CRT Based Somewhat Homomorphic Encryption Over the Integers

by

Ali Saeed Alzahrani
B.Sc., Umm Alqura University, 2010

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Applied Science

in the Department of Electrical and Computer Engineering

CRT Based Somewhat Homomorphic Encryption Over the Integers

by

Ali Saeed Alzahrani
B.Sc., Umm Alqura University, 2010

Supervisory Committee

Dr. Fayez Gebali, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Haytham El Miligi , Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. Fayez Gebali, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Haytham El Miligi , Member
(Department of Electrical and Computer Engineering)

ABSTRACT

Over the last decade, the demand for privacy and data confidentiality in communication and storage processes have increased exponentially. Cryptography can be the solution for this demand. However, the critical issue occurs when there is a need for computing publicly on sensitive information or delegating computation to untrusted machines. This must be done in such a way that preserves the information privacy and accessibility. For this reason, we need an encryption algorithm that allows computation on information without revealing details about them. In 1978 Rivest, Adleman and Dertouzos [RAD78] raised a crucial question: can we use a special privacy homomorphism to encrypt the data and do an unlimited computations on it while it remains encrypted without the necessity of decrypting it? Researchers made extensive efforts to achieve such encryption algorithm. In this paper, we introduce the implementation of the CRT-based somewhat homomorphic encryption over the integers scheme. The main goal is to provide a proof of concept of this new and promising encryption algorithm.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

In the name of Allah, the Most Gracious and the Most Merciful

Alhamdulillah, all praises belongs to Allah the merciful for his blessing and guidance. He gave me the strength to reach what I desire. I would like to thank:

My parents, my family, for supporting me at all stages of my education and their unconditional love.

My Supervisor, Dr. Fayez Gebali, for all the support, encouragement, and encouragement he provided to me during my work under his supervision. It would not have been possible to finish my research without his invaluable help of constructive comments and suggestions.

My Committee, Dr. Haytham El Miligi, for his precious time and valuable suggestions for the work done in this dissertation.

My colleague, Mr. Samer Moein, for introducing me to this field, and his invaluable help, reviewing and guidance on the way to finish my thesis.

My labmate, Mr. Nicholas Houghton, for his endless efforts and participant to build the code.

WestGrid, Dr. Belaid Moa, for his invaluable comments, suggestion, and supporting us with the latest hardware to test the code.

The Ministry of Higher Education in Saudi Arabia, for funding me with a Scholarship.

Ali Alzahrani

DEDICATION

To my parents, **Saeed Alzahrani and Jumah Alzahrani** for their love, prayers, and encouragement.

To my lovely wife, **Reem Alzahrani** for always standing by me, and believing in me.

To my beautiful daughter and son **Aryam, and Muhammad**.

Chapter 1

Introduction

1.1 Overview

The growth of information processing and telecommunication is continuing. The capabilities of such technology have increased dramatically. People are interacting and communicating by using a group of digital packets with each other. This type of communication has a huge impact on our personal and economic lives. Gradually, we rely on technology to do things we used to do face-to-face and on paper. There are many issues associated with such technology. How do we preserve the privacy of every syllable transmitted over a channel. How do we authenticate the identity of an entity without any physical proof. How could we investigate the authenticity of a piece of information. Over the last decade, the demand for privacy and data confidentiality in communication and storage processes have increased exponentially. Cryptography can be the solution for this demand. Cryptography provides several methods for guaranteeing information privacy, for ensuring data has not been changed and for verifying the source of information. Moreover, tamper-resistant hardware is also used to store and process delicate information. This work is mainly focused on cryptography methods to secure the data. Cryptography is an old and fascinating art. David Kahn traces the development of the cryptography study [Kah74]. He reported all majors advancements starting from the first discovered attempt of secret writing by Egyptians nearly 4000 years ago, to the mid twentieth century. The revolution of computers and communication brought the need to protect data in digital form.

1.2 Motivation

Cryptography is an old and fascinating art. David Kahn traces the development of the cryptography study [Kah74]. He reported all majors advancements starting from the first discovered attempt of secret writing by Egyptians nearly 4000 years ago, to the mid twentieth century. The revolution of computers and communication brought the need to protect data in digital form.

The demand of information security and privacy started the modern cryptography era. Modern cryptography is mainly based on mathematical theory. Cryptographic algorithms are built around the hardness of computation assumption, making them almost impossible to break by an attacker. Theoretically it is possible to overcome such algorithms but it is hard to do so in any practical mean. Modern field of cryptography algorithms can be divided into two main areas: first one is *Symmetric (Private Key) Algorithms*. Symmetric algorithms refers to encryption methods where the same key is shared by the sender and the receiver.

The most important advancement of cryptography happened in 1976 when the notion of public key algorithm was described by Whitfield Diffie and Martin Hellman in their breakthrough paper *new direction in cryptography* [DH76]. This paper introduced the second field of cryptography algorithms which is *Asymmetric (Public Key) Algorithms*. In asymmetric algorithms, sender and receiver use different keys for encryption and decryption. In 1978, the first and most popular asymmetric encryption algorithm RSA was introduced by three cryptographers Rivest, Shamir, and Adleman [RSA78]. The RSA algorithm has two keys, public key and private key. Sender will use the public key to encrypt the massage and the receiver uses the private key to retrieve the original massage. Therefore, without the private key the massage content is unreadable, and we reach the goal of encryption algorithms.

However, the critical issue occurs when there is a need for computing publicly on sensitive information or delegating computation to untrusted machines. This must be done in such a way that preserves the information privacy and executability. For this reason, we need an encryption algorithm that allows computation on information without revealing details about them. In 1978 Rivest, Adleman and Dertouzos [RAD78] raised a crucial question: can we use a special privacy homomorphism to

encrypt the data and do an unlimited computations on it while it remains encrypted without the necessity of decrypting it?

After this question, researchers made an extensive efforts to achieve such encryption algorithm. Unfortunately, there has been little progress in realizing if such encryption algorithms exist. In 2009, Craig Gentry in his Phd thesis theoretically introduced the first possible construction of such homomorphic encryption algorithm [Gen09]. Gentry's encryption scheme is based on ideal lattices. After Gentry's breakthrough, several homomorphic encryption algorithms have been proposed that are based on different mathematical assumptions. Van Dijk, Craig Gentry, Shai Halevi and Vinod Vaikuntanathan (DGHV) described the first fully homomorphic encryption over the integers scheme [VDGHV10].

Many efforts have been made toward the improvement of the scheme. The batch fully homomorphic encryption over the integers scheme was introduced by Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrede Lepoint, Mehdi Tibouchi, and Aaram Yun [CCK$^+$13]. This variant allows an encryption of a vector of bits.

1.3 Contributions

The main goal of our work is to realize the existing of homomorphic encryption schemes by implementing them. The contributions of our work can be summarized as follow:

- Proof of concept of a new CRT-based somewhat homomorphic encryption over the integers scheme. We presents our implementation and a full description of the scheme. This implementation is based on a variant version of the (DGHV) namely batch fully homomorphic encryption over the integers scheme.

- Comparison between our implementation results and the published works.

1.4 Thesis Organization

This section presents a map of the thesis and a short description of each chapter.

Chapter 2 reviews basic background of cryptography algorithms and several important function and concept related to cryptography. Chapter 3 gives an overview of the research that been done in the area in order to achieve a fully homomorphic encryption scheme. Also it presents a short description about a variant state of the art fully homomorphic encryption schemes. Chapter 4 recalls essential elementary number notations for fully homomorphic encryption over the integers schemes algorithms. It also presents a full description of the somewhat homomorphic encryption algorithms that been implemented in our works. Chapter 5 contains a description of our platform, results of the thesis, that includes the evaluation of the reported result and a comparisons with the work of others. Chapter 6 concludes our thesis and restates the contributions of this work.

Chapter 2

Cryptography

2.1 Cryptography Foundation

2.1.1 Terminology

The art of cryptography contains several important operations. *Encryption* which is the process of transforming readable information into unreadable information. The encryption process requires an *encryption key*. *Plaintext* is the readable data, the output from the encryption process is known as *ciphertext*. The conversion of a plaintext into a ciphertext is performed by the encryption algorithm. *Decryption* is the process of converting encrypted data back to readable form. The decryption procedure requires a *decryption key* in order to decipher or decrypt the message. The decryption key can be the same as the encryption key or different.

Figure 2.1: Encryption and Decryption

An *entity* or *party* is a person or a device (*e.g.* Computer) which transmits or receives data. A *sender* is an entity that legitimately sends some information to another party through a transmission channel. A *receiver* is the intended recipient of information. An adversary is a malicious entity in communication system which

tries to prevent legitimate users from achieving their goal (*e.g.* information security and privacy).

Cryptology is the study of secret writing, which splits into two opposites aspects. *Cryptography* is the art and technique of preserving messages security, with the goal of enabling a secure communication mechanism. *Cryptographers* are the practitioners of cryptography. *Cryptanalysis* the art and process of deciphering coded messages without possessing the key. *Cryptanalysts* are the practitioners of cryptanalysis. Cryptanalysis is a very important aspect for modern cryptosystems. Because it is the only way to make sure that an encryption algorithm is secure. Cryptography is not a new technique. In fact, cryptography is an old art, with ancient schemes (e.g. Egyptian codes) dating back to more than 2000 B.C.E. [PPP09]. Figure 2.2 shows an overview of the field of cryptography and related fields.

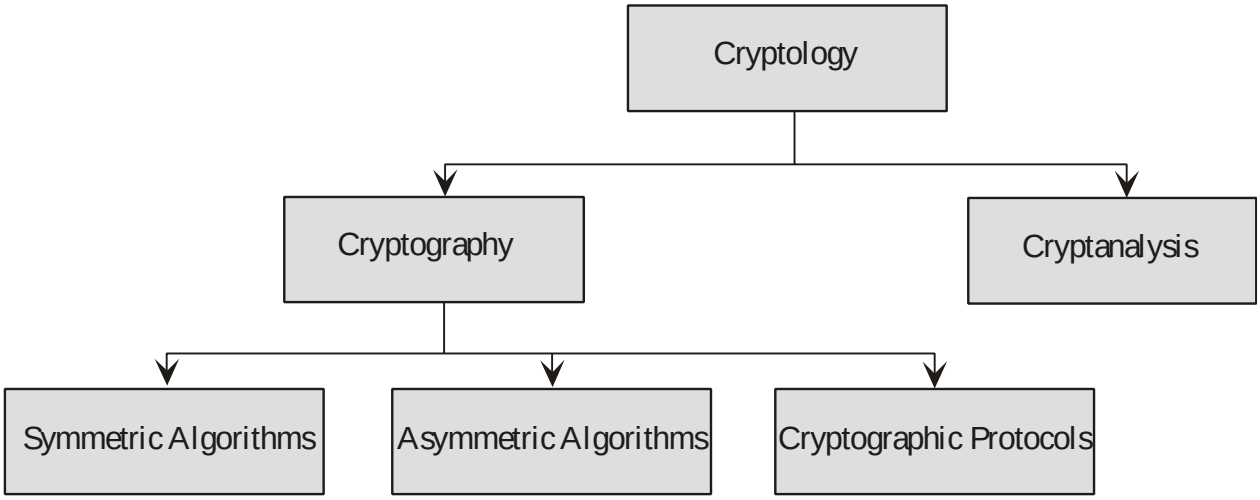

Figure 2.2: Cryptology Fields

2.2 Encryption Algorithms

In this section we recall few important fundamentals of cryptography. Cryptography can be divided into three branches: *Symmetric (Private Kay) Algorithms*, *Asymmetric (Public Key) Algorithms* and *Cryptographic Protocols.* When we have two parties performing an encryption and decryption methods using the same private key, we have symmetric encryption. When they used different keys one for the encryption

and another for the decryption, we have asymmetric encryption. Cryptographic protocols, also called crypto protocols generally deal with algorithm applications and how the algorithms should be used.

