2) + \zeta a_{odd}(\zeta^2),\\
&\vdots\\
a(\zeta^{n/2-1}) &= a_{even}(\zeta^{n-2}) + \zeta^{n/2-1} a_{odd}(\zeta^{n-2}),\\
a(\zeta^{n/2}) &= a_{even}(\zeta^{n}) + \zeta^{n/2} a_{odd}(\zeta^{n}),\\
a(\zeta^{n/2+1}) &= a_{even}(\zeta^{n+2}) + \zeta^{n/2+1} a_{odd}(\zeta^{n+2}),\\
&\vdots\\
a(\zeta^{n-1}) &= a_{even}(\zeta^{2n-2}) + \zeta^{n-1} a_{odd}(\zeta^{2n-2}).
\end{aligned}$$

Since ζ is a ptimitive n-th root of unity we know that $\zeta^n = 1$ and $\zeta^{n/2} = -1$. If we replace $\zeta^{n/2}$ by -1 in the equations above, we get

$$\begin{aligned}
a(1) &= a_{even}(1) + a_{odd}(1),\\
a(\zeta) &= a_{even}(\zeta^2) + \zeta a_{odd}(\zeta^2),\\
&\vdots\\
a(\zeta^{n/2-1}) &= a_{even}(\zeta^{n-2}) + \zeta^{n/2-1} a_{odd}(\zeta^{n-2}),\\
a(-1) &= a_{even}(1) - a_{odd}(1),\\
a(-\zeta) &= a_{even}(\zeta^2) - \zeta a_{odd}(\zeta^2),\\
&\vdots\\
a(-\zeta^{n/2-1}) &= a_{even}(\zeta^{n-2}) - \zeta^{n/2-1} a_{odd}(\zeta^{n-2}).
\end{aligned}$$

The set $S_1 = \{1, \zeta^2, \zeta^4, \ldots, \zeta^{n-2}\}$ is generated by ζ^2 and it can be easily seen that ζ^2 is a primitive $n/2$-th root of unity. So the set $S_1 = \langle \zeta^2 \rangle$ is the set of all $n/2$-th roots of unity. Therefore, the problem is reduced to evaluating two polynomials $a_{even}(x)$ and $a_{odd}(x)$ with degree bound $n/2$ at $n/2$-th roots of unity, that is we have two half size problems. We need $n/2$ multiplications and n additions to combine them. If we denote the number of additions by $\mathcal{S}(n)$ and the number of multiplications by $\mathcal{M}(n)$ which are needed to solve the size n problem then, clearly

$$\mathcal{S}(n) = 2\mathcal{S}(n/2) + n \quad \text{and} \quad \mathcal{M}(n) = 2\mathcal{M}(n/2) + n/2.$$

Moreover, if we define $S_i = \langle \zeta^{2^i} \rangle$ for $i = 0, \ldots, k$ the set S_i is the set of all $n/2^i$-th roots of unity since ζ^{2^i} is a primitive $n/2^i$-th root of unity from the fact that

$$(\zeta^{2^i})^{n/2^i} = 1 \quad \text{and} \quad (\zeta^{2^i})^{n/2^{i+1}} = -1,$$

for all $i = 0, \ldots, k$. This means that we can reduce all size $n/2^i$ problems to two size $n/2^{i+1}$ problems and it allows us to solve the original problem by solving half size

problems recursively. Then, the complexity can be written as follows:

$$\begin{aligned}
\mathcal{S}(n) &= 2\mathcal{S}(n/2) + n, & \mathcal{M}(n) &= 2\mathcal{M}(n/2) + n/2,\\
\mathcal{S}(n) &= 2^2\mathcal{S}(n/2^2) + 2n, & \mathcal{M}(n) &= 2^2\mathcal{M}(n/2^2) + 2n/2,\\
&\vdots & &\vdots\\
\mathcal{S}(n) &= 2^k\mathcal{S}(n/2^k) + kn, & \mathcal{M}(n) &= 2^k\mathcal{M}(n/2^k) + kn/2.
\end{aligned}$$

Recall that we chose $n = 2^k$ and therefore, we have $k = \log n$. It is easy to see that $\mathcal{S}(1) = \mathcal{M}(1) = 0$ since evaluating a polynomial with degree bound 1 (i.e. a constant polynomial) at a point requires no algebraic operations. If we substitute these values in the last line above we get

$$\mathcal{S}(n) = n \log n \quad \text{and} \quad \mathcal{M}(n) = n/2 \log n.$$

Hence, ***FFT*** requires

$$3n/2 \log n \in \mathcal{O}(n \log n)$$

ring operations to evaluate $\textbf{\textit{DFT}}_n(A)$ where A is the coefficient representation of the polynomial $a(x)$.

Now we show how to go back to the coefficient representation from the point value representation $\textbf{\textit{DFT}}_n(A)$. Suppose we are given the values b_i such that $a(\zeta^i) = b_i$ for $i = 0, \ldots, n-1$ where ζ is a primitive root of unity, that is $\textbf{\textit{DFT}}_n(A) = B$ where $A = (a_0, \ldots, a_{n-1})$ and $B = (b_0, \ldots, b_{n-1})$. We use the fact that, evaluating the polynomial

$$a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1}x^{n-1}$$

at the points $\{1, \zeta, \zeta^2, \ldots, \zeta^{n-1}\}$ is equivalent to compute the following matrix vector product:

$$\underbrace{\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{pmatrix}}_{B} = \underbrace{\begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \zeta & \zeta^2 & \ldots & \zeta^{n-1} \\ 1 & \zeta^2 & \zeta^4 & \ldots & \zeta^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \zeta^{n-1} & \zeta^{2(n-1)} & \ldots & \zeta^{(n-1)(n-1)} \end{pmatrix}}_{M} \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}}_{A},$$

where the $n \times n$ matrix M is an order n Vandermonde matrix determined by the elements $1, \zeta, \zeta^2, \ldots, \zeta^{n-1}$, that is $M = V(1, \zeta, \zeta^2, \ldots, \zeta^{n-1})$ as in in definition 2.7. When we want to go back from point value representation to the coefficient representation of $a(x)$, we just have to multiply both sides of the equation with the inverse of M. Then, we have

$$B = MA \implies M^{-1}B = M^{-1}MA \implies M^{-1}B = A.$$

The existence of M^{-1} is obvious since the elements $1, \zeta, \zeta^2, \ldots, \zeta^{n-1}$ are all distinct, $det(M) \neq 0$ by definition 2.7. Thanks to the specialty of M, computing it inverse

is very easy and does not require any arithmetic operation unlike the general matrix inversion which is done in $\mathcal{O}(n^3)$ time using Gaussian elimination [10]. The inverse of M is equal to

$$M^{-1} = \frac{1}{n} V(1, \zeta^{-1}, \zeta^{-2}, \ldots, \zeta^{-(n-1)}).$$

Let $\zeta^{-1} = \zeta^{n-1} = \xi$. Since $gcd(n, n-1) = 1$ we know that ξ is also a primitive n-th root of unity. Then, multiplying B by $M^{-1} = \frac{1}{n} V(1, \xi, \xi^2, \ldots, \xi^{n-1})$ is equivalent to evaluate the polynomial $b(x) = b_0 + b_1 x + \cdots + b_{n-1} x^{n-1}$ at n-th roots of unity and to divide each component of the result by n. This means that we can compute $\textit{DFT}_n^{-1}(B) = A$ by using a modified version of *FFT* which we explained above. For this we need $3n/2 \log n + n \in \mathcal{O}(n \log n)$ ring operations.

Hence, we can transform the polynomial $a(x)$ back and forth between its point value and coefficient representations with $3n \log n + n$ ring operations which is equivalent to $\mathcal{O}(n \log n)$ asymptotically.

2.4 Toeplitz Matrix Vector Product

Toeplitz matrix vector product(TMVP) arises in many cryptographic applications. For example, the use of it in binary extension fields can be seen in [13]. We use *TMVP* for polynomial multiplication in the finite polynomial ring $\mathbb{Z}_q[x]/\langle x^n + 1 \rangle$ in Chapter 5 and give the computational complexity by similar techniques in [13].