2.2.1 Symmetric Algorithms

Symmetric algorithms require that both encryption and decryption operations are performed with the same key. Until 1976, all cryptography algorithms were based on symmetric methods. Symmetric algorithms are still useful, and more efficient and popular than asymmetric algorithms. In the symmetric algorithm the sender and the receiver have to share the key (secret-key) in advance to be able to perform the encryption and the decryption procedures. The key distribution is an issue associated usually with symmetric schemes. The algorithm needs a secure channel to share the key between two parties. Hence, every party has to store many secret-keys to communicate with different parties. Also, an important fact about symmetric algorithm that is encryption and decryption algorithms are known publicly. Secrecy lies in the chosen key. Two types are known under the symmetric algorithms: first one is the *block ciphers* which encrypts a block of bits at the same time with on key, for example *Advanced Encryption Standard* (AES)[DR00, DR02]. The second type is *stream ciphers* which encrypts bits individually, for instance, One-time pad [Ver26, EJ03].

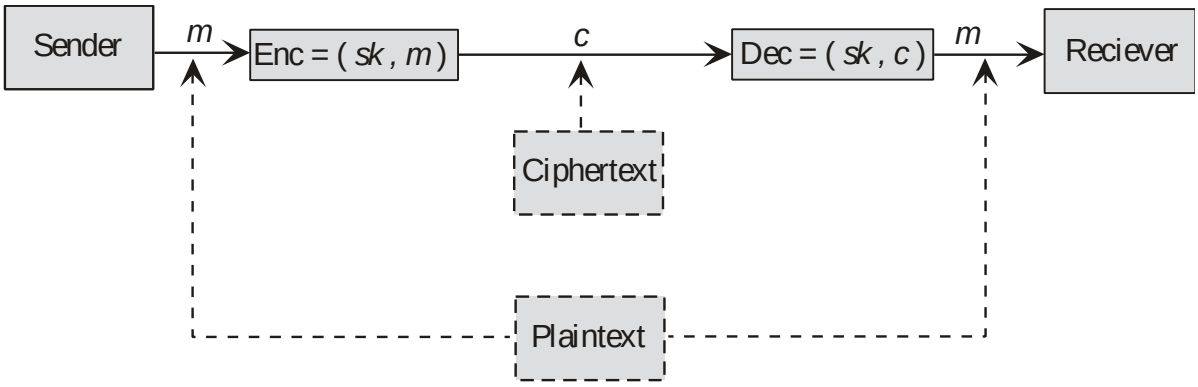

Figure 2.3: Secret Key Model

2.2.2 Asymmetric (Public Key) Algorithms

In contrast to the symmetric family, the asymmetric algorithms introduce a different encryption algorithm. In 1976 the notion of public key algorithm was described by Whitfield Diffie and Martin Hellman in their breakthrough paper [DH76]. After one year, in 1978 the first public key encryption implementation was constructed by Rivest, Shamir and Adleman (RSA) [RSA78]. In public key algorithms, a participant uses two different keys: public key for encryption and secret key for decryption. Each participant has to publish his public key if he want to receive a message, the sender will use the published key for encryption. Then, the receiver will use his secret key to decrypt the message.

Those schemes are much practical than the symmetric ones because the key distribution is very easy. There is no need for key exchange procedure in advance like symmetric algorithms. Even though, the public key and the secret key are different. The keys are linked mathematically in such away that the public key in some point could reveal some information about the secret key. Yet, an adversary who tries to recover some information about the secret key should take years to do so. However, due to the mathematical computations asymmetric encryption algorithms are less efficient than symmetric encryption algorithms. In order to understand the public key scheme, Figure 2.4 illustrates the algorithm mechanism.

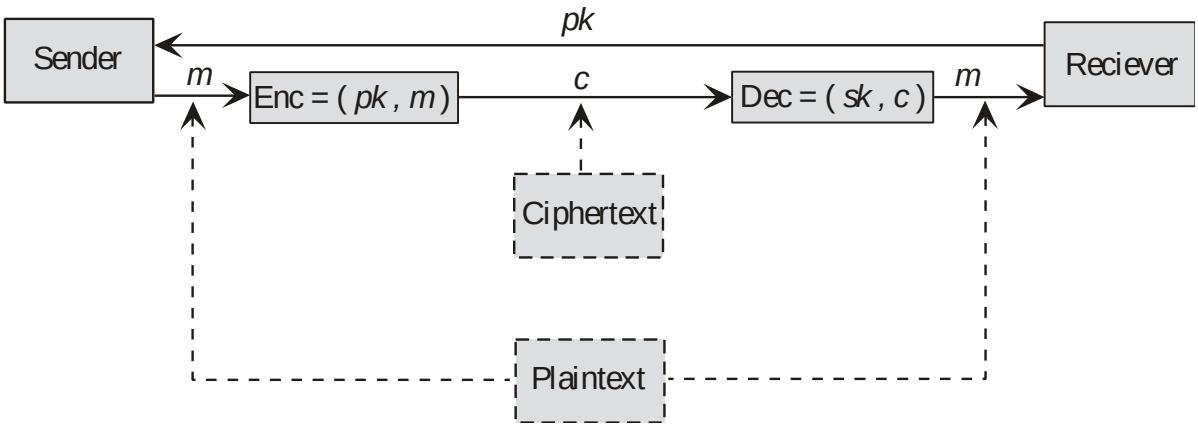

Figure 2.4: Public Key Model

A public key algorithm has three main algorithms: Key generation *(KeyGen)* algorithm, Encryption *(Enc)* algorithm, and Decryption *(Dec)* algorithm. To understand

them we recall some important cryptographic notations:

- A *message space* set is denoted by M, which is the message to be handled. An element of M is called *plaintext message* or just a *plaintext*.
- A *ciphertext space* is denoted by C, an encryption of the plaintext message. A component of C is called *ciphertext*.
- A *key space* is denoted by K, which is the range of all possible values of *key*.

Now we introduced the public key procedures:

- Key generation procedure**(KeyGen)**: Randomized algorithm, given a security parameter λ, outputs a key pair public and secret key (pk, sk) denoted by:

  $$KeyGen = (pk, sk) \in K$$

  The public key pk is known to the public, the secret key sk is kept private by the owner.

- Encryption procedure **(Enc)**: Randomized algorithm, that takes the plaintext $m \in M$ and a public key pk and outputs the ciphertext $c \in C$ denoted by:

  $$Enc(pk, m) = c$$

- Decryption procedure **(Dec)**: Deterministic algorithm, that takes the ciphertext $c \in C$ and a secret key sk and outputs the plaintext $m \in M$ denoted by:

  $$Dec(sk, c) = m$$

2.2.3 One-way function

The encryption procedure **(Enc)** implements a trapdoor One-way function. The concept of one-way function is crucial to the public key system. One-way functions are not difficult to perform but it is very hard to reverse. For example, given x it is possible to perform $f(x)$, but are hard to recover x given $f(x)$. A hard problem

is defined as a task that will take millions of years to conduct using state of the art computing resources. In the public key cryptography one-way functions can not be implemented directly. A plaintext $m \in M$ encrypted by a one-way function is not useful; because it is undecipherable.

For a public key algorithms we need a modified version of one-way functions. The trapdoor one-way functions is another version of one-way functions, that contains a secret trapdoor. Moreover, a trapdoor one-way function is simple to implement and hard to reverse, unless one has access to the secret trapdoor. That is, given the ciphertext c, where $c \in C$, it must be hard to recover the plaintext m, where $m \in M$ by computations using the public key pk and the cipher c. However, there is a secret trapdoor called secret key sk, such that given the ciphertext c and the secret key sk it is simple to recover the plaintext m.

There are three families of trapdoor one-way functions. All three types are based on mathematical problems. The first is integer factorization based, that relies on difficulty to factor large integers. The most known public key algorithm under this type is RSA [RSA78]. The second is discrete logarithm, several schemes are based on this problem. The most outstanding public key algorithms include the Digital Signature Algorithm (DSA) [FIP00], DiffieHellman(DH) key exchange [DH76], and Elgamal encryption [ElG85]. The third is elliptic curve relationships, which is a generalization of the discrete logarithm scheme. The most prominent schemes in this family include Elliptic Curve Digital Signature Algorithm (ECDSA) [JMV01], and the Elliptic Curve DiffieHellman key exchange (ECDH).

The first and second families were introduced to the public in the mid-seventies, and the last family was introduced in the mid-eighties. These schemes are well studied and secure against any types of attack if the parameters are chosen carefully. Moreover, there have been several proposed public key schemes based on variant mathematical problems, but there are some security or practicality issues. In addition, public key schemes do not require a secure initial secret key exchange between the sender and the receiver.

2.2.4 Deterministic vs. probabilistic encryptions

Most of the encryption schemes are deterministic, given a plaintext and an encryption key, the output of the encryption algorithm is always the same ciphertext. An adversary can then get some information about the plaintext after performing computation on the ciphertext. Using a deterministic algorithm scheme without the addition of some randomness makes it easy to realize if the same message is sent twice. As a result, we need a probabilistic algorithm scheme is needed in practice.

The notion of probabilistic encryption was first described by Shafi Goldwasser and Silvio Micali [GM84]. The Idea behind probabilistic encryption is to preserve the information integrity after it has been encrypted by public key algorithms. This is done by introducing randomness in the encryption algorithm. By doing so, the encryption of the same information many times will generate variant ciphertexts. An adversary who tries to get some information by performing some computation on the ciphertext will not be able to learn anything about the message. The encryption algorithm used a noise r usually as a randomness factor. As a result of probabilistic encryption schemes we need to consider something called *message expansion*. In a probabilistic scheme there exist a several possible ciphertexts for one plaintext. The number of different ciphertexts are larger than the number of all possible plaintext. Therefore, the size of the ciphertext must be absolutely longer than the size of the plaintext. The ratio between the length of the plaintext and the ciphertext is known as *message expansion*.

Chapter 3

Homomorphic Encryption

3.1 Introduction

This section will present a brief overview of the history of homomorphic encryption and basic definition and concepts related to *homomorphic encryption*.

3.1.1 Need for Homomorphic Encryption

The main purpose of encryption is maintaining the confidentiality of sensitive information. In the past, encryption was mainly implemented for military and commerce. However, encryption has been used much more widely over time. Encryption algorithms are used nowadays in various life aspects such as financial transactions, exchanging email privately, and messaging. Modern technologies such as cloud computing allow users and companies to connect, store, and share computing resources.

Cloud computing is a suitable way to store data and make use of some cloud services to manipulate the data. Due to the nature of cloud technology, the security and privacy of the data require a user trust on the cloud provider. Analyzing current implementation of cloud computing, data can be encrypted during the transfer phase. But, storing the data in an encrypted mode does not help. The cloud providers need access to userś private plaintext data to be able to respond to their requests and perform computations this might not be acceptable by many users. Moreover, the cloud providers will not be able to carry on any computations on encrypted data. Cryptography can solve many of the privacy and security issues which are related to

cloud computing.

Encryption seems to be the solution for the cloud computing issues, but there are some limitations of this technique. Any system that operates on encrypted data can, up to a certain point store, or recover the data for the user; any more advanced operations would require the decryption of the data before being operated on.

Therefore, shortly after Whitfield and Martin Hellman introduced the idea of public key cryptosystem in 1976. Cryptographers started searching for a practical public key cryptosystem implementation. In 1978 three famous cryptographers introduced the most known public key scheme, RSA, which was named after their names Ron Rivest, Adi Shamir, and Leonard Adleman [RSA78]. The scheme consists of two keys, public key and private key. The security of the scheme is based on the integer factorization problem, which is the trapdoor one-way function for the system. Multiplying two large primes is not difficult, but recovering the correct factorization from the product is very hard. However, the RSA public key algorithm is slower than the symmetric algorithm, since RSA and other asymmetric schemes require more computations to implement. In addition, the RSA scheme requires an access to the private key in order to enable the message decryption. As a result, the ciphertext is completely useless without the private key. In that case, there was a need for a scheme that can manipulates the data without decrypting it.

In 1978 Rivest, Adleman and Dertouzos [RAD78] raised an important question: can we utilize a special privacy homomorphism to encrypt the data and do an unlimited computations on it while it remains encrypted without the necessity of decrypting it? In their paper, they proposed a homomorphic encryption scheme under the name "**privacy homomorphism**" to solve this issue. Unfortunately, few years later Brickell and Yacobi in [BY88] identified some security issues related to the proposal of Rivest *et al.*

After the realization of the privacy homomorphism notion in [RAD78], numerous efforts have been made to obtain a homomorphic scheme. Cryptographers initially developed several schemes that can perform addition or multiplication on ciphertexts bot not both. The first semanticlly secure homomorphic encryption scheme is Goldwasser and Micali [GM84]. A various encryption schemes that either additively or multiplicatively homomorphic were proposed later. For example, in [ElG85] Taher

Elgamal proposed an encryption system that support homomorphic multiplication operations. Later, an encryption scheme that allows homomorphic addition operations was invented by Paillier [Pai99]. In addition, many others homomorphic encryption schemes were introduced [DJN10, AD97, Reg04, Reg09, CF85, NS98, OU98]. Other encryption systems were developed, that enabled both additions and multiplications operations, but for limited number of operations [BGN05, GHV10, MGH10, SYY99]. A lot of such homomorphic schemes are already quite useful in different applications.

3.1.2 Definition

Homomorphic encryption allows arbitrary number of operations on information without the requirement of decryption functions. A homomorphic encryption scheme is a quadruple of four algorithms: *KeyGen*, *Enc*, *Dec*, plus an additional algorithm called Evaluation *Eval*.

Let (*M, C, K, Enc, Dec, Eval*) be an encryption scheme where **M** is the plaintext space, **C** is the ciphertext space, **K** is the key space, **Enc** is the encryption algorithm, **Dec** is the decryption algorithm and **Eval** is the evaluation algorithm.

M is the plaintext space has two operations. Addition is the first operation denoted by $+$. The second operation is multiplication denoted by $\times$.

C is the ciphertext space has two operations. The first one is addition denoted by $\oplus$. The second operation is multiplication denoted by $\otimes$.

Enc is the encryption algorithm, which is a map from M to C, *i.e.*, $E_k : M \to C$, where $k \in K$.

Dec is the decryption algorithm, which is a map from C back to M. *i.e.*, $D_k : C \to M$, where $k \in K$.

Eval is the evaluation algorithm which takes a key $k \in K$, a function f, and a set of ciphertexts $c_1, \cdots, c_\ell$, apply both addition and multiplication operations of f on the ciphertexts and outputs c_f.

$$Eval(k, f, c_1, \cdots, c_\ell) \to c_f$$

For all $a, b \in M$, ciphertexts $c_f \in c$ and $k \in K$, an encryption scheme is said to

be *homomorphic* under addition and multiplication, if

$$Dec(c_a \oplus c_b) = a + b$$

and

$$Dec(c_a \otimes c_b) = a \times b$$

The semantic security of homomorphic encryption defines by two properties *circuit-privacy* and *compactness*. Circuit-privacy is when an adversary can not obtains any information about the evaluation operations from the ciphertext generated by evaluate algorithm. Compactness that is the ciphertext size generated by evaluate does not depend on the complexity of circuit C.

3.2 Recent Developments

In this section we will review the latest fully homomorphic encryption schemes. Also a general idea about the research directions on the homomorphic encryption will be covered.

3.2.1 Fully Homomorphic Encryption Based on Ideal Lattices

In 2009 Craig Gentry introduced the first fully homomorphic encryption scheme on his Stanford PhD thesis [Gen09]. Gentry's FHE construction supports both addition and multiplication without any limitations on the numbers of operations that can be performed. The security of Gentry's scheme is based on two assumed hardness problems: certain worst-case problems over ideal lattices and (average case) sparse subset-sum problem. In his thesis Gentry described a blueprint for constructing the fully homomorphic encryption scheme. This blueprint consists of three steps. First, Gentry outlines a somewhat homomorphic scheme that supports limited numbers of operations and does not support high-polynomial degree functions. This limitation is due to the noise associated with each bit ciphertext. Each homomorphic function performed on a ciphertext increases the noise component of the ciphertext. If the

noise exceeds a certain limit the outcome ciphertext does not decrypt correctly.

Second, Gentry describes the decryption circuit *squash* procedure that relies on the hardness assumption of the (average case) sparse subset-sum problem. This problem is used to reduce the degree of the decryption polynomial. The sparse subset-sum problem was considered to be the main limitation of Gentry scheme. This is because it is not a well-studied cryptographic problem.

Lastly, Gentry main idea, *"bootstrapping"* algorithm to achieve a fully homomorphic scheme. The goal of this step is to reduce the noise on the ciphertext, so it can be used in more additions and multiplications operations. Therefore, it is called "ciphertext refresh" procedure. For more details ones can refers to the original work [Gen09].

Since Gentry's breakthrough result, three main families of fully homomorphic encryption schemes were proposed:

1. The first category follows Gentry's original scheme [Gen09], which is based on ideal lattices. Smart and Vercauteren presented the first attempt to implement Gentry's scheme [SV10]. However, they could not achieve a bootstrappable scheme. In 2011 Craig Gentry and Shai Halevi announced the first implementation of Gentry's FHE scheme [GH11b]. The implementation contains many optimizations and some ideas from Smart and Vercauteren first attempt. They reported for their highest secure level, where $\lambda = 72$ bit, the public key size is 2.3 GB and the ciphertext refresh technique required 30 minutes. Moreover, Gentry and Halevi described a new approach for constructing fully homomorphic schemes [GH11a]. The scheme eliminates the squashing procedure and replaces it with the Decision Diffie-Hellman from Elgamal scheme.

2. Second classification is based on the Learning with Errors (LWE) and Ring Learning with Errors (RLWE) problems. Brakerski and Vaikuntanathans proposed the first two schemes in this area [BV11a][BV11b]. Several contributions and modifications have been made to improve the scheme include the scale-free variant of Brakerski [Bra12] and the NTRU encryption [LATV12]. Moreover, a fully homomorphic encryption scheme with better bootstrapping was introduced by Gentry, Halevi, and Smart [GHS12a]. A fully homomorphic encryp-

tion without Gentry's bootstrapping procedure and supports an encryption of vector of bits was proposed in [BGV12]. Many schemes with batch capability are introduced in [GHS12b, BGH13]. An implementation is described in [GHS12c] based on [BGV12, GHS12b] schemes. More effort on optimizing the Ring-learning with error scheme is presented [BGH13]

3. Last branch is based on fully homomorphic encryption over the integers scheme. The first scheme was proposed by van Dijk, Craig Gentry, Shai Halevi and Vinod Vaikuntanathan (DGHV) [VDGHV10]. Several improvements have been made, describing new optimizations in order to increase the efficiency and reduce the public key size of the scheme [CMNT11] [CNT12]. A batching technique was proposed recently to enhance the performance level of integer based encryption schemes [CLT13, CCK$^+$13, KLYC13]. Furthermore, Coron, Lepoint, and Tibouchi, introduced a scale-free fully homomorphic encryption over the integers scheme [CLT14] following Brakerski work in [Bra12].

This thesis mainly focus on the somewhat homomorphic encryption over the integers schemes (**DGHV**).

3.2.2 Fully Homomorphic based on integers

This section will review the recent numerous works on the fully homomorphic encryption over the integers schemes.

Fully Homomorphic Encryption over the integers (DGHV)

At Eurocrypto 2010 van Dijk, Craig Gentry, Shai Halevi and Vinod Vaikuntanathan introduced the first Fully Homomorphic scheme over the integers [VDGHV10]. *DGHV* is the simplest possible FHE scheme, using only elementary modular arithmetic. The scheme follows Gentry's blueprint to achieve a fully homomorphic encryption scheme. They started with the first step in Gentry Scheme which is the somewhat homomorphic encryption scheme supporting a limited number of arithmetic operations namely addition and multiplication over encrypted bits. DGHV demonstrates that FHE can be attained without the need for the complexity of ideal lattices. The scheme's

security is based on the approximate integer greatest common divisors (Approximate-GCD)($AGCD$) problem which is constructed by Howgrave-Graham [HG01].

The AGCD problem gives many approximation of multiples of an integer, compute the greatest common deviser. Therefore to secure the scheme against AGCD known attacks, the DGHV parameters selection results in a public-key size of $\tilde{\mathcal{O}}(\lambda^{10}) \approx 25\ GB$, where λ is the security parameter, which is very large to be practical. In order to reduce the public-key size, Coron, Mandal, Naccache and Tibouchi [CMNT11] have described a FHE over the Integers with Shorter Public Keys.

Fully Homomorphic Encryption over the Integers with Shorter Public Keys

Coron, Mandal, Naccache and Tibouchi [CMNT11] introduced the first attempts toward making the DGHV scheme practical. The idea of reducing the public key size is implemented in two steps. First, storing only a small portion of the public key. Second, generating the full public key on the fly. However, determining a secure set of concrete parameters was challenging. The implementation obtains similar performances as of Gentry and Halevi [GH11b]. The security of the scheme is under the (stronger) error-free approximate GCD problem. In their modified version of the somewhat homomorphic encryption they were able to reduce the public-key size from $\tilde{\mathcal{O}}(\lambda^{10})$down to $\tilde{\mathcal{O}}(\lambda^{7}) \approx 1\ GB$.

Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers

Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi described an optimization method to reduce the public key size of Van Dijk *et al.* (DGHV) schem [CNT12]. Their technique consists of three main steps. First generate a prime integer p (the secret key). Then, using a pseudo-random number generator f and a public random seed to generate a set of integers. Finally, apply variant computations on the integer set to get smaller public key elements. The scheme reduces the public key size to $\tilde{\mathcal{O}}(\lambda^{5})$ rather than $\tilde{\mathcal{O}}(\lambda^{7})$ as in [CMNT11]. The scheme obtains a public key size of $10.1MB$. The paper also described a new module switching technique for DGHV

scheme. This method produced a fully homomorphic scheme without Gentry's bootstrapping procedure.