Definition 2.8. A $n \times n$ Toeplitz matrix T has a specific type of the form:

$$T = \left(\begin{array}{ccccc|ccccc} a_0 & a'_1 & a'_2 & \ldots & a'_{n/2-1} & a'_{n/2} & \ldots & a'_{n-3} & a'_{n-2} & a'_{n-1} \\ a_1 & a_0 & a'_1 & \ldots & a'_{n/2-2} & a'_{n/2-1} & \ldots & a'_{n-4} & a'_{n-3} & a'_{n-2} \\ a_2 & a_1 & a_0 & \ldots & a'_{n/2-3} & a'_{n/2-2} & \ldots & a'_{n-5} & a'_{n-4} & a'_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ a_{n/2-1} & a_{n/2-2} & a_{n/2-3} & \ldots & a_0 & a'_1 & \ldots & a'_{n/2-2} & a'_{n/2-1} & a'_{n/2} \\ \hline a_{n/2} & a_{n/2-1} & a_{n/2-2} & \ldots & a_1 & a_0 & \ldots & a'_{n/2-3} & a'_{n/2-2} & a'_{n/2-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ a_{n-3} & a_{n-4} & a_{n-5} & \ldots & a_{n/2-2} & a_{n/2-3} & \ldots & a_0 & a'_1 & a'_2 \\ a_{n-2} & a_{n-3} & a_{n-4} & \ldots & a_{n/2-1} & a_{n/2-2} & \ldots & a_1 & a_0 & a'_1 \\ a_{n-1} & a_{n-2} & a_{n-3} & \ldots & a_{n/2} & a_{n/2-1} & \ldots & a_2 & a_1 & a_0 \end{array}\right),$$

which can be represented by the partitioned matrix

$$T = \left(\begin{array}{c|c} T_0 & T_1 \\ \hline T_2 & T_0 \end{array}\right),$$

where the partitions T_i are also $\frac{n}{2} \times \frac{n}{2}$ Toeplitz matrices.

The sum of two $n \times n$ Toeplitz matrices requires $2n-1$ additions when general $n \times n$ matrices need n^2. Another useful property is, the sum of two *Toeplitz* matrices is again a *Toeplitz* matrix. This property allows us to make recursive calculations. Now we show the method of multiplying a $n \times n$ Toeplitz matrix by a $n \times 1$ vector in $\mathcal{O}(n^{\log 3})$ which is better than the schoolbook method's complexity $\mathcal{O}(n^2)$. Let us denote B as a $n \times 1$ vector with partitions B_0 and B_1 are $n/2 \times 1$ vectors.

$$B = \begin{pmatrix} b_0 \\ \vdots \\ b_{n/2-1} \\ \hline b_{n/2} \\ \vdots \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} B_0 \\ B_1 \end{pmatrix}.$$

The product of a $n \times n$ Toeplitz matrix T and a $n \times 1$ vector B can be calculated as follows:

$$\begin{aligned} T &= \begin{pmatrix} T_0 & T_1 \\ T_2 & T_0 \end{pmatrix} \begin{pmatrix} B_0 \\ B_1 \end{pmatrix} \\ &= \left[\begin{pmatrix} T_0 & T_0 \\ T_0 & T_0 \end{pmatrix} + \begin{pmatrix} 0 & T_1 - T_0 \\ T_2 - T_0 & 0 \end{pmatrix} \right] \begin{pmatrix} B_0 \\ B_1 \end{pmatrix} \\ &= \begin{pmatrix} T_0 & T_0 \\ T_0 & T_0 \end{pmatrix} \begin{pmatrix} B_0 \\ B_1 \end{pmatrix} + \begin{pmatrix} 0 & T_1 - T_0 \\ T_2 - T_0 & 0 \end{pmatrix} \begin{pmatrix} B_0 \\ B_1 \end{pmatrix} \\ &= \begin{pmatrix} T_0(B_0 + B_1) \\ T_0(B_0 + B_1) \end{pmatrix} + \begin{pmatrix} (T_1 - T_0)B_1 \\ (T_2 - T_0)B_0 \end{pmatrix} \\ &= \begin{pmatrix} P_1 + P_2 \\ P_1 + P_3 \end{pmatrix}, \end{aligned}$$

where $\quad P_1 = T_0(B_0 + B_1), \quad P_2 = (T_1 - T_0)B_1, \quad P_3 = (T_2 - T_0)B_0.$

Therefore, the product reduces to 3 Toeplitz matrix vector products, 2 Toeplitz matrix additions and 3 vector additions, all of which have half size. Now lets take a closer look to $T_1 - T_0$ and $T_2 - T_0$. The last $n/2-1$ elements in the first column of $T_1 - T_0$ are the negatives of the first $n/2-1$ elements in the last column of $T_2 - T_0$, in a different order. That means we can reuse this elements for $T_2 - T_0$ after calculating $T_1 - T_0$ by negating them. Considering negation is a free operation, this trick reduces the number of additions to $3n/2 - 1$ from $2n - 2$. Let us denote the number of additions by $\mathcal{S}(n)$ and the number of multiplications by $\mathcal{M}(n)$ which are needed to solve the size n problem.

Then, the number of additions required is

$$\begin{aligned}
\mathcal{S}(n) &= 3\mathcal{S}(n/2) + 3n - 1, \\
\mathcal{S}(n) &= 3^2\mathcal{S}(n/2^2) + 3n + \frac{3}{2}3n - 1 - 3, \\
&\vdots \\
\mathcal{S}(n) &= 3^k\mathcal{S}(n/2^k) + 3n\left(1 + \cdots + \frac{3^{k-1}}{2^{k-1}}\right) - (1 + \cdots + 3^{k-1}), \\
\mathcal{S}(n) &= 3^k\mathcal{S}(n/2^k) + 6n\frac{3^k - 2^k}{2^k} - \frac{3^k}{2} + \frac{1}{2},
\end{aligned}$$

and the number of multiplications required is

$$\begin{aligned}
\mathcal{M}(n) &= 3\mathcal{M}(n/2), \\
\mathcal{M}(n) &= 3^2\mathcal{M}(n/2^2), \\
&\vdots \\
\mathcal{M}(n) &= 3^k\mathcal{M}(n/2^k).
\end{aligned}$$

For the value $n = 2^k$ we have $\mathcal{S}(n) = 5.5n^{\log 3} - 6n + 1/2$ and $\mathcal{M}(n) = n^{\log 3}$. Hence, multiplying a $n \times n$ *Toeplitz matrix* by a $n \times 1$ vector requires

$$6.5n^{\log 3} - 6n + 1/2 \in \mathcal{O}(n^{\log 3}) \approx \mathcal{O}(n^{1.58})$$

ring operations in total.

CHAPTER 3

HOMOMORPHIC ENCRYPTION AND ITS PRACTICAL APPLICATIONS

This chapter contains more theoretical information about homomorphic encryption schemes and practical applications of them. We use some of the definitions and properties from the previous chapter.

3.1 Homomorphic Encryption

In this section we give more formal yet simplified definitions and properties of homomorphic encryption.

Definition 3.1. Let $\mathcal{E}$ be encryption map and $\mathcal{D}$ be decryption map of a cryptosystem, and let (k_e, k_d) is the private-public key pair. Suppose that $\mathcal{E}(m_i, k_e) = c_i$ and $\mathcal{D}(c_i, k_d) = m_i, i = 1, 2 \dots$ for all m_i in plaintext space and c_i in ciphertext space.

If there exists an operation $\star$ defined on ciphertext space satisfying

$$\mathcal{D}(c_1 \star c_2, k_d) = \mathcal{D}(c_1, k_d) + \mathcal{D}(c_2, k_d) = m_1 + m_2, \tag{3.1}$$

then the encryption scheme is said to be *additively homomorphic*.

If there exists an operation $\diamond$ defined on ciphertext space satisfying

$$\mathcal{D}(c_1 \diamond c_2, k_d) = \mathcal{D}(c_1, k_d).\mathcal{D}(c_2, k_d) = m_1.m_2, \tag{3.2}$$

then the encryption scheme is said to be *multiplicatively homomorphic*.

Encryption schemes which are homomorphic with respect to a single operation, in other words encryption schemes satisfying either one of the properties above are called *Partially Homomorphic Encryption(PHE)*. As we mention in 1.1, *RSA*[31], *ElGamal* [12] and *Goldwasser-Micali* [18] cryptosystems are some of the examples of *PHE* schemes.

Encryption schemes that have homomorphic properties with respect to both algebraic operations are called *Fully Homomorphic Encryption(FHE)*. The first *FHE* scheme

was presented in 2009 by Craig Gentry [17]. Following that, other *FHE* schemes has been presented [6, 14, 16, 35, 39] but non of the proposed schemes are practical enough to use in real life yet.

Encryption schemes which support a limited number of homomorphic operations are regarded as *Somewhat Homomorphic Encryption(SwHE)*. Since *SwHE* schemes are seen more likely to be practical, studies are being pursued mostly on them. Another reason to prefer working on *SwHE* schemes is Craig Gentry's technique to obtain *FHE* from a *SwHE*. Some of the studies about *SwHE* can be found in [6, 7, 33, 41].

Homomorphic encryption is a very interesting area of cryptography and it is very open to developments. Researches on homomorphic encryption have been going on for almost 40 years and still there is a long way to go. With a homomorphic encryption scheme we can perform operations on ciphertexts which give the sum or product of corresponding plaintexts when decrypted. If we could have an efficient FHE scheme, we would be able to compute arbitrary functions on encrypted data homomorphically, by using arithmetic circuits. In the next section we give some examples of practical uses of SwHE schemes and illustrate some possible applications of FHE schemes under the assumption that there exists one which is efficient enough to use.

3.2 Practical Applications

In this section, we give some example functions which can be computed on encrypted data homomorphically by using a SwHE sheme with depth D. By D we refer the multiplicative depth of the aritmetic circuit of the function to be computed. We use the same notations in the previous section.