Batch Fully Homomorphic Encryption over the Integers

Another step towards a fully homomorphic encryption scheme was introduced by Jean-Sébastien Coron, Tancréde Lepoint, and Mehdi Tibouchi[CLT13]. This paper proposed an extension of the fully homomorphic encryption over the integers scheme proposed by Van Dijk *et al.* [VDGHV10] to batch fully homomorphic encryption. They presented a scheme that encrypts a vector of plaintext bits as a single ciphertext. The implementation remains secure under the error free approximate-GCD problem. The scheme used the Chines Reminder Theorem (CRT) to encrypt a multiple bits m_i into a single ciphertext c. Also in the scheme, instead of using a single prime number p the scheme employed a tuple of coprime integers $p_0, \ldots, p_{\ell-1}$. Applying the compression technique from [CNT12] yield a public key of size $\tilde{\mathcal{O}}(\lambda^7)$ instead of $\tilde{\mathcal{O}}(\lambda^5)$.

Chapter 4

From RSA to Somewhat Homomorphic Encryption Algorithms

Before we introduce the encryption algorithms, this section provides an overview of some basic mathematical concepts that will be used later in the encryption schemes.

4.1 Essential Elementary Number Theory for Public Key Algorithm

Below we recall several number theory techniques which are important for public key algorithms. All theories are briefly described, for more details readers may refer to [PPP09, Sch07].

4.1.1 Notation

We denote the set of integers by $\mathbb{Z}$. The notation $a \leftarrow S$ indicates the action of choosing randomly an element a independently and uniformly from a set S. When $\mathcal{D}$ is a distribution, $a \leftarrow \mathcal{D}$ denote the action of returning an element a according to the distribution $\mathcal{D}$.

Theorem 4.1 (Division with remainder property)**.** Let a and b be positive integers.

Then we say a divided by b, where $a > b$, has a unique q (the quotient) and r (the remainder) such that

$$a = bq + r, \quad \text{and} \qquad 0 \leq r < b \tag{4.1}$$

4.1.2 Floors and Ceilings

Let a be a real number. Then we define three operations:

- Floor function denoted $m = \lfloor a \rfloor$ to be the greatest integer such that $m \leq a < m + 1$.
- Ceiling function denoted $m = \lceil a \rceil$ to be the smallest integer such that $m - 1 < a \leq m$.
- Round function denoted $m = \lfloor a \rceil$ to be the closest integer to a.

4.1.3 Greatest Common Divisor(GCD)

For all $a, b \in \mathbb{Z}$, there exists a unique greatest common divisor d, denoted $d = gcd(a, b)$, which is the largest positive integer that divide both a and b. The greatest common divisor exists if both integers a and $b \neq 0$.

For $a, b \in \mathbb{Z}$, if the $gcd(a, b) = 1$, then a and b are relatively coprime. In that case, the only common divisor of a and b is 1.

4.1.4 Euclidean Algorithm

Euclidean algorithm is a method to compute the greatest common divisor of two number efficiently such that

$$gcd(a_0, b_1) = gcd(a_0 - b_1, b_1), \quad \text{where} \qquad a_0 > b_1 \tag{4.2}$$

4.1.5 Euler's Totient or Phi Function

For a positive integer $n \geq 1$, define a function that finds the number of positive integers less than n and relatively prime to n.

Definition 4.1 (Euler's Totient or Phi Function)**.** Given a positive integer n, the number of primes less than n is denoted by $\phi(n)$.

Fact 4.1 (Euler's Totient function properties)**.**

(i) $\phi(p) = p - 1$, if p is a prime.

(ii) Euler's Totient function is multiplicative, that is if $gcd(a, b) = 1$, then $\phi(ab) = \phi(a) \times \phi(b)$.

Now we define some theorems related to the ϕ function and are useful for public key cryptography. The *Euler's Totient or Phi Function* is not efficient and extremly slow if the integers are huge. For public key algorithms we need a faster method to compute the huge number. Therefore, we can use the following theorem to do so if we know the factorization of n.

Theorem 4.2. Assume n be a positive integer

$$n = p_1^{e_1}, p_2^{e_2}, \ldots, p_m^{e_m} \qquad p_i < n \tag{4.3}$$

where $p_1, p_2, \cdots, p_i$ are the prime factorization of n, and e_i are positive integers. Then

$$\phi(n) = \prod_{i=1}^{m} (p_i^{e_i} - p_i^{e_{i-1}}) \tag{4.4}$$

4.1.6 Fermat's Little Theorem and Euler's Theorem

Now we describes two important theorems for public key algorithms. The first one is Fermat's Little Theorem, which is useful for primality testing and other features of public key algorithms.

Theorem 4.3 (Fermat's Little Theorem)**.** Let a be a positive integer and p a prime with $gcd(a, p) = 1$, then

$$a^{p-1} \equiv 1 \pmod{p} \tag{4.5}$$

A generalization of Fermat's Little Theorem is Euler's Theorem, where it can be a mod to any integer.

Theorem 4.4 (Euler's Theorem)**.** Let a and n be positive integers with $gcd(a, n) = 1$, then

$$a^{\phi(n)} \equiv 1 \pmod{n} \tag{4.6}$$

4.1.7 Chinese Reminder Theorem

The Chinese reminder theorem was published by the famous Chinese mathematician *Sun Tzu* in the fourth century [BEAS09]. The Chinese reminder theorem is used to solve systems of linear congruences. If we have a system of k congruences in which the moduli are pairwise relatively prime, there is a solution and the solution is unique modulo the product of the pairwise relatively prime[Ros93].

Theorem 4.5. The basic idea is the Chinese remainder theorem will determine a value x that when divided by given pairwise coprime divisors leaves given remainders. The problem can be stated as follows. Let $m_1, m_2, \ldots, m_k$ be pairwise coprime integers with $gcd(m_i, m_j) = 1$ whenever $i \neq j$. Let M be the product of primes $M = m_1 m_2 \cdots m_k$. Let $a_1, a_2, \ldots, a_k$ be any integers, where $k = 1, 2, 3, \ldots, N$. There is an integer x solving the following system of simultaneous congruences [KS92].

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_k \pmod{m_k} \end{aligned}$$

Then there exists exactly one unique integer x satisfying this system.

Algorithm [Ros93] The solution to the system of equations may be obtained by the following algorithm.

Algorithm 4.1 Chinese Remainder Theorem x

Require: Input: Relatively coprime integers $m_1, m_2, \ldots, m_k$ and the reminder integers $a_1, a_2, \ldots, a_k$ {we can solve the system of equations as follows}

1: let $M = \prod_{k=1}^{n} m_k$
2: **for** $i = 1$ to k **do**
3: $z_i = M/m_i = m_1 \times m_2 \cdots m_{i-1} \times m_{i+1} \cdots m_k$
4: **end for**
5: **for** $i = 1$ to k **do**
6: determine the multiplicative inverse y_i which is given by $z_i y_i \equiv 1 \pmod{m_i}$
7: **end for**
8: $x = a_1 y_1 z_1 + \cdots + a_k y_k z_k \pmod{M}$
9: **RETURN** (x)

Example:

Solve the following system of simultaneous congruences:

$$x \equiv 1 \pmod{3}$$
$$x \equiv 2 \pmod{5}$$
$$x \equiv 3 \pmod{7}$$

Solution:

Step 1: Establish the basic notation. We have $k = 3$, $a_1 = 1, a_2 = 2, a_3 = 3, m_1 = 3, m_2 = 5, m_3 = 7$, and $M = 3 \times 5 \times 7 = 105$.

Step 2: $z_1 = M/m_1 = \frac{\cancel{3}\times 5\times 7}{\cancel{3}} = 35, z_2 = M/m_2 = \frac{3\times\cancel{5}\times 7}{\cancel{5}} = 21, z_3 = M/m_3 = \frac{3\times 5\times\cancel{7}}{\cancel{7}} = 15$.

Step 3: Solve $z_i y_i \equiv 1 \pmod{m_i}, i = 1, 2, 3$.

$$35y_1 \equiv 1 \pmod{3}$$
$$21y_2 \equiv 1 \pmod{5}$$
$$15y_3 \equiv 1 \pmod{7}$$

By trail and error we have $y_1 = 2, y_2 = 1$ and $y_3 = 1$.

Step 4: Substituting these numbers into:

$$x = a_1 y_1 z_1 + \cdots + a_k y_k z_k \pmod{M}.$$

$x = (1 \times 2 \times 35) + (2 \times 1 \times 21) + (3 \times 1 \times 15) \pmod{M} = 157 \pmod{105}$
$x = 52.$

4.2 RSA Encryption

4.2.1 Introduction to RSA

RSA encryption is based on the integer factorization problem. The RSA algorithm actually consists of three algorithms: key generation, encryption algorithm, and decryption algorithm. RSA algorithm has a public key and a secret key. The public key is known by everyone and is used to perform the encryption algorithm. The secret key is used to recover the message by using the decryption algorithm and it is only known by its legitimate owner.

4.2.2 The Construction

RSA.KeyGen: generate the two keys as follows:

1. Choose two random large primes p and q. For high security level, the integers p and q must be chosen of equal bit-length.

2. Compute the product:
$$n = p \cdot q. \tag{4.7}$$

3. Compute
$$\phi(n) = \phi(p)\phi(q) = (p-1)(q-1). \tag{4.8}$$
Where ϕ is Euler's totient function.

4. Randomly choose an integer e, such that $e \in \{1, 2, \cdots, \phi(n) - 1\}$ and
$$gcd(e, \phi(n)) = 1. \tag{4.9}$$
In other words, e and $\phi(n)$ are pairwise relatively prime. This condition also ensures the existence of the multiplicative inverse of e.

5. use the extended Euclidean algorithm to compute the secret key d such that

$$e \cdot d \equiv 1 \pmod{\phi(n)} \tag{4.10}$$

in other words,

$$d = e^{-1} \pmod{\phi(n)} \tag{4.11}$$

Note that d and n are also relatively prime. d is the multiplicative inverse of e.

Remark 4.1. The resulting numbers n and e are the public key components, the integer e is sometimes called encryption(or public) exponent.

Remark 4.2. The resulted value d is the private key, and sometimes called encryption(or private) exponent.

Remark 4.3. The first two primes p and q must be kept secret.

RSA.Encrypt: To encrypt a message m, first divide m into blocks such that $0 < m < n$. After that using the public key (n, e) as follows:

$$c = m^e \pmod{n} \tag{4.12}$$

RSA.Decrypt: Using the private key d to recover the message m from c.

$$m = c^d \pmod{n} \tag{4.13}$$

4.3 The Somewhat Homomorphic DGHV Scheme Over the Integers

The simple concept of somewhat homomorphic encryption was described by van Dijk, Gentry, Halevi and Vaikuntanathans [VDGHV10]. Numerous efforts have been made to improve the efficiency of the scheme [CMNT11]. In this section, we recall the Dijk's somewhat homomorphic encryption algorithms as constructed in [VDGHV10].

4.3.1 The Parameters

The scheme has many parameters, ensuring the number of integers in the public key does not exceed a certain threshold and the bit-length of different integers. The parameters are:

λ the security parameter.

τ the number of integers in the public key $x_{1 \leqslant i \leqslant \tau}$.

γ the sum of bit-length of integers in the public key $x_{1 \leqslant i \leqslant \tau}$.

η the bit-length of the secret key p.

ρ the bit-length of the noise in the public key r_i.

$\rho^{'}$ the secondary noise used for encryption.