Example 3.1. It is possible to construct a system with a *SwHE* to compute the mean of k numbers homomorphically. Suppose we have k numbers $z_1, \ldots, z_k$ which we store in the cloud encrypted. Assume $\mathcal{E}(z_i, k_e) = z_i'$, $i = 1, \ldots, k$. The mean of $z_1, \ldots, z_k$ is defined by

$$\mu = \frac{\sum_{i=1}^{k} z_i}{k}.$$

We can ask the cloud to compute $z' = z_1' \star \cdots \star z_k'$. The cloud does this with $k-1$ homomorphic addition operations and returns us the pair (z', k) of a ciphertext and the amount of the numbers involve this computation. When we decrypt the ciphertext we get the sum of the k numbers.

$$\begin{aligned}\mathcal{D}(z') = \mathcal{D}(z_1' \star \cdots \star z_k', k_d) &= \mathcal{D}(z_1', k_d) + \cdots + \mathcal{D}(z_k', k_d) \\ &= z_1 + \cdots + z_k = \mu k.\end{aligned}$$

Then, dividing it by k gives us the mean of the numbers.

If we had a *FHE* scheme then division can be performed homomorphically too.

Example 3.2. Let σ be the standard deviation of the terms in $\{z_1, z_2, \ldots, z_k\}$ in the

previous example. Then,

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(\mu - z_i)^2}{k}}.$$

The variance of these terms is defined by σ^2. Since we can not divide numbers efficiently on encrypted data, we can arrange the terms in this formula to be able to compute homomorphically.

$$\begin{aligned}\sigma &= \sqrt{\frac{\sum_{i=1}^{n}(\mu - z_i)^2}{k}} \\ &= \sqrt{\frac{\sum_{i=1}^{n}(\frac{\sum_{i=1}^{k} z_i}{k} - z_i)^2}{k}} \\ &= \sqrt{\frac{\sum_{i=1}^{n}(\mu k - k z_i)^2}{k^3}}.\end{aligned}$$

Then, the system can compute

$$z'' = \left[((z' \star (-kz'_1)) \diamond ((z' \star (-kz'_1))\right] \star \cdots \star \left[((z' \star (-kz'_k)) \diamond ((z' \star (-kz'_k))\right],$$

where z' and z'_i as in the previous example. Note that we use $-kz'_i$ notation for a ciphertext which decrypts $-kz_i$. Normally it can be found by $\mathcal{E}(-k) \diamond \mathcal{E}(z_i)$, but it is possible to find a trick to define multiplication by scalars homomorphically depending on the scheme we use, as we see in example 4.2. For this reason we abuse the notation for a simple explanation.

Decryption of z'' gives us $\sum_{i=1}^{k}(\mu k - k z_i)^2$. Therefore if we set the system to output the pair (z'', k) or (z'', k^3) we can compute the variance by division after decryption.

Example 3.3. Suppose for a prime integer q, we need to check whether two integers in $\mathbb{Z}_q$ are equal or not, while they are encrypted. To do this we can define a function EQ such that for two integers x and y

$$EQ(x, y) = \begin{cases} 1 \text{ if } x = y \\ 0 \text{ if } x \neq y. \end{cases}$$

We can define EQ function as $EQ(x, y) = 1 - (x - y)^{q-1} \mod q$ by using *Fermat's little* theorem. This function checks equality on encrypted data with a *SwHE* scheme of depth $\lceil \log q \rceil$, which would not be a desirable for large q values. In example 4.2 we give a more detailed explanation about this function and the required depth of the *SwHE*.

Example 3.4. It is possible to compare two integers by defining a function $COMP$ such that for two numbers x and y in $\mathbb{Z}_q$ for a prime q

$$COMP(x, y) = \begin{cases} 1 \text{ if } x < y \\ 0 \text{ if } x \geq y. \end{cases}$$

The construction of an arithmetic circuit that computes the function above is given in [9] with details. We omit the details and just share the requirements of the scheme. This function can be computed on bitwise encrypted data with a *SwHE* scheme with depth approximately $\log(\log q)$.

We gave some examples of the functions to be computed on encrypted data. The parameters of the system affects efficiency, security and the number of homomorphic operations(especially for multiplication) that the system supports. So the parameters must be readjusted as the function changes, according to the SwHE scheme used.

Now we present some made up scenarios to explain the potential use of fully homomorphic encryption in applications. For the first scenario, suppose a clinic wants to pursue a genomic research. The clinic needs the genomic data of lots of people. Since collecting, storing and processing this many data requires an expensive infrastructure and maintenance the clinique decides to use a cloud service. The cloud lets test subjects upload their genomic data to the cloud and the clinic could use them when they need to. But even if only the researchers have access to this sensitive data, it is not private and secure. Who can access this plain data could recover biological characteristics of an individual or of a community with this information.

Possible threats can be avoided by using homomorphic encryption. A private key-public key pair is generated. The private key is shared with people who are authorized to use it, in this case the researchers. The test subjects upload their genomic data after encrypting it with the public key. A researcher sends a query to the cloud and the cloud makes the requested computations on encrypted data. Since the cloud does not have the private key it returns the encrypted result to the researcher. Then the researcher decrypts it to obtain the result of the query. If trusting the researchers is not an option then some control mechanisms can be integrated into the system. For example, this could be a mechanism which does not return values when the query has very few results.

The studies for this scenario were started a few years ago. The Secure Genome Analysis Competition has taken place every year since 2015. They publish a list of challenges consisting of tasks to complete. For example in the 2015 competition, the task of *secure comparison between genomic data* was given under the challenge of *homomorphic encryption based secure genomic data analysis*. The competitors were given two genomic datasets of two individuals and required to develop a cryptographic protocol using a homomorphic encryption scheme which can compute the hamming distance or the edit distance between these two datasets while they are encrypted.

Our second scenario consists of two companies A and B. Suppose company B finds formulas to analyze financial data and compute the risks and rewards of a business to a high degree of accuracy and it wants to keep these formulas secret. On the other hand, company A needs this financial analysis to make a decision for an investment, without revealing its financial data. The solution is that company B offers a *FHE* scheme to A with the option to choose its own private key. Then A decrypts its data with the public key and sends it to B. Since the encryption scheme is constructed by B, B knows how to apply its secret formulas homomorphically. After the computations B sends

the resulting ciphertexts to A without knowing the actual results. Only A can see the actual results by decrypting the encrypted results with its private key. Now A can make decisions based on this analysis and B can make money from its discovery. Everything works well and neither of the companies need reveal their secret information.

This scenario can be extended to a version where company A uploads its data to the cloud and company B describes its secret formulas to the cloud by encrypting them. The cloud applies the encrypted functions to encrypted data and returns the encrypted results to A.

We think that even the idea of being able to use *FHE* is very exciting and satisfactory. As we see in presented scenarios, it provides a different kind of privacy, security and usability to data, that none of the existing cryptosystems could provide.

CHAPTER 4

A SOMEWHAT HOMOMORPHIC ENCRYPTION SCHEME (SwHE) BASED ON THE RING LEARNING WITH ERRORS (RLWE) PROBLEM

4.1 The Ring Learning with Errors(RLWE) Problem

The *Ring Learning with Errors (RLWE)* problem was presented in 2010 by Vadim Lyubashevsky, Chris Peikert and Oded Regev [24]. The *RLWE* problem is seen as a powerful candidate for public key cryptography in the future because it is believed that even with quantum computers solving *RLWE* is hard. In this section we give a simplified definition of *RLWE* problem. Before the definition of the problem we need to construct a background for it.

Let q be a prime number and n be a positive integer. Define the finite field $\mathbb{F}_q = \mathbb{Z}_q$ and the polynomial ring $\mathbb{Z}_q[x]$ over $\mathbb{Z}_q$. Suppose $\phi(x) \in \mathbb{Z}_q[x]$ is irreducible over $\mathbb{Q}$, hence over $\mathbb{Z}$. The finite quotient ring $R_q = \mathbb{Z}_q[x]/\langle\phi(x)\rangle$ is a subring of $\mathbb{Z}_q[x]$ and it consists of integer polynomials of degree less than n and coefficients from $\mathbb{Z}_q$.

To complete the construction, an error distribution χ on $R = \mathbb{Z}[x]/\langle\phi(x)\rangle$ which produces polynomials with relatively small coefficients must be defined. For now we do not specify χ and we assume samples chosen according to χ have sufficiently small coefficients. Suppose that the pairs $(a_i(x), b_i(x))$ are given where

- $a_i(x)$ are chosen uniformly random from R_q,
- $s(x)$ is chosen uniformly random from R_q,
- $e_i(x)$ are chosen random according the error distribution χ,
- $b_i(x) = a_i(x)s(x) + e_i(x)$.