The concrete parameters of the DGHV scheme must satisfy some constraints:

1. $\rho = \omega(\log \lambda)$, to be secure against brute-force attacks on the noise,
2. $\eta \geq \rho\Theta(\lambda \log^2 \lambda)$, to permit High-order polynomial multiplication,
3. $\gamma = \omega(\eta^2 \log \lambda)$, to be secure against various lattice-based attacks,
4. $\tau \geq \gamma + \omega(\log \lambda)$, in order to use left-over hash lemma [VDGHV10] in the security proof.

The scheme chooses a convenient parameter set which is:

$$\tau = \gamma + \lambda \qquad \gamma = \tilde{\mathcal{O}}(\lambda^5) \qquad \eta = \tilde{\mathcal{O}}(\lambda^2) \qquad \rho = \lambda \qquad \rho^{'} = 2\lambda \tag{4.14}$$

The scheme's complexity is $\tilde{\mathcal{O}}(\lambda^{10})$.

For a specific η-bit odd integer p, we use the following distribution over γ-bit integers:

$$\mathcal{D}_{\gamma,\rho}(p) = \{\text{Choose} q \leftarrow \mathbb{Z} \cap [0, 2^{\gamma}/p), r \leftarrow \mathbb{Z} \cap (-2^{\rho}, 2^{\rho}) \text{Output} x = q \cdot p + r\}. \tag{4.15}$$

4.3.2 The Construction

DGHV.KeyGen(λ): Generate a random odd integer p of η-bit size:

$$p \leftarrow (2\mathbb{Z}+1) \cap [2^{\eta-1}, 2^{\eta}) \tag{4.16}$$

For the public-key, sample

$$x_i \leftarrow \mathcal{D}_{\gamma,\rho}(p) \quad \text{for} \qquad 0 \leqslant i \leqslant \tau \tag{4.17}$$

Relabel so x_0 is the largest among all samples x_i's. Restart unless x_0 is odd and the reminder of $x_0 \pmod p$ is even. The public key and the secret key are as follow: The public key is the collection of x_i's as obtained from equation 4.17.

$$pk = (x_0, x_1, \ldots, x_\tau) \tag{4.18}$$

The secret key is an odd integer as obtained from equation 4.16

$$sk = p \tag{4.19}$$

DGHV.Encrypt(pk,$m \in \{0,1\}$**):** Choose a random subset $S \subseteq \{1, 2, \ldots, \tau\}$ and a random integer r in $(-2^{\rho'}, 2^{\rho'})$, and output the ciphertext:

$$c = \left[m + 2r + 2\sum_{i \in S} x_i \right]_{x_0} \tag{4.20}$$

DGHV.Decrypt(sk,c**)**: Output

$$m = [c \pmod p] \pmod 2 \tag{4.21}$$

DGHV.Evaluate(pk, $C, c_1, \ldots, c_t$**):** Takes as input public key, given the circuit C with t ciphertexts c_i, apply both addition and multiplication gates of C to the ciphertexts, by performing all the additions and multiplications over the integers, and return the resulting ciphertexts.

Hence the DGHV scheme is somewhat homomorphic encryption that permitted limited number of homomorphic operations to be performed. Given two ciphertext:

$$c_1 = p \cdot q_1 + 2r_1 + m_1 \tag{4.22}$$

and

$$c_2 = p \cdot q_2 + 2r_2 + m_2 \tag{4.23}$$

Where p was obtained from 4.16, r_i and q_i was obtained from 4.15.

The addition operation is defined as:

$$c_1 + c_2 = (q_1 + q_2) \times p + 2(r_1 + r_2) + (m_1 + m_2) \tag{4.24}$$

The following from equations 4.24, the noise pound after addition is given by:

$$r_1 + r_2 < p/2 \tag{4.25}$$

The multiplication operation is defined as:

$$\begin{aligned} c_1 \times c_2 = p(pq_1q_2 + q_1r_1 + q_1m_2 + q_2r_1 + q_2m_1) \\ +2(2r_1r_2 + r_1m_2 + r_2m_1) + m_1m_2 \end{aligned} \tag{4.26}$$

The following from equations 4.26, the noise pound after multiplication is given by:

$$2r_1r_2 + r_1m_2 + r_2m_1 < p/2 \tag{4.27}$$

The resulting ciphertext $c_1 + c_2$ is an encryption of $m_1 + m_2$ with noise of size $= (\rho^{'} + 1)$-bit integers, and the ciphertext $c_1 \cdot c_2$ is an encryption of $m_1 \cdot m_2$ with noise of size $\simeq 2\rho'$-bits. We have

$$m_1 + m_2 = (c_1 + c_2 \pmod p) \pmod 2 \tag{4.28}$$

$$m_1 \times m_2 = (c_1 \times c_2 \pmod{p}) \pmod{2} \tag{4.29}$$

As a result, the DGHV is somewhat homomorphic scheme. The noise size on the ciphertext must remains under the size of the bit number of $sk = p$ for correct decryption. The scheme permits η/ρ' of homomorphic operations on ciphertexts.

4.3.3 Semantic Security

In DGHV somewhat homomorphic scheme is secure based on the approximate-GCD problem [VDGHV10]. Given a set of randomly chosen integers $(x_0, x_1, \ldots, x_\tau)$, all are near multiple of p, find this "common near divisor" p. The definition of the scheme security will be defines in 4.2 on 31.

The public key encryption is done by masking the message m_i with a subset sum randomly chosen form the public key elements $x_i = q_i \cdot p + r_i$. The scheme semantic security is proved by applying the leftover Hash lemma [VDGHV10] on the subset sum modulo x_0, after that using the $2r$ for more randomization on the ciphertext modulo p.

4.4 The Approximate-GCD Problems

The security of the somewhat homomorphic encryption over the integers and its variants is based on the *approximate greatest common divisor* problem. The approximate-GCD problem was defined in 2001 by Howgrave-Graham [HG01]. Generally speaking, the problem gives only approximations x' and y', compute the greatest common deviser (GCD) of two integers x and y. There are two kinds of approximate-GCD problem. The first one is the *general approximate-GCD* problem and the second one is the *partially approximate-GCD* problem.

The general approximate-GCD problem was used first in [VDGHV10], given many approximations more than two. This problem becomes the hardness assumption of the simple somewhat homomorphic encryption scheme introduced in 4.3.

Several works following the DGHV scheme used the approximate-GCD problem and different versions include [CMNT11, CNT12, CLT13, CCK$^+$13, KLYC13, CLT14]

Now we provides a formal definitions of the general approximate-GCD problem and the partial approximate-GCD problem.

Definition 4.2 (General Approximate GCD)**.** The (ρ, η, γ)-Computationally approximate-GCD problem: for a randomly chosen η-bit odd integer p, given polynomially many samples from $\mathcal{D}_{\gamma,\rho}(p)$, output p.

Definition 4.3 (Partially Approximate GCD)**.** The (ρ, η, γ)-Computationally approximate-GCD problem: for a randomly chosen η-bit odd integer p, given polynomially many samples from $\mathcal{D}_{\gamma,\rho}(p)$ and a γ-bit integer $x_0 = pq_0$, output p.

Howgrave-Graham [HG01] introduces two types of attacks to examines the hardness of the approximate-GCD problem. First attack is *continued fraction approach*, and the other is *a lattice based approach.*

4.4.1 Error-Free Variant of the Computational Approximate GCD Problem

The first variant of the Approximate-GCD problem is the error-free Approximate-GCD problem. The problem consists of working with one multiple of p, which is an error-free element. In other words, the element is without noise and known publicly namely $x_0 = q_0 \cdot p$. Although this extra information makes the error-free Approximate-GCD problem easy to compute than the AGCD-problem. The complexity of the error-free AGCD problem remains exponential in the size of ρ, where ρ is the noise. There are several known methods to factor x_o, General Number Field Sieve [LLJMP90] and the Elliptic Curve Method [LJ87]. Note that, for the following somewhat homomorphic encryption schemes, parameter selection must be sufficient so that the factorization of x_0 is untraceable.

Below we defines the Error-Free Approximate-GCD problem.

Definition 4.4 (Error-Free Approximate GCD problem)**.** The (ρ, η, γ)-Computational Error-Free Approximate-GCD **(EF-AGCD)** problem is, for a randomly chosen η-bit odd integer p, and a uniformly chosen $q_0 \in [0, 2^\gamma/p)$, given polynomially many samples from $\mathcal{D}_\rho(p, q_0)$ and a γ-bit integers $x_0 = p \cdot q_0$, output p.

4.4.2 Decisional Error-Free Variant of the Computational Approximate GCD Problem

The decisional Error-Free Approximate-GCD problem is the base of the security of the batch somewhat homomorphic encryption over the integers. The modified problem says that, given a distribution $\mathcal{D} = \mathcal{D}_\rho(p, q_0)$ and some integer z, it is a hard problem to find out whether z is chosen from $\mathcal{D}$ or not. The problem remains secure against known attacks and the parameters selection must satisfied that.

Definition 4.5 (Decisional Error-Free Approximate GCD problem)**.** The (ρ, η, γ)-decisional Error-Free Approximate-GCD **(DEF-AGCD)** problem:

- For a randomly chosen η-bit odd integer p and a uniformly chosen $q_0 \in [0, 2^\gamma/p)$.
- Given polynomially many samples from $\mathcal{D} := \mathcal{D}_\rho(p, q_0)$, and a γ-bit integers $x_0 = p \cdot q_0$.

Determine $b \in \{0, 1\}$ from $z = [x + r \cdot b]_{x_0}$ where $x \leftarrow \mathcal{D}$ and $r \leftarrow \mathbb{Z} \cap [0, x_0)$.

4.4.3 ℓ-Decisional Approximate-GCD$_Q$ Problem

Now we extend the Error-Free Approximate-GCD problem. The extension provides an *DEF-AGCD* problem for several primes rather than one as of the previous versions. The new assumption called ℓ-*DEF-AGCD*$_Q$. This new approach will be useful to prove the security of the batch somewhat homomorphic encryption over the integers scheme in section 4.5. Moreover, if the parameters were chosen properly the scheme remains secure. Below we define the new assumption.

Definition 4.6 (ℓ-Decisional Error-Free Approximate GCD$_Q$ Problem: ℓ-DEF-AGCD$_Q$)**.** The $(\rho, \eta, \gamma, \ell_Q)$-$\ell$-decisional Error-Free Approximate-GCD$_Q$ (ℓ**-DEF-AGCD**$_Q$**)** problem:

- For a randomly chosen η-bit ℓ coprime integers $p_0, \ldots, p_{\ell-1}$, a ℓ_Q-bit integers $Q_0, \ldots, Q_{\ell-1}$, and a uniformly chosen coprime $q_0 \in [0, 2^\gamma / \prod_{i=0}^{\ell-1} p_i)$, where $\prod_{i=0}^{\ell-1} p_i = p_0 \times \cdots \times p_{\ell-1}$.

- Given polynomially many samples from $\mathcal{D} := \mathcal{D}_\rho((p_i)^\ell; (Q_i)^\ell; q_0)$, a set of X consisting of ℓ integers x'_i, and a γ-bit integers $x_0 = q_0 \cdot \prod_{i=0}^{\ell-1} p_i$.

Determine $b \in \{0, 1\}$ from $z = [x + r \cdot b]_{x_0}$ where $x \leftarrow \mathcal{D}$ and $r \leftarrow \mathbb{Z} \cap [0, x_0)$.

4.5 Batch Somewhat Homomorphic Encryption Over the Integers

Below we described the scheme that has been implemented in our simulation [CCK+13]. The scheme described an extended version of the fully homomorphic encryption DGHV scheme [VDGHV10]. Using the Chinese Reminder Theorem, the extended scheme can encrypt a vector of plaintexts of size ℓ, $m_0, \ldots, m_{\ell-1}$ into one ciphertext. For the somewhat homomorphic encryption the setting allows an encryption of a vector elements not only bit. The message space, given a public key element $Q_0, \ldots, Q_{\ell-1} = 2$, We can encrypt $m_0, \ldots, m_{\ell-1} \in \{0, 1\}$ into a single ciphertext as follow:

$$c = \mathrm{CRT}_{q_0, p_0, \ldots, p_{\ell-1}}(q, Q_0 r_0 + m_0, \ldots, Q_{\ell-1} r_{\ell-1} + m_{\ell-1}) \tag{4.30}$$

Where q is uniformly and randomly modulo q_0 and r_i's is a small noises. The decryption function can be performed such that

$$m_i = [c \pmod{p_i}] \pmod{Q_i} \tag{4.31}$$

Then homomorphic computations can be done on the resulting ciphertexts including addition and multiplication. The scheme is a generalized version of DGHV scheme, but with larger plaintext. In order to allow a homomorphic encryption as of the original scheme, this can only be done when $Q_0 = \cdots = Q_{\ell-1} = 2$.

The scheme introduced extra elements to the public key, in order to allow public key encryption. The public key elements includes integers such x_i and x'_i.

The first public key element is defined as:

$$x_i (\mathrm{mod} p_j) = Q_j r_{i,j} \tag{4.32}$$

The second public key element is defined as:

$$x'_i(\mathrm{mod} p_j) = Q_j r'_{i,j} + \delta_{i,j} \tag{4.33}$$

for all i, j and $\delta_{i,j}$ is Kronecker delta.

Therefore, to encrypt a vector of plaintexts m, the encryption equation becomes as follows:

$$c = \left[\sum_{i=0}^{\ell-1} m_i \cdot x'_i + \sum_{i \in S} x_i \right]_{x_0} \tag{4.34}$$

In order to simplify the scheme construction, the extra noise $2r$ in equation (4.20), of the DGHV scheme is not used. This has been done to facilitate the security proof. Since adding the same random term $2r$ will break the security proof, the scheme used another subset-sum elements to prove the security of the scheme.

4.5.1 The Parameters

Below we recall some description about the parameters.
λ the security Parameter.
τ the number of elements in the public key.
γ the bit-length of Integers in the public key.
η the bit-length of the secret key.
ρ the bit-length of the noise in the public key.
ℓ The number of distinct secret primes.
ℓ_Q the bit-length of the $Q_0, \ldots, Q_{\ell-1}$
Also ℓ specifies the message space size. The scheme concrete parameters should be as follows:

1. $\rho = \tilde{\mathcal{O}}(\lambda)$, to be secure against Chen and Nguyen's attack [CN12], and Howgrave-Graham's attack [HG01],

2. $\eta = \tilde{\Omega}(\lambda^2 + \rho \cdot \lambda)$, to resist the factoring attack using the elliptic curve method [LJ87], and to permit high-order polynomial multiplication,

3. $\gamma = \eta^2\omega(\log\lambda)$, to resist Cohn and Heninger's attack [CH11], and the attack using Lagarias algorithm [Lag85] on the approximate GCD problem,

4. $\tau = \gamma + \omega(\log\lambda)$, in order to use left-over hash lemma in the security proof.

The scheme chooses a similar convenient parameter as of the DGHV scheme [VDGHV10] that is

$$\gamma = \tilde{\mathcal{O}}(\lambda^5) \qquad \eta = \tilde{\mathcal{O}}(\lambda^2) \qquad \rho = 2\lambda \qquad \tau = \gamma + \lambda \tag{4.35}$$

4.5.2 The Construction

BDGHV.KeyGen($\lambda, \rho, \eta, \gamma, \tau, \ell, \ell_Q$): Generate the key pair as follow:

1. Choose η-bit pairwise coprimes integers $p_0, \ldots, p_{\ell-1}$, and denote their product $\prod_{i=0}^{\ell-1} p_i$.

2. Choose $q_0 \leftarrow \mathbb{Z} \cap [0, \frac{2^\gamma}{\prod_{i=0}^{\ell-1} p_i})$ and define the error-free public key element $x_0 = q_0 \cdot \prod_{i=0}^{\ell-1} p_i$.

3. Choose ℓ_Q-bit integers $Q_0, \ldots, Q_{\ell-1}$.

4. Test Q_j and x_0, so that the $gcd(Q_j, x_0) = 1$ and abort otherwise.

5. Choose the public key elements x_i and x'_i, uniformly and independently distributed in $\mathbb{Z} \cap [0, q_0)$.

6. For $0 \leqslant j < \ell - 1$, compute x_i as follow:

$$\{x_i = \mathrm{CRT}_{q_0, p_0, \ldots, p_{\ell-1}}(q_{i0}, Q_{i,j} r_{i,j}, \ldots, Q_{i,\ell-1} r_{i,\ell-1})\}_{i=1}^{\tau}$$

7. For $0 \leqslant j < \ell - 1$, compute x'_i as follow:

$$\{x'_i = \mathrm{CRT}_{q_0, p_0, \ldots, p_{\ell-1}}(q'_{i0}, Q_{i,j} r'_{i,j} + \delta_{i,j}, \ldots, Q_{i,\ell-1} r'_{i,\ell-1} + \delta_{i,\ell-1})\}_{i=0}^{\ell-1}$$

where $q_{i0}, q'_{i0} \leftarrow \mathbb{Z} \cap [0, q_0)$, $r_{i,j}, r'_{i,j} \leftarrow \mathbb{Z} \cap [-2^\rho, 2^\rho)$ and $\delta_{i,j}$ is Kronecker delta.

8. Finally, let $\mathbf{pk} = \left\{ x_0, (Q_i)_{0 \leqslant i \leqslant \ell-1}, (x_i)_{1 \leqslant i \leqslant \tau}, (x'_i)_{0 \leqslant i \leqslant \ell-1} \right\}$
9. Output the secret key $\mathbf{sk} = (p_0, \ldots, p_{\ell-1})$.

BDGHV.Encrypt(pk, $m \in \{0,1\}$): For any $\mathbf{m} = (m_0, \ldots, m_{\ell-1})$ with $m_i \in \mathbb{Z}_{Q_i}$, output the ciphertext:

$$c = \left[\sum_{i=0}^{\ell-1} m_i \cdot x'_i + \sum_{i=1}^{\tau} b_i \cdot x_i \right]_{x_0} \tag{4.36}$$

where b is a random integer vector $\mathbf{b} = (b_i)_{1 \leqslant i \leqslant \tau} \in \{0,1\}^\tau$.

BDGHV.Decrypt(sk,c): Output $\mathbf{m} = (m_0, \ldots, m_{\ell-1})$ as follow:

$$m_i = ((c \bmod p_0) \bmod Q_0, \cdots, (c \bmod p_{\ell-1}) \bmod Q_{\ell-1}). \tag{4.37}$$

BDGHV.Evaluate(pk,$C, c_1, \ldots, c_t$): Takes a public key pk as an input, the circuit C, and t-tuple ciphertexts c_i. Outputs $C_f(c_1, \ldots, c_t)$ using **Add** and **Mul** equations as given below:

Eval.Add (pk, c_1, c_2)**:**

$$c_1 + c_2 \bmod x_0. \tag{4.38}$$

Eval.Mul (pk, c_1, c_2)**:**

$$c_1 \times c_2 \bmod x_0. \tag{4.39}$$

4.5.3 Semantic Security

The security of the batch homomorphic encryption over the integers scheme is based on the the variant ℓ-decisional Error-Free Approximate-GCD$_Q$ problem ℓ-**DEF-AGCD**$_Q$. This variant was derived from the simpler variant approach the decisional Error-Free Approximate-GCD problem. So the security of the scheme is based on the **DEF-AGCD**.

Definition 4.7 (ℓ-Decisional Approximate-**GCD**$_Q$ Problem: ℓ-**DEF** $-$ **AGCD**$_Q$)**.** The $(\rho, \eta, \gamma, \ell_Q)$-$\ell$-decisional Error-Free Approximate-GCD$_Q$ problem:

- For η-bit distinct primes $p_0, \ldots, p_{\ell-1}$, a ℓ_Q-bit integers $Q_0, \ldots, Q_{\ell-1}$, and a uniformly chosen coprime $q_0 \in [0, 2^\gamma / \prod_{i=0}^{\ell-1} p_i)$.
- Given a γ-bit integer $x_0 := q_0 p_0 \cdots p_{\ell-1}$, with $\gcd(x_0, Q_i) = 1$ for $i = 0, \ldots, \ell-1$.
- Given polynomially many samples from $\mathcal{D} := \mathcal{D}_\rho((p_i)^\ell; (Q_i)^\ell; q_0)$, a set of X consisting of ℓ integers $x'_i = \mathrm{CRT}_{q_0, p_0, \ldots, p_{\ell-1}}(q'_{i0}, Q_{i,j} r'_{i,j} + \delta_{i,j}, \ldots, Q_{i,\ell-1} r'_{i,\ell-1} + \delta_{i,\ell-1})\}$, where $q'_{i0} \leftarrow \mathbb{Z} \cap [0, q_0)$, $r'_{i,j} \leftarrow \mathbb{Z} \cap [-2^\rho, 2^\rho)$.

Determine $b \in \{0, 1\}$ from $z = x + r \cdot b \mod x_0$ where $x \leftarrow \mathcal{D}$ and $r \leftarrow \mathbb{Z} \cap [0, x_0)$.

Chapter 5

Discussion, Platform and Results

5.1 Introduction

In this chapter we present a full description of our implementation of the batch somewhat homomorphic encryption over the integers scheme. We also demonstrate our interpretation of the algorithm and how we managed to convert it from a theoretical form into a software implementation. Also, we used a flowchart which provides comprehensive details of some variables of the algorithm. Then, we gives a complete description of our platform and setup that has been used to implement the algorithm code. Also, we present a an overview of software that was used to built our implementation. Moreover, we draw a close comparison between the overall performance of this work and the previous batch somewhat homomorphic encryption scheme implementation [CCK$^+$13]. The scheme implementation has three security complexity levels. Based on these levels we were able to make the comparison with other scheme.

5.2 Scheme Construction

This section provides a detailed road maps of our implementation of the batch somewhat homomorphic encryption over the integers scheme. Following the steps will enables anyone to replicate the scheme without any difficulties.

The scheme implementation consists of many steps, fall under four main algorithms. The first algorithm is the *KeyGen*, the second algorithm is *encrypt*, the third

algorithm is *decrypt*, and the last algorithm is *evaluate*. Below we recall each one of the four algorithms in details as we implement them.