Definition 4.1. (*The Ring Learning with Errors Problem (RLWE)*)

Decision-RLWE: Given arbitrarily many pairs $(a_i(x), b_i(x)) \in R_q^2$, distinguish these pairs from the pairs $(a_i(x), u_i(x)) \in R_q^2$ where $u_i(x)$ are truly random elements in R_q.

Search-RLWE: Given arbitrarily many pairs $(a_i(x), b_i(x)) \in R_q^2$, find $s(x) \in \mathbb{R}_q$.

In [24] they proved that, when $\phi(x)$ is chosen to be a cyclotomic polynomial, solving the search version of ***RLWE*** is equivalent to solving the Approximate Shortest Vector Problem(α-SVP) [1] on lattices in the ring $\mathbb{Z}[x]/\langle\phi(x)\rangle$. Since lattice problems are very hard and lattice based cryptography is thought to be resistant to quantum computers [4], ***RLWE*** problem holds a great promise for future cryptosystems.

The error distribution χ is generally chosen to be discrete Gaussian distribution [11]. What makes the Gaussian distribution preferential is it enables to give hard lattice problem reductions for ***RLWE*** and security proofs for algorithms based on ***RLWE***. Sampling Gaussian distributed elements is one of the factors that affects the running time of the algorithm [11]. According to the timings given in [26], sampling takes a considerable amount of time in the encryption process.

4.2 A Somewhat Homomorphic Encryption Scheme

In this section we explain the SwHE scheme presented by Zvika Brakerski and Vinod Vaikuntanathan in 2011 [7]. The scheme has parameters $q, t, n, f(x), \chi, A$ and D where q is a large prime number, $t \in \mathbb{Z}_q^*$ is a prime number which is much less than q, n is a positive integer, $f(x)$ is a degree n polynomial in $Z[x]$, χ is discrete Gaussian distribution with mean 0 and standard deviation σ on $R = Z[x]\langle f(x)\rangle$ and $A, D \in \mathbb{N}$ are the maximum numbers of homomorphic additions and multiplications (respectively) that the scheme supports. Encryption and decryption processes of this scheme uses polynomial arithmetic on the ring $Rq = Z_q[x]\langle f(x)\rangle$. The message space of the scheme is the ring $R_t = Z_t[x]/\langle f(x)\rangle$ and homomorphism will be over this ring.

The scheme has both symmetric key and public key variances. We explain them both but we mostly studied on the public key variant of the scheme. The operations are ordinary polynomial operations in R_q for both variants of the scheme.

4.2.1 Symmetric Key Variant of the Scheme

In this section we explain key generation, encryption and decryption algorithms of the symmetric key scheme and verify the decryption algorithm . We give a condition on some parameters in order to perform decryption correctly.

4.2.1.1 Key Generation

The secret key sk of the scheme is a vector $sk = (1, s(x), s^2(x), \ldots, s^D(x)) \in R_q^{D+1}$ where $s(x)$ is chosen by sampling a ring element $s(x) \in R_q$ according to distribution χ.

The power of $s(x)$ are used in decryption process. To decrypt a ciphertext with γ components we only use $sk = (1, s(x), s^2(x), \ldots, s^{\gamma-1}(x)) \in R_q^{\gamma}$. Private key can be set as $sk = s(x)$ and the powers are calculated when needed in the decryption process.

4.2.1.2 Encryption

To encode the message $m \in R_t$ sample polynomials $a(x) \in R_q$ uniformly at random and $e(x) \in R_q$ according to χ. Compute

$$c_0(x) = a(x)s(x) + te(x) + m \quad \text{and} \quad c_1 = -a(x).$$

The ciphertext is the pair $c = (c_0(x), c_1(x)) \in R_q^2$.

Even though the encryption map outputs ciphertexts as pairs in R_q^2, homomorphic operations may increase the number of components of the ciphertexts as we will see in 4.2.3.2. For this reason we denote a valid ciphertext by $c = (c_0(x), \ldots, c_\gamma(x)) \in R_q^{\gamma+1}$ where $\gamma \leq D$. The ciphertexts produced by an encryption algorithm will be referred to as fresh ciphertexts.

We also use the term ℓ_∞ *norm* while explaining the correctness of the algorithms. The ℓ_∞ norm of a vector $a = [a_0, a_1, a_1, \ldots, a_n]$ is defined as $max|a_i|$ and denoted by $\|a\|_\infty$. When we mention the ℓ_∞ norm of a polynomial, we mean the ℓ_∞ norm of its coefficient representation.

4.2.1.3 Decryption

To decrypt a ciphertext $c = (c_0(x), \ldots, c_\gamma(x))$ we just use the first $\gamma + 1$ element of the secret key, say $sk_\gamma = (1, s(x), \ldots, s^\gamma(x))$. Compute the following inner product

$$\langle c, sk_\gamma \rangle = \sum_{i=0}^{\gamma} c_i(x)s^i(x).$$

Then, the message can be obtained by $m = \langle c, sk_\gamma \rangle \mod t$.

Correctness of decryption of a fresh ciphertext:

Let $c = (c_0(x), c_1(x))$ be the encryption of the message m as given in 4.2.1.2. Then, the decryption process goes as follows:

$$\begin{aligned} c_0(x) + c_1(x)s(x) \mod t &= a(x)s(x) + te(x) + m + (-a(x))s(x) \mod t \\ &= (m + te(x)) \mod t. \end{aligned}$$

Claim 4.1. *The decryption process works correctly if only the error term (noise) satisfies the inequality* $\|e(x)\|_\infty < q/t$.

To prove this claim, let $e(x) = e_0 + \cdots + e_k x^k + \cdots + e_{n-1}x^{n-1}$ and $\|e(x)\|_\infty \geq q/t$. We know that the operations in the scheme are polynomial arithmetic in R_q and we don't usually indicate "mod q" in calculations. For the following calculations we think it would be better to use it to prove our point.

Without loss of generality we assume $\|e(x)\|_\infty = e_k$ and $e_k \geq q/t$, $e_i < q/t$ for $i \neq k, i = 0, \ldots, n-1$. We have $e_k \geq q/t \implies te_k \geq q \implies te_k = qb + r$ for some

positive integers b and $r < q$. Then,

$$\begin{aligned}(m + te(x)) \mod q &= (m + te_0 + \cdots + te_k x^k + \cdots + te_{n-1}x^{n-1}) \mod q \\ &\equiv m + te_0 + \cdots + rx^k + \cdots + te_{n-1}x^{n-1}, \\ ((m + te(x)) \mod q) \mod t &= (m + te_0 + \cdots + rx^k + \cdots + te_{n-1}x^{n-1}) \mod t \\ &\equiv m + rx^k \neq m.\end{aligned}$$

Therefore, in order to perform decryption correctly, the polynomial $e(x)$ must satisfy $\|e(x)\|_\infty < q/t$. We use this fact in upcoming verifications.

The symmetric key variant of the scheme is more efficient than the public key variant since it requires less polynomial multiplication. But as we give in Chapter 3 the benefits of public key homomorphic encryption schemes are greater than of symmetric keys.

4.2.2 Public Key Variant of the Scheme

This section contains the key generation, encryption and decryption algorithms of the public key variant of the *SwHE* scheme in [7]. It also includes verification of decryption of a fresh ciphertext.

4.2.2.1 Key Generation

Sample three ring elements $a_1(x), e(x)$ and $s(x)$ from R_q. Here $a_1(x)$ is chosen uniformly at random, $s(x)$ and $e(x)$ are chosen according to error distribution χ. The polynomial $s(x)$ is the private key. Compute $a_0(x) = -(a_1(x)s(x) + te(x))$. Publish the pair $pk = (a_0, a_1) = (-(a_1(x)s(x) + te(x)), a_1(x))$ as the encryption key.

4.2.2.2 Encryption

To encrypt the message $m \in R_t$ with a public key (a_0, a_1), sample three polynomials $u(x), h(x)$ and $g(x)$ in R_q according to distribution χ. Compute

$$c_0(x) = a_0(x)u(x) + th(x) + m \quad \text{and} \quad c_1(x) = a_1(x)u(x) + tg(x),$$

then the ciphertext is the pair $c = (c_0(x), c_1(x)) \in R_q^2$.

4.2.2.3 Decryption

Decryption is the same as the symmetric key variant's. We just compute

$$\left(\sum_{i=0}^{\gamma} c_i(x)s^i(x)\right) \mod t$$

to decrypt the ciphertext $c = (c_0(x), \ldots, c_\gamma(x)) \in R_q^{\gamma+1}$.

Correctness of decryption of a fresh ciphertext:

Let $c = (c_0(x), c_1(x))$ be the decryption of the message $m \in R_t$ given in 4.2.2.2. Then, the decrypiton goes as follows:

$$\begin{aligned}
&c_0(x) + c_1(x)s(x)\\
=&a_0(x)u(x) + th(x) + m + (a_1(x)u(x) + tg(x)s(x))\\
=&(-a_1(x)s(x) - te(x))u(x) + th(x) + m + a_1(x)s(x)u(x) + tg(x)s(x)\\
=&-a_1(x)s(x)u(x) - te(x)u(x) + th(x) + m + a_1(x)s(x)u(x) + tg(x)s(x)\\
=&m + t(-e(x)u(x) + h(x) + g(x)s(x)).
\end{aligned}$$

By using the same argument in the proof of claim 4.1, we can say that the decryption works correctly if only the error term satisfies the inequality

$$\|-e(x)u(x) + h(x) + g(x)s(x)\|_\infty < q/t.$$

4.2.3 Homomorphic Operations

The homomorphic operations of the *SwHE* [7] are defined the same for both symmetric and public key variants. Basically, the homomorphic operations treat ciphertexts as the coefficient representations of some polynomials and perform polynomial arithmetic on them.

4.2.3.1 Homomorphic Addition ($\star$)

Let $c = (c_0(x), \ldots, c_\gamma(x)) \in R_q^{\gamma+1}$ and $c' = (c'_0(x), \ldots, c'_\gamma(x)) \in R_q^{\gamma+1}$ be two valid ciphertexts. We assumed that the ciphertexts have the same size without loss of generality. If they have different lengths we pad the shorter one with zero components. The homomorphic addition is done by componentwise addition of ciphertexts. Then,

$$\begin{aligned}
c_{add} = c \star c' &= (c_0(x), \ldots, c_\gamma(x)) + (c'_0(x), \ldots, c'_\gamma(x))\\
&= (c_0(x) + c'_0(x), \ldots, c_\gamma(x) + c'_\gamma(x))\\
&= (\tilde{c_0}(x), \ldots, \tilde{c_\gamma}(x)).
\end{aligned}$$

Clearly, homomorphic addition does not increase the size of the ciphertexts.

Correctness of homomorphic addition:

Consider two fresh ciphertexts c and c' which are the encryptions of the messages m and m' respectively. Then, the sum of the ciphertexts

$$c = (c_0(x), c_1(x)) \quad \text{and} \quad c' = (c'_0(x), c'_1(x))$$

is equal to

$$c_{add} = (\tilde{c_0}(x), \tilde{c_1}(x)) = (c_0(x) + c'_0(x), c_1(x) + c'_1(x)).$$

The decryption of c_{add}

$$\begin{aligned}
&\tilde{c}_0(x) + \tilde{c}_1(x)s(x) \mod t \\
=&(c_0(x) + c'_0(x)) + (c_1(x) + c'_1(x))s(x) \mod t \\
=&c_0(x) + c_1(x)s(x) + c'_0(x) + c'_1(x)s(x) \mod t \\
=&m + te(x) + m' + te'(x) \mod t \\
=&m + m' + t(e(x) + e'(x)) \mod t
\end{aligned}$$

works correctly if only $\|e(x) + e'(x)\|_\infty < q/t$ and $m + m'$ is a valid message in R_t. Homomorphic encryption increases the noise just a modest amount.

4.2.3.2 Homomorphic Multiplication ($\diamond$)

Let $c = (c_0(x), \ldots, c_\gamma(x)) \in R_q^{\gamma+1}$ and $c' = (c'_0(x), \ldots, c'_\alpha(x)) \in R_q^{\alpha+1}$ be two valid ciphertexts. Choose a symbolic variable y and define two polynomials $f_1(y)$ and $f_2(y)$ such that

$$f_1(y) = \sum_{i=0}^{\gamma} c_i(x)y^i \quad \text{and} \quad f_2(y) = \sum_{j=0}^{\alpha} c'_j(x)y^j.$$

Multiply these polynomials

$$f_1(y)f_2(y) = \left(\sum_{i=0}^{\gamma} c_i(x)y^i\right)\left(\sum_{j=0}^{\alpha} c'_i(x)y^j\right) = \sum_{k=0}^{\gamma+\alpha} \tilde{c}_k(x)y^k.$$

Then, the result of the homomorphic multiplication $c \diamond c'$ is $c_{mult} = (\tilde{c}_0(x), \ldots, \tilde{c}_{\gamma+\alpha}(x)) \in R_q^{\gamma+\alpha+1}$. So homomorphic multiplication increases the number of components in ciphertexts.

Correctness of homomorphic multiplication:

Consider two fresh ciphertexts c and c' which are the encryptions of the messages m and m' respectively. Then, the product of the ciphertexts

$$c = (c_0(x), c_1(x)) \quad \text{and} \quad c' = (c'_0(x), c'_1(x))$$

is the three component ciphertext

$$\begin{aligned} c_{mult} &= (\tilde{c}_0(x), \tilde{c}_1(x), \tilde{c}_2(x)) \\ &= (c_0(x)c'_0(x), c_0(x)c'_1(x) + c_1(x)c'_0(x), c_1(x)c'_1(x)). \end{aligned}$$

The decryption of c_{mult}

$$\begin{aligned} &(\tilde{c}_0(x) + \tilde{c}_1(x)s(x), \tilde{c}_2(x)s^2(x)) \mod t \\ =&(c_0(x)c'_0(x)) + (c_0(x)c'_1(x) + c_1(x)c'_0(x))s(x) + (c_1(x)c'_1(x))s^2(x)) \mod t \\ =&(c_0(x) + c_1(x)s(x))(c_0(x) + c_1(x)s(x)) \mod t \\ =&(m + te(x))(m' + te'(x)) \mod t \\ =&mm' + t(me'(x) + m'e(x) + te(x)e'(x)) \mod t. \end{aligned}$$

Decryption of a ciphertext produced by homomorphic multiplication works correctly if $\|me'(x) + m'e(x) + te(x)e'(x)\|_\infty < q/t$ and mm' is in R_t. Homomorphic multiplication increases the noise a considerable amount.

The conditions we give in the verifications in 4.2.3.1 and 4.2.3.2 don't tell so much about the parameters of the scheme. The scheme has several dependent parameters and the following theorem gives a more clear condition on these parameters. The variables used in theorem are the scheme parameters as explained at the beginning of Section 4.2.

Theorem 4.2 ([26])**.** *The SwHE scheme given above can perform D multiplications followed by A additions if the following inequality holds:*

$$q \geq 4(2t\sigma^2\sqrt{n})^{D+1}(2n)^{D/2}\sqrt{A}.$$

This theorem offers a lower bound for the prime modulus q but in [26] it is stated that smaller q values can be used if they are confirmed by experiments.

4.2.4 Re-linearization

We see that homomorphic multiplication increases the number of components of ciphertexts. Brakerski and Vaikuntanathan presented a technique called *re-linearization* to reduce the ciphertext size [8]. This technique also works for this *SwHE* scheme as well and transforms a ciphertext with many components into a two component ciphertext. The output of this transformation is again a valid ciphertext and it can be decrypted in the same way.

Suppose that $ct_{mult} = (ct_0(x), ct_1(x), ct_2(x))$ is the ciphertext obtained by applying homomorphic multiplication to fresh ciphertexts

$$c = (c_0(x), c_1(x)) \quad \text{and} \quad c' = (c'_0(x), c'_1(x)),$$

which are the encryptions of the messages m and m' respectively. To perform relinearization, an extra key, called homomorphism key, is generated as a part of the

public key. A homomorphism key is defined as a vector $hk = (h_0, h_1, \ldots, h_{\lceil log_t q \rceil - 1})$ such that

$$h_i = (\alpha_i(x), \beta_i(x)) = (\alpha_i(x), -(\alpha_i(x)s(x) + te_i(x)) + t^i s^2(x)),$$

for $i = 0, 1, \ldots, \lceil log_t q \rceil - 1$ where $\alpha_i(x) \in R_q$ are chosen randomly and $e_i(x) \in R_q$ are chosen according to the error distribution χ for every i.

Step by step re-linearization:

1. Write the polynomial $ct_2(x)$ in base t representation,

$$ct_2(x) = \sum_{i=0}^{\lceil log_t q \rceil - 1} ct_{2,i}(x)t^i.$$

2. Set

$$c_1^{relin}(x) = ct_1(x) + \sum_{i=0}^{\lceil log_t q \rceil - 1} ct_{2,i}(x)\alpha_i(x), \tag{4.1}$$

$$c_0^{relin}(x) = ct_0(x) + \sum_{i=0}^{\lceil log_t q \rceil - 1} ct_{2,i}(x)\beta_i(x), \tag{4.2}$$

   where $h_i = (\alpha_i(x), \beta_i(x))$ come from the homomorphism key.