5.2.1 BDGHV.KeyGen

The steps required to generate the public key are:

1. The notation 1^{λ} is a unary representation of the security parameter, which indicate the complexity of the cipher construction. The implementation begins by creating a base-10 integer.
2. λ_{10} is an integer comprised of λ-number of $1`s$; as an example if $\lambda = 3$, $\lambda_{10} = 111$.
3. Then, we used a function to convert λ_{10} from a decimal representation to a binary.
4. We also use another function to find the number of binary bits used to represent λ_{10}.
5. The number of bits is squared and the result is used as the upper limit for the total number of bits which can represent a group of coprime numbers.
6. The group of coprime numbers is randomly selected and is comprised of ℓ base-10 prime integers, each of which are less than $2^{NumBits(\lambda_{10})}$ and are referred to as p_i.
7. Given $2^{NumBits(\lambda_{10})}$, randomly choose a sets of ℓ-relative coprime $(p_0, \ldots, p_{\ell-1})$ as shown in Figure 5.1.
8. Count the required bits length to represent ℓ-relative coprime given $(p_0, \ldots, p_{\ell-1})$.
9. If the number of bits that needed to represent ℓ-relative coprime is $\leq \eta = \tilde{\mathcal{O}}(\lambda^2)$ then:
   (a) Set the secret key of the scheme to be $(p_0, \ldots, p_{\ell-1})$
   (b) The bit length of the secret key $= \eta$.
   (c) Search all relative coprime and determine the largest prime.

10. If the number of bits $> \eta$ abort and start again from step number 7.

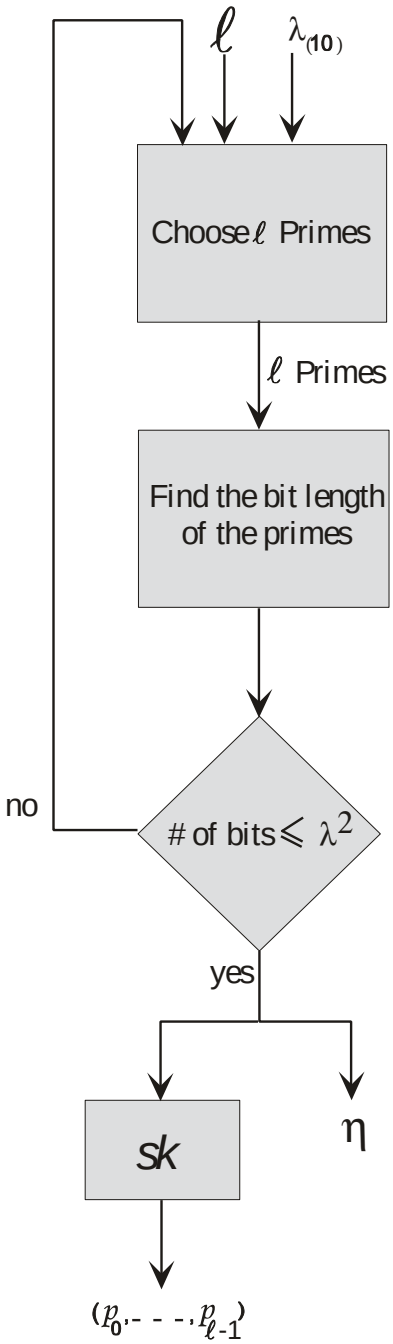

Figure 5.1: Secret Key Generation

.

11. Given $(p_0, \ldots, p_{\ell-1})$, compute $\prod_{i=0}^{\ell-1} p_i$.

12. Choose q_0 where $q_0 \leftarrow \mathbb{Z}[0, 2^{\lambda^5}/\prod_{i=0}^{\ell-1} p_i)$, q_0 must be $>$ largest prime and relatively prime with $p_0, \ldots, p_{\ell-1}$.

13. Generate ℓ-integers $Q_0, \ldots, Q_{\ell-1}$, the number of bit size is denoted by ℓ_Q. In this scheme, Q_i is the second element of the public key. For this scheme values of Q_i are $Q_0 = \cdots = Q_{\ell-1} = 2$.

14. Compute the error-free integer $x_0 = q_0.\prod_{i=0}^{\ell-1} p_i$, make sure that the $gcd(Q_j, x_0) = 1$ and abort otherwise. In our implementation, x_0 is the first element in the public key.

15. Using the Chinese Reminder Theorem compute x_i as follow:

$$\{x_i = \mathrm{CRT}_{q_0, p_0, \ldots, p_{\ell-1}}(q_{i0}, Q_{i,j} r_{i,j}, \ldots, Q_{i,\ell-1} r_{i,\ell-1})\}_{i=1}^{\tau}$$

    For all $0 \leqslant j < \ell - 1$, where $q_{i0}, \leftarrow \mathbb{Z} \cap [0, q_0)$, $r_{i,j}, \leftarrow \mathbb{Z} \cap [-2^{\rho}, 2^{\rho})$.

16. The computation will generate a matrix of size $\tau \times \ell$ as follow:

$$q_{i0}, r_{i,j} = \begin{pmatrix} q_{1,0} & Q_{1,j} \times r_{1,1} & \cdots & Q_{1,\ell-1} \times r_{1,\ell-1} \\ q_{2,0} & Q_{2,j} \times r_{2,1} & \cdots & Q_{2,\ell-1} \times r_{2,\ell-1} \\ \vdots & \vdots & \ddots & \vdots \\ q_{\tau,0} & Q_{\tau,1} \times r_{\tau,1} & \cdots & Q_{\tau,\ell-1} \times r_{\tau,\ell-1} \end{pmatrix}$$

17. After computation we will have a huge matrix of x_i. The size of the public element x_i is τ. In our implementation, $x_i's$ is the third elements in the public key.

$$x_i = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_\tau \end{pmatrix}$$

18. Moreover, using the CRT determine x'_i as below:

$$\{x'_i = \mathrm{CRT}_{q_0,p_0,\ldots,p_{\ell-1}}(q'_{i0}, Q_{i,j}r'_{i,j} + \delta_{i,j}, \ldots, Q_{i,\ell-1}r'_{i,\ell-1} + \delta_{i,\ell-1})\}_{i=0}^{\ell-1}$$

For all $0 \leqslant j < \ell - 1$, where $q'_{i0} \leftarrow \mathbb{Z} \cap [0, q_0)$, $r'_{i,j} \leftarrow \mathbb{Z} \cap [-2^\rho, 2^\rho)$ and $\delta_{i,j}$ is Kronecker delta.

Remark 5.1. The scheme has a Kronecker delta denoted by $\delta_{i,j}$. The $\delta_{i,j} = 1$ if $i = j$, otherwise it is $= 0$.

19. The computation will generate a matrix of size $\ell \times \ell$ as follow:

$$q'_{i0}, r'_{i,j} = \begin{pmatrix} q'_{1,0} & Q_{1,1} \times r'_{1,1} + \delta_{1,1} & \cdots & Q_{1,\ell-1} \times r'_{1,\ell-1} \\ q'_{2,0} & Q_{2,2} \times r'_{2,1} & \cdots & Q_{2,\ell-1} \times r'_{2,\ell-1} \\ \vdots & \vdots & \ddots & \vdots \\ q'_{\ell-1,0} & Q_{\ell-1,1} \times r'_{\ell-1,1} & \cdots & Q_{\ell-1,\ell-1} \times r'_{\ell-1,\ell-1} + \delta_{\ell-1,\ell-1} \end{pmatrix}$$

20. We have a matrix of size ℓ. In our work the last public key element is the values of x'_i.

$$x'_i = \begin{pmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_{\ell-1} \end{pmatrix}$$

21. Finally, we have the public key and the secret key in the following form:

$$pk = \left\{ x_0, Q_i, (x_1, x_2, \cdots, x_\tau), (x'_1, x'_2, \cdots, x'_{\ell-1}) \right\} \tag{5.1}$$

$$sk = (p_0, \ldots, p_{\ell-1}) \tag{5.2}$$

5.2.2 CRT Function

Before we talk about the encryption procedure, we will provide more details about the Chinese Remainder Theorem (CRT). The CRT function plays a major role in

generating the public key elements. The CRT function is used to compute two element of public key x_i and x_i'. The first element x_i is consist of table of integers. Each row of the table is used by the NTL libraries [Sho15] incremental Chinese Remainder Theorem (CRT) function to generate a value x_i. To save memory in the system the table is generated a single row per iteration and then discarded; there are τ rows in the table requiring τ iterations.

The NTL CRT function is used incrementally across the row to generate a final value; the function is called repeatedly in a loop to solve simultaneous congruences between the values in the row and the prime numbers previously selected. CRT takes four parameters, CRT(a,p,A,P). For the initial iteration the parameters are:

$$a = q_{i0} \qquad p = q_0 \qquad A = r_{i1} \qquad P = p_0 \tag{5.3}$$

Where:

- q_{i0} the first entry in row i
- q_0 is the first chosen prime
- r_{i1} is the first r_{ij} of row 0
- p_0 is the second chosen prime

To ensure that A is within the range of P:

$$A = A \pmod{P} \tag{5.4}$$

$$CRT(a, p, A, P) \tag{5.5}$$

The first iteration simultaneously solves for the congruency between $a \pmod{p}$ and $A \pmod{P}$. The result is an integer that is congruent for both relations. The CRT function is what is called an implicit function; explicit functions return values in a mode common to natural human thought (i.e. $y = foo(x)$) where implicit functions such as NTL's CRT return the value in one of the parameters given. In the case of the two simultaneous congruences above the resulting base-10 integer solution is placed back into the parameter a.

For the following iterations the parameters are assigned sequentially while the result of each iteration is maintained in the parameter a and used for the subsequent computation. To clarify, for the second iteration the parameters become:

$$a_2 = a_1, p = p_0, A = r_{i2}, P = p_1 \tag{5.6}$$

$$CRT(a_2, p, A, P) \tag{5.7}$$

This iterative pattern is continued until the congruence between each prime number and respective r_{ij} value is accumulated into the parameter a. Upon completion of the entire row the final x_i value is calculated and stored.

$$x_i = a \pmod{x_0} \tag{5.8}$$

Upon completion of the entire table an array of x_i values of τ-length has been created. These values are used as removable noise to mask each bit of the message.

5.2.3 BDGHV.Encrypt

The encryption operation used the following steps:

1. Given the public key, encrypt ℓ-bit victor massage $m_0, \ldots, m_{\ell-1}$. The encryption algorithm consist of three steps.

2. First, given the CRT result x_i', compute the massage vector as shown in Figure 5.2.

3. Second, given the CRT result x_i, generate a random subset sum of $1, \cdots, \tau$.

4. Lastly, add the results from 2 and 3, then $\pmod{x_0}$, output the ciphertext c.

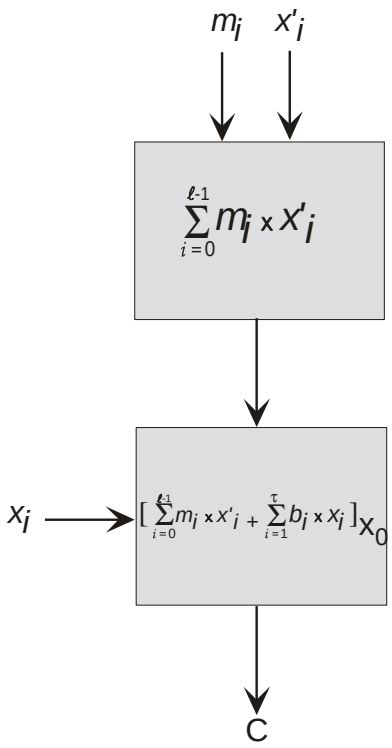

Figure 5.2: Encryption

.