3. Output the ciphertext $ct_{mult} := (c_0^{relin}, c_1^{relin})$ which has size two.

Verification of re-linearization:

We have

$$ct_0(x) + ct_1(x)s(x) + ct_2(x)s^2(x) = mm' + te_{\text{mult}}(x).$$

By Equation (4.2),

$$\begin{aligned}
c_0^{relin}(x) =& ct_0(x) + \sum_{i=0}^{\lceil log_t q \rceil - 1} ct_{2,i}(x)\beta_i(x) \\
=& ct_0(x) + \sum_{i=0}^{\lceil log_t q \rceil - 1} ct_{2,i}(x)(-(\alpha_i(x)s(x) + te_i(x)) + t^i s^2(x))) \\
=& ct_0(x) - \underbrace{\left(\sum_{i=0}^{\lceil log_t q \rceil - 1} ct_{2,i}(x)\alpha_i(x)\right)}_{c_1^{relin}(x) - ct_1(x) \text{ by } (4.1)} s(x) - t \underbrace{\sum_{i=0}^{\lceil log_t q \rceil - 1} ct_{2,i}(x)e_i(x)}_{e_{relin}(x)} \\
&+ \underbrace{\left(\sum_{i=0}^{\lceil log_t q \rceil - 1} ct_{2,i}(x)t^i(x)\right)}_{ct_2(x)} s^2(x)
\end{aligned}$$

$$\implies c_0^{relin}(x) + c_1^{relin}(x)s(x) = ct_0(x) + ct_1(x)s(x) + ct_2(x)s^2(x) - te_{relin}(x)$$
$$\implies c_0^{relin}(x) + c_1^{relin}(x)s(x) = m(x)m'(x) + t(e_{mult}(x) - e_{relin}(x))$$
$$\implies \left(c_0^{relin}(x) + c_1^{relin}(x)s(x)\right) \mod t = m(x)m'(x).$$

Hence, the ciphertext produced by the re-linearization process could be decrypted correctly if the final error $e_{mult}(x) - e_{relin}(x)$ is small enough.

Although the re-linearization technique reduces the ciphertext size considerably it also increases the public key size. The public key has extra $\lceil log_t q \rceil$ elements. Running time of the homomorphic multiplication is also increased because extra $\lceil log_t q \rceil$ polynomial multiplication and addition are added to this process. In [26] they say that the benefits of re-linearization seem to dominate its negative effects for large t values. From this point of view a control mechanism can be added to implementations to decide whether using re-linearization is an advantage or not.

4.2.5 How to Encode Messages and How to Choose Message Space

There are several factors that affect the efficiency of the algorithm. In this section we pay attention to the following factors: the message space parameter t, the number of encryptions needed and the depth of the arithmetic circuit of the function to be computed on encrypted data. Since addition costs much less than multiplication, we just count multiplications to calculate the depth [9]. With the given examples, we try to form an opinion about the effects of different encoding techniques that are chosen according to the function to be computed on encrypted data.

Note that, we use b and $b(x)$ notations interchangeably for a polynomial $b(x)$ in the following examples.

Example 4.1. Suppose we want to compute the sum of k integers $z_1, \ldots, z_k \in \mathbb{Z}$ homomorphically by using the public key *SwHE* we explained above with the private key $s \in R_q$ and the public key $(a_0, a_1) \in R_q^2$. Consider the following cases

1. For the first case, suppose we don't use any encoding techniques and treat the integers as plaintexts. After k encryptions we have $z_1', \ldots, z_k' \in R_q$ where

   $$z_i' = \mathcal{E}(z_i) = (a_0u_i + th_i + z_i, a_1u_i + tg_i).$$

   We want to compute a ciphertext by applying the operation $\star$ to the elements $z_1', \ldots, z_k'$, which gives the result $\sum_{i=1}^k z_i$ when decrypted. Clearly,

   $$z_1' \star \cdots \star z_k' = \left(a_0 \sum_{i=1}^k u_i + t\sum_{i=1}^k h_i + \sum_{i=1}^k z_i, a_1 \sum_{i=1}^k u_i + t\sum_{i=1}^k g_i\right).$$

   We assume the condition on the ℓ_∞ norm of the error term is satisfied. The other condition on $\sum_{i=1}^k z_i$ being in R_t would be satisfied if $t > \sum_{i=1}^k z_i$. We have

- k encryptions $\longrightarrow 2k$ polynomial multiplications,
- $t > \sum_{i=1}^{k} z_i$,
- $k-1$ homomorphic addition.

Therefore, the computation above can be done with a *SwHE* with $D = 1$ and $t > \sum_{i=1}^{k} z_i$ since it requires no multiplication.

2. Let us encode the integers as polynomials of degree less than n and coefficients less than an integer w for the second case. Suppose $z_i < w^n$ for every $i = 1, \ldots, k$. To be more precise we define the polynomials $f_i(x) = \sum_{j=1}^{n-1} z_{i,j} x^j$ where $z_{i,j}$ are the digits of base w representation of z_i. Therefore, $f_i(w) = z_i$. We take $f_i(x) = f_i$ as our plaintexts. After k encryptions we have $f_1', \ldots, f_k' \in R_q$, where

$$f_i' = \mathcal{E}(f_i) = (a_0 u_i + t h_i + f_i, a_1 u_i + t g_i).$$

Applying the $\star$ operation gives us

$$f_1' \star \cdots \star f_k' = \left(a_0 \sum_{i=1}^{k} u_i + t \sum_{i=1}^{k} h_i + \sum_{i=1}^{k} f_i, a_1 \sum_{i=1}^{k} u_i + t \sum_{i=1}^{k} g_i \right).$$

Assuming that the ℓ_∞ norm of the final error is less than q/t, the only thing that we should be concerned about is $\sum_{i=1}^{k} f_i$ being an element of the message space R_t. Since the coefficients of the polynomials f_i are less than w, the coefficients of the sum must be less than kw. So choosing $t > kw$ would suffice. In this scenario we have

- k encryptions $\longrightarrow 2k$ polynomial multiplications,
- $t > wk$,
- $k-1$ homomorphic additions.

Therefore, the computation above can be done with a *SwHE* with $D = 1$ and $t > wk$ since it requires no multiplication.

For this example the second case seems better since $wk < \sum_{i=1}^{k} z_i$. By Theorem 4.2 smaller values of t implies smaller values for q.

Example 4.2. Suppose we want to check homomorphically whether two l-bit integers x and y are equal or not. We can construct an arithmetic circuit for the following function:

$$EQ(x, y) = \begin{cases} 1 \text{ if } x = y \\ 0 \text{ if } x \neq y. \end{cases}$$

Let us consider the following cases:

1. We don't use any encoding techniques and encrypt the messages as integers in $\mathbb{Z}_q$. Let $\mathcal{E}(x) = x'$ and $\mathcal{E}(y) = y'$. By *Fermat's little* theorem we know that

$$k^{q-1} \mod q \equiv \begin{cases} 1 \text{ if } k \neq 0 \\ 0 \text{ if } k = 0, \end{cases}$$

for every $k \in \mathbb{Z}_q$. Therefore, we can define the function such that

$$EQ(x, y) = 1 - (x - y)^{q-1}.$$

To check the equality of x and y homomorphically, in other words to compute $1 - (x - y)^{q-1}$ homomorphically we should do the following:

$$(1, \underbrace{0, \ldots, 0}_{q-1}) \star \left[(-1, 0) \diamond \underbrace{(x' \star y') \diamond \cdots \diamond (x' \star y')}_{q-1}\right].$$

Then, we have

- 2 encryptions $\longrightarrow$ 4 polynomial multiplications,
- $t > max(x, y)$,
- $\approx q$ homomorphic multiplications, q homomorphic additions.

In this case the computations can be done homomorphically with a *SwHE* with $D = \lceil \log p \rceil$.

2. We encrypt the integers bitwise. Let $x = (x_{l-1}, \ldots, x_0)$ and $y = (y_{l-1}, \ldots, y_0)$ be the bit representations. Suppose $\mathcal{E}(x_i) = x'_i$ and $\mathcal{E}(y_i) = y'_i$. Then, the equality function can be defined by:

$$EQ(x, y) = \Lambda_{i=1}^{l} \left(1 \oplus x_i \oplus y_i\right).$$

To check the equality we should calculate:

$$[(1, 0) \star (x'_{l-1}) \star (y'_{l-1})] \diamond \cdots \diamond [(1, 0) \star (x'_0) \star (y'_0)].$$

Then, we have

- $2l$ encryptions $\longrightarrow$ $4l$ polynomial multiplications,
- $t = 2$,
- $\approx l$ homomorphic multiplications, l homomorphic additions.

In this case the computations above can be done homomorphically with a *SwHE* with $D = \lceil \log(l - 1) \rceil$.

For this example encoding the messages as bit strings and encrypting them bit wise seems to be the better solution to check the equality of two integers in encrypted form.

These examples clearly tell us that encoding technique plays an important role on the efficiency of the algorithm. Then, it should be chosen carefully based on the message space, the number of encryptions and the number of multiplications.

CHAPTER 5

OUR WORK

We studied the polynomial multiplication in the ring $R_q = \mathbb{Z}_q[x]/\langle f(x)\rangle$ that was used for encryption and decryption in the *SwHE* scheme explained in the previous chapter. The modulus polynomial $f(x)$ is set to be $x^n + 1$ which is referred as the standard parameter setting in [7]. We use the same specifications for the parameters n and q as in [26]. The characteristics of these parameters are as follows:

- n is a positive integer which is a power of 2,
- q is a large prime number and $q \equiv 1(\mod 2n)$.

Our study is mostly concentrated on reducing the computational complexity of the algorithm with the specified parameter settings. Considering that the predominant operation in the scheme is polynomial multiplication in $R_q = \mathbb{Z}_q/\langle x^n + 1\rangle$, we decided to work on ideas for speeding up this operation.

The choice of n makes the polynomial $f(x) = x^n + 1$ the $2n$-th cylotomic polynomial by Proposition 2.1.4. Applying Theorem 2.2 to $2n$-th cyclotomic polynomial $f(x)$ with $q \equiv 1(\mod 2n)$, gives that the polynomial $x^n + 1$ factors into n linear polynomials over $\mathbb{Z}_q$. In other words $\mathbb{Z}_q$ includes all primitive $2n$-th roots of unity, hence all $2n$-th roots of unity. These parameter settings are used in many studies in order to take advantage of *FFT* for fast polynomial multiplication over the finite field $\mathbb{Z}_q$. As we describe in 5.1.1 using *FFT* technique instead of naive method for polynomial multiplication in R_q leads to an improvement from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$.

In our studies, we use a different approach for efficient polynomial multiplication. When we examine the coefficients of the product of two polynomials in R_q by replacing x^n by -1, we notice a special situation. It turns out that we can represent a polynomial multiplication in R_q as a *TMVP*. We can multiply two polynomials in R_q in $\mathcal{O}(n^{\log 3})$ with this technique as we explain in 5.1.2.

5.1 Polynomial Multiplication Modulo $x^n + 1$

Let $a(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$ and $b(x) = b_0 + b_1x + b_2x^2 + \cdots + b_{n-1}x^{n-1}$ be two polynomials in $R_q = \mathbb{Z}_q/\langle x^n + 1\rangle$. Product of these two polynomials modulo

$x^n + 1$ is again a degree less than n polynomial, say $c(x)$. Then, we have

$$\begin{aligned} c(x) &= a(x).b(x) \mod x^n + 1 = c'(x) \mod x^n + 1 \\ &= (a_0 + a_1x + \cdots + a_{n-1}x^{n-1})(b_0 + b_1x + \cdots + b_{n-1}x^{n-1}) \mod x^n + 1 \\ &= c'_0 + c'_1x + c'_2x^2 + \cdots + c'_{2n-2}x^{2n-2} \mod x^n + 1 \\ &= (c'_0 - c'_n) + (c'_1 - c'_{n+1})x + (c'_2 - c'_{n+2})x^2 + \cdots + (c'_{n-2} - c'_{2n-2}) + c'_{n-1}x^{n-1} \\ &= c_0 + c_1x + c_2x^2 + \cdots + c_{n-1}x^{n-1}, \end{aligned}$$

where

$$c'_k = \sum_{i+j=k} a_ib_j \quad \text{and} \quad c_k = \sum_{i+j=k} a_ib_j - \sum_{i+j=n+k} a_ib_j.$$

Our goal is to find the coefficients c_i efficiently. For this, we work on two techniques that utilize *FFT* and *TMVP* algorithms we explained in Sections 2.3 and 2.4. We calculate their computational complexity and compare the efficiency of the algorithms.

5.1.1 Multiplication using FFT

Since we know Z_q has all $2n$-th primitive roots of unity (by Theorem 2.2), we can use *FFT* to multiply two polynomials. First we need one of the roots of $f(x) = x^n + 1$ which is one of the primitive $2n$-th roots of unity. That means we are looking for an element ζ in $\mathbb{Z}_q^*$ such that $\zeta^{2n} = 1$ and $\zeta^n = -1$. By *Fermat's little* theorem we know $z^{q-1} \equiv 1 \mod q$ for every $z \in \mathbb{Z}_q^*$. We choose $q \equiv 1 \mod 2n$, then $q = 2nk + 1$ for some positive integer k. Therefore, $z^{2nk} = (z^k)^{2n} = 1$ for every $z \in \mathbb{Z}_q^*$. Since the square roots of 1 are 1 and -1 we have $z^{kn} = 1$ or $z^{kn} = -1$. The latter means z^k is a primitive $2n$-th root of unity. There are exactly $q - 1 = 2nk$ elements in $\mathbb{Z}_q^*$ and these elements satisfy either $z^{kn} = 1$ or $z^{kn} = -1$ [22]. By choosing a few random elements and checking whether they satisfy the second equation or not, we can get a primitive $2n$-th root of unity $\zeta = z^k$.

Suppose we have ζ, a primitive $2n$-th root of unity which is one of the roots of the polynomial $x^n + 1$ in $\mathbb{Z}_q^*$. First we pad the polynomials $a(x)$ and $b(x)$ with higher degree zero coefficient terms until their degree become $2n - 1$. Then, we evaluate the polynomials $a(x)$ and $b(x)$ at the points $1, \zeta, \ldots, \zeta^{2n-1}$ using *FFT* as described in Section 2.3.

Let A and B be the coefficient representations of the polynomials $a(x)$ and $b(x)$ respectively. To evaluate $\boldsymbol{DFT}_{2n}(A)$ and $\boldsymbol{DFT}_{2n}(B)$ we need $6n \log n$ ring operations as we explained in 2.3. Then, we compute

$$C' = DFT_{2n}(A) \cdot DFT_{2n}(B),$$

where $(\cdot)$ represents the dot product of two vectors. Obviously, the result C' is the point value representation of $c'(x)$ at $2n$-th roots of unity. As explained in 2.3, interpolating C' by

$$C' = DFT_{2n}^{-1}(DFT_{2n}(A) \cdot DFT_{2n}(B))$$

gives us the coefficient representation of the polynomial $c'(x)$ and we can do this with $3n \log n$ ring operations. Once we calculate c'_i, finally by computing $c_i = c'_i - c'_{i+n}$ we get the coefficient representation of the polynomial $c(x)$ and we need n ring additions for this step. Thus, multiplying two polynomials modulo $x^n + 1$ requires

$$9n \log n + \mathcal{O}(n)$$

ring operations in total when we use ***FFT***.

5.1.2 Multiplication using TMVP

When we write the coefficients of $c(x)$ in expanded form it is easy to see that there is a pattern.

$$\begin{aligned}
c_0 = (c'_0 - c'_n) &= a_0b_0 - a_1b_{n-1} - a_2b_{n-2} - \cdots - a_{n-3}b_3 - a_{n-2}b_2 - a_{n-1}b_1, \\
c_1 = (c'_1 - c'_{n+1}) &= a_0b_1 + a_1b_0 - a_2b_{n-1} - a_3b_{n-2} - \cdots - a_{n-2}b_3 - a_{n-1}b_2, \\
c_2 = (c'_2 - c'_{n+2}) &= a_0b_2 + a_1b_1 + a_2b_0 - a_3b_{n-1} - \cdots + a_{n-2}b_4 - a_{n-1}b_3, \\
&\vdots \\
c_{n-1} = c'_{n-1} &= a_0b_{n-1} + a_1b_{n-2} + \cdots + a_{n-3}b_3 + a_{n-2}b_1 + a_{n-1}b_0.
\end{aligned}$$

By using the pattern in the equalities above we can express the coefficients c_i of the product polynomial $c(x)$ for $i = 0, .., n-1$ as a resulting vector of a ***TMVP*** as follows:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ \vdots \\ c_{n-2} \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} a_0 & -a_{n-1} & -a_{n-2} & \dots & -a_3 & -a_2 & -a_1 \\ a_1 & a_0 & -a_{n-1} & \dots & -a_4 & -a_3 & -a_2 \\ a_2 & a_1 & a_0 & \dots & -a_5 & -a_4 & -a_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & a_{n-4} & \dots & a_1 & a_0 & -a_{n-1} \\ a_{n-1} & a_{n-2} & a_{n-3} & \dots & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_{n-2} \\ b_{n-1} \end{bmatrix}.$$

Hence, the coefficients of the product of two polynomials modulo $x^n + 1$ can be calculated by multiplying a Toeplitz matrix and a vector in

$$6.5n^{\log 3} + \mathcal{O}(n) \in \mathcal{O}(n^{\log 3}),$$

as we explain in Section 2.4. To specify a $n \times n$ Toeplitz matrix it is enough to give an array of length $2n - 1$ which contains the elements in the first column and the first row of the matrix. For example the Toeplitz matrix above can be represented by the array $[a_{n-1}, a_{n-2}, \dots, a_0, -a_{n-1}, \dots, -a_1]$ of length $2n - 1$. This saves a lot of memory in implementations because to store a general $n \times n$ matrix we need an array of dimension n^2. We implemented this algorithm and we observed that the multiplication of two polynomial in R_q takes 1020 milliseconds on average when $n = 2^{14}$ and $\lfloor \log q \rfloor = 615$.

5.1.3 Comparison

The computational complexity of the techniques we describe in Sections 5.1.1 and 5.1.2 are $9n \log n + \mathcal{O}(n)$ and $6.5n^{\log 3} + \mathcal{O}(n)$ respectively. Although the first technique(the one using *FFT*) has slightly lower complexity, we think that the second technique(the one using *TMVP*) can also be used for implementations. Memory requirements of implementations using *FFT* method is more than the ones using *TMVP*. *FFT* method is efficient when n is a power of 2 while *TMVP* might be efficient for other values of n.

CHAPTER 6

CONCLUSION

Being able to operate on encrypted data would offer a whole new level of privacy. Theoretically, it is possible to do this with *FHE*. To date, several cryptosystems which satisfy fully homomorphism have been presented, but they are not efficient enough to use in practical applications yet. Because evaluating arbitrary functions on encrypted data requires lots of arithmetic operations. *SwHE* schemes, that support a limited number of operations, are seen to be more eligible for practical applications. In this thesis we examine the *SwHE* scheme presented by Zvika Brakerski and Vinod Vaikuntanathan [7].

In Chapter 2, we gave some background information to help to understand the subject better. We gave some basic definitions such as roots of unity, cyclotomic polynomial, coefficient and point value representations of a polynomial. Also some properties were given. We explained discrete Fourier transform *(DFT)* of a polynomial and fast Fourier transform *(FFT)* to perform *DFT* faster. Toeplitz matrices were introduced and efficient Toeplitz matrix vector product *(TMVP)* algorithm explained. Computational complexities of both *FFT* and *TMVP* were computed.

In Chapter 3,we defined homomophic encryption formally. We mentioned some possible practical applications and tried to provide some insight on constructing functions that can be computed homomorphically. In Chapter 4 we explained the *RLWE* problem and the *SwHE* scheme based on this problem. The scheme is built on the arithmetic on the polynomial ring $\mathbb{Z}_q[x]/\langle x^n + 1\rangle$ and uses a randomized map for encryption. The encryption and decryption algorithms of symmetric key and public key variants of the scheme were explained. For both of them conditions on the parameters that ensure the correctness of the decryption process were given. Then, we spelled out the homomorphic operations. We observed that the homomorphic multiplication increases ciphertext size and blows up the error term. The re-linearization technique, which reduces the size of the ciphertext after homomorphic multiplication, was introduced. At the end of Chapter 4 we gave some encoding techniques with examples.

In Chapter 5, we described two techniques for polynomial multiplication in the ring $\mathbb{Z}_q[x]/\langle x^n + 1\rangle$ using *FFT* and *TMVP*. We calculated computational complexity of the algorithms and compared their efficiency. Although the computational complexity of *FFT* technique is a little less than *TMVP* technique, we think that *TMVP* technique works as well as *FFT* technique in implementations. We shared our arguments about

this conclusion.

To summarize, during our research we studied on a proposed *SwHE* scheme in detail and mostly focused on developing ideas to reduce the running time of implementations. In this context, we adapted a fast polynomial multiplication technique to the algorithm and shared the computational complexity. Different multiplication techniques might be developed to use in this scheme as a future study. We hope that our research will be helpful for others who are willing to study on this subject.

REFERENCES

[1] M. Ajtai, The shortest vector problem in L 2 is NP-hard for randomized reductions, in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 10–19, ACM, 1998.

[2] M. Bellare and P. Rogaway, Optimal asymmetric encryption, in *Workshop on the Theory and Application of of Cryptographic Techniques*, pp. 92–111, Springer, 1994.

[3] J. Benaloh, Dense probabilistic encryption, in *Proceedings of the workshop on selected areas of cryptography*, pp. 120–128, 1994.

[4] D. J. Bernstein, J. Buchmann, and E. Dahmen, *Post-quantum cryptography*, Springer Science & Business Media, 2009.

[5] D. Boneh, E.-J. Goh, and K. Nissim, Evaluating 2-DNF formulas on ciphertexts., in *TCC*, volume 3378, pp. 325–341, Springer, 2005.

[6] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, (leveled) fully homomorphic encryption without bootstrapping, ACM Transactions on Computation Theory (TOCT), 6(3), p. 13, 2014.

[7] Z. Brakerski and V. Vaikuntanathan, Fully homomorphic encryption from ring-lwe and security for key dependent messages, in *Annual cryptology conference*, pp. 505–524, Springer, 2011.

[8] Z. Brakerski and V. Vaikuntanathan, Efficient fully homomorphic encryption from (standard) lwe, SIAM Journal on Computing, 43(2), pp. 831–871, 2014.

[9] J. H. Cheon, M. Kim, and K. Lauter, Homomorphic computation of edit distance, in *International Conference on Financial Cryptography and Data Security*, pp. 194–212, Springer, 2015.

[10] T. H. Cormen, *Introduction to algorithms*, MIT press, 2009.

[11] N. C. Dwarakanath and S. D. Galbraith, Sampling from discrete Gaussians for lattice-based cryptography on a constrained device, Applicable Algebra in Engineering, Communication and Computing, 25(3), pp. 159–180, 2014.

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

[13] H. Fan and M. A. Hasan, A new approach to subquadratic space complexity parallel multipliers for extended binary fields, IEEE Transactions on Computers, 56(2), pp. 224–233, 2007.

[14] C. Gentry and S. Halevi, Fully homomorphic encryption without squashing using depth-3 arithmetic circuits, in *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pp. 107–109, IEEE, 2011.

[15] C. Gentry and S. Halevi, Implementing Gentry's fully-homomorphic encryption scheme., in *EUROCRYPT*, volume 6632, pp. 129–148, Springer, 2011.

[16] 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*, pp. 75–92, Springer, 2013.

[17] C. Gentry et al., Fully homomorphic encryption using ideal lattices., in *STOC*, volume 9, pp. 169–178, 2009.

[18] S. Goldwasser and S. Micali, Probabilistic encryption & how to play mental poker keeping secret all partial information, in *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pp. 365–377, ACM, 1982.

[19] Y. Ishai and A. Paskin, Evaluating branching programs on encrypted data, in *TCC*, volume 4392, pp. 575–594, Springer, 2007.

[20] D. Kahn, *The Codebreaker: The Story of Secret Writing*, Macmillam, 1976.

[21] D. Levy, Introduction to numerical analysis, Department of Mathematics and Center for Scientific Computation and Mathematical Modeling (CSCAMM) University of Maryland, pp. 2–2, 2010.

[22] R. Lidl and H. Niederreiter, *Introduction to finite fields and their applications*, Cambridge university press, 1994.

[23] 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*, pp. 1219–1234, ACM, 2012.

[24] V. Lyubashevsky, C. Peikert, and O. Regev, On ideal lattices and learning with errors over rings, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 1–23, Springer, 2010.

[25] D. Naccache and J. Stern, A new public key cryptosystem based on higher residues, in *Proceedings of the 5th ACM conference on Computer and communications security*, pp. 59–66, ACM, 1998.

[26] M. Naehrig, K. Lauter, and V. Vaikuntanathan, Can homomorphic encryption be practical?, in *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pp. 113–124, ACM, 2011.

[27] T. Okamoto and S. Uchiyama, A new public-key cryptosystem as secure as factoring, Advances in Cryptology—EUROCRYPT'98, pp. 308–318, 1998.

[28] C. Paar and J. Pelzl, *Understanding cryptography: a textbook for students and practitioners*, Springer Science & Business Media, 2009.

[29] P. Paillier et al., Public-key cryptosystems based on composite degree residuosity classes, in *Eurocrypt*, volume 99, pp. 223–238, Springer, 1999.

[30] K. B. Petersen and M. S. Pedersen, The matrix cookbook. version: November 15,, 2012.

[31] R. L. Rivest, L. Adleman, and M. L. Dertouzos, On data banks and privacy homomorphisms, Foundations of secure computation, 4(11), pp. 169–180, 1978.

[32] R. L. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM, 21(2), pp. 120–126, 1978.

[33] T. Sander, A. Young, and M. Yung, Non-interactive cryptocomputing for NC/sup 1, in *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pp. 554–566, IEEE, 1999.

[34] A. Shpilka, A. Yehudayoff, et al., Arithmetic circuits: A survey of recent results and open questions, Foundations and Trends® in Theoretical Computer Science, 5(3–4), pp. 207–388, 2010.

[35] N. P. Smart and F. Vercauteren, Fully homomorphic encryption with relatively small key and ciphertext sizes., in *Public Key Cryptography*, volume 6056, pp. 420–443, Springer, 2010.

[36] N. P. Smart and F. Vercauteren, Fully homomorphic SIMD operations, Designs, codes and cryptography, pp. 1–25, 2014.

[37] D. Stehlé and R. Steinfeld, Faster fully homomorphic encryption, Advances in Cryptology-ASIACRYPT 2010, pp. 377–394, 2010.

[38] J. Stoer and R. Bulirsch, *Introduction to numerical analysis*, volume 12, Springer Science & Business Media, 2013.

[39] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, Fully homomorphic encryption over the integers, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 24–43, Springer, 2010.

[40] H. S. Wilf, Algorithms and complexity, Summer, 1994.

[41] A. C. Yao, Protocols for secure computations, in *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pp. 160–164, IEEE, 1982.