5. A random number of the x_i values are chosen from the resulting array and are summed; the values chosen are randomly selected and pertain to no-order. The resultant sum is summed with the base-10 integer value of each message bit (i.e. 0 or 1). To create the cipher the result of the summation of the x_i values, added with the message bit is modulated by X_0.

$$Cipher_i = (sum_i + m_i) \bmod X_0 \tag{5.9}$$

5.2.4 BDGHV.Decrypt

The decryption operation uses the following steps:

1. Given the secret key $p_0, \ldots, p_{\ell-1}$, decrypt the ciphertext c.

2. Given the ciphertext, $(\,(\mathrm{mod} p_i) \mod 2)$ will generate the corresponding original encrypted massage $m_0, \ldots, m_{\ell-1}$ as shown in Figure 5.3.

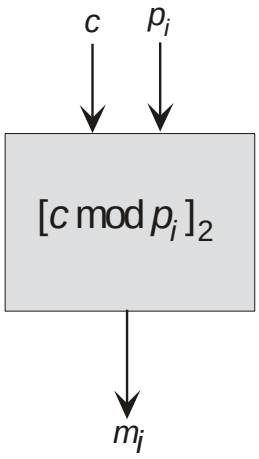

Figure 5.3: Decryption

.

5.2.5 BDGHV.Evaluate

The evaluation operation uses the following steps:

1. Given the public key, the circuit C with t-tuple of ciphertexts c.

2. Perform all the additions and multiplications operations over ciphertexts as shown in Figure 5.4.

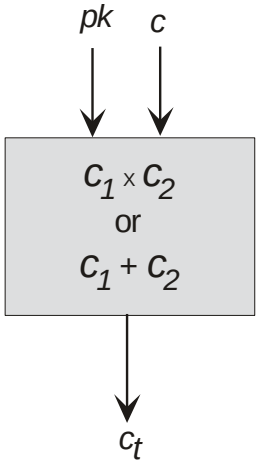

Figure 5.4: Evaluate

.

5.3 Platform

The proof of concept for a new CRT-based fully homomorphic encryption scheme, intended for development in a hardware environment is being performed through software. The implementation was built using C++11. The algorithm developed calls for the use of integer numbers which are too large for ordinary computer operations to handle. Computer systems create data structures for numerical values that have fixed limits based on hardware restrictions of the system. These pre-defined data structures limit the capability of cryptographic integrity. Numerical values which exceed the predefined limitations of a system are referred to as *Big Nums* for short. To handle such constructs, specialized memory management techniques must be employed; to accommodate this we need to use Victor Shoup's NTL [Sho15]library. NTL is a high-performance, portable C++ library which provides developers data structures and algorithms that dynamically manage memory to allow for *Big Nums* computation.

5.3.1 File Layout

The implementation folder layout contains five files. First, CryptCode.cpp is the central file for the implementation which contains the main function. The main file must be compiled with a link to the NTL static library and inclusion of NTL's header files. For optimal performance it should be run with the *openMP* parallelization API. Second, Encryption.h file contains functions for multiple methods of encryption and their support functionality. Third, the Decryption.h contains two methods that belong to the decryption methodology. Fourth, the Evaluation.h file comprises of two main methods: *SumEval* and *MulEval*. Lastly, the CryptUtilities.h file is comprised support functions which are used by the other files.

5.3.2 NTL

The NTL library provides data structures that allow for arbitrary length integers by creating structures that list pointers to locations in memory. A standard data format, such as in 'integer' is allocated a defined amount of space in memory. The rigid definition only allows for certain number of bits per entity; in the case of 'integer' the defined number of bits is 32 resulting in a maximum representable value of $2^{32} - 1$.

NTL provides multiple data structures that overcome the obstacle of finite size data entities. The implementation employs the structure known as "ZZ" type. The ZZ data type is a structure that holds pointers to multiple places in memory. When an integer of type ZZ is created that requires more bits than the operating system pre-defines the ZZ construct requests additional places in memory from the operating system and keeps track of them via a list of pointers. When the total value is required for display or processing the construct aligns all memory locations pointed to in the list to create the total value.

Due to the complex nature of the ZZ data type the library includes procedures that define how the system is to operate on data that is partitioned over multiple locations in memory. Basic algebraic operations to complex algorithms are provided for the computation of *Big Num* mathematics and are used in this implementation.

5.3.3 Optimization

To improve performance an application-program interface (API) called *openMP* was used to automatically parallelize portions of the implementation. The manual creation of threading is fraught with difficulty resulting from synchronization issues, shared data concerns and effective workload distribution. *OpenMp* uses 'pragmas', simple one line compiler directives recognizes as commands for use of the API. The most common and effective use is the parallelization mutually exclusive iterations of a loop such as that which uses NTL's CRT function to generate the x_i values. For a loop a 'pragma' can be used to instruct the compiler to refer to the *openMP* API for the development and management of threads.

```
//Fills a vector of length taw with random integers between 0 and Q0
void FillQ0(ZZ* rQ0, const ZZ &Q0, const long &taw)
{
    ZZ two(2);
#pragma omp parallel
 {
    SetSeed(ZZ(omp_get_thread_num()));
    #pragma omp for
    for(int i = 0 ; i < taw; i++){

        do{
            RandomBnd(rQ0[i], Q0);
        } while(rQ0[i] > (Q0/two) || IsOdd(rQ0[i]) != 1);
    }
 }
}
```

Figure 5.5: Code Sample

.

The above Figure 5.5 shows the implementation for the *FillQ0* function used to generate the q_{i0} values used in encryption. The directive ***#pragma omp parallel*** notifies the compiler that the enclosed section is intended for parallelization and that it is to refer to the *openMP* API. The directive ***pragma omp for*** signals to the compiler that the subsequent line is the start of a loop that is to be parallelized. The compiler refers to the API to determine the ideal number of threads to generate for the system being used. The number of iterations to execute are divided evenly among the generated threads and applicable data is kept private between them while the target array, q_{i0} is shared. The execution of the loop is improved by a factor equal to the number of threads created. Upon completion the threads are terminated and the serial execution continues. The use of compiler directives allows the code to maintain portability. When execute on a compiler that does not have the API the instructions are viewed as comments and ignored.

5.4 Simulation Results

In this section, we will provides previous implementation results of the somewhat homomorphic encryption scheme [CCK$^+$13]. Then, we draw our implementation

results of the scheme. Also, we demonstrate a full comparison between our work results and others.

The previous implementation by Cheon *et al.* [CCK+13] was built using C++. They used the GMB library to accommodate their code. The batch DGHV scheme test platform was implemented on an Intel Core i7 machine, with 3.4Ghz processor and 32GB of memory.

5.4.1 Test Platform

Our platform, tests of the implementation were executed on either the Hermes or Nestor cluster of the *WestGrid* research computing network. The Nestor cluster is preferred when the use of the *openMP* API is employed. Each node in the Nestor or Hermes clusters is an IBM iDataplex server with eight 2.67-GHz Xeon x3330 cores, each with 24GB of RAM.

Our implementation consists of two parts. The first one is based on the work of Cheon *et al.*. To provide a basis for fair comparison with the previous work, we assign the exact values to all different variables that has been used in the implementation. In the Batch DGHV, they have three test settings.

First test setting is the small setting $\lambda = 52$, where λ determines the security of the construction. In this test we enforce the same parameter values to make sure that we can make a fair comparison. For the small setting our public key generating timing is longer than the previous work as shown in Table 5.1. This occurs due to our threading techniques. On the other hand we were able to compress the public key size.

Table 5.1: The batch DGHV vs this work results for small security level $\lambda = 52$

Instance	λ	ℓ	ρ	η	$\gamma/10^6$	τ	pk size	**KeyGen(s)**	**Encrypt(s)**	**Decrypt(s)**	**ADD Eval(s)**	**MUL Eval(s)**
Pervious Work [CCK+13]	52	37	41	1558	0.90	661	13 MB	1.74s	0.23s	0.02s	-	-
This Work	52	37	41	1558	0.038	661	5.02 MB	2.71s	0.00047s	0.0024s	0.0037s	0.041s

In term of security, we maintain the same security level of the original scheme. In term of efficiency, for encryption and decryption processes experimental results demonstrate that our implementation runs an order of magnitude faster than previous work. In our work, we also report the addition and multiplication processes timing

over ciphertexts.

Second test setting is for $\lambda = 62$. In our implementation we established a threading procedure to increase the efficiency of the scheme. As shown in Table 5.2, the size of the public key have been reduced by more than two thirds of the previous work. Moreover, for the medium setting the key generation timing of our implementation is faster than the other work. As we increase the security complexity level of the scheme, we obtained a better timing results. Also, the encryption and decryption timing results of our work is faster than the previous work by order of magnitude.

Table 5.2: The batch DGHV vs this work results for medium security level $\lambda = 62$

Instance	λ	ℓ	ρ	η	$\gamma/10^6$	τ	pk size	**KeyGen(s)**	**Encrypt(s)**	**Decrypt(s)**	**ADD Eval(s)**	**MUL Eval(s)**
Pervious Work [CCK+13]	62	138	56	2128	4.6	2410	304 MB	73s	3.67s	0.45s	-	-
This Work	62	138	56	2128	0.703	2410	90.38 MB	30s	0.014s	0.088s	0.097s	1.94s

Last one is the large setting where $\lambda = 72$. Furthermore, the public key size of the scheme have been minimized in our implementation by more than two third as shown in Table 5.3. Also, as of the medium setting, the effect of the threading approach on key generation time is obvious. The Batch DGHV requires almost an hour to generate the public key, but in our work we were able to do that in half an hour. In addition, the encryption and decryption timing is also faster than the previous work. In the large setting, our implementation can encrypt a vector of massage of size 531-bit in parallel.

Table 5.3: The batch DGHV vs this work results for large security level $\lambda = 72$

Instance	λ	ℓ	ρ	η	$\gamma/10^6$	τ	pk size	**KeyGen(s)**	**Encrypt(s)**	**Decrypt(s)**	**ADD Eval(s)**	**MUL Eval(s)**
Pervious Work [CCK+13]	72	531	71	2698	21	8713	5.6 GB	3493s	61s	9.8s	-	-
This Work	72	531	71	2698	12.23	8713	1.58 GB	1129s	0.22s	1.12s	1.23s	23.8s

In our implementation, after each evaluation operation the noise element increase by doubles on addition and squares on multiplication, for all three security settings small, medium and large. To investigate the durability of the ciphers under repeated operations of a single nature, either multiplication or addition a binary tree structured test is implemented for each operation. The evaluation begins by encrypting each message a total of eight times to generate sixteen individual ciphers. The eight cipher pairs are operated together (dependant on the tree) to create a new cipher. These new

ciphers belong to the second level of their respective trees. At each level a decryption of the resultant cipher occurs to verify the original message is still attainable. This has been done for all three security setting to determine how many operations we can perform.

Chapter 6

Contributions

In this thesis, we introduced the implementation of the CRT-based somewhat homomorphic encryption over the integers scheme. The main contributions of the thesis can be summarized as follows:

1. Provide a proof of concept of the CRT-based somewhat homomorphic encryption over the integers algorithm.

2. Constructed a detailed road map of the implemented scheme. All previous published works were unclear, vague, ambiguous and hard to replicate.

3. Implemented the algorithm on three security complexity settings. The three settings are small, medium, and large, where $\lambda = 52, 62,$ and 72, respectively. The choices of λ facilitates meaningful comparisons with previously published results.

4. Succeeded in reducing the scheme public key size significantly. For the small security level we reduced the size of the public key around $\approx 60\%$. For the medium setting, the public key size is reduced about $\approx 70\%$. Moreover, for the large setting a reduction in term of public key size some $\approx 70\%$.

5. Parallelized the implementation using C++11 augmented with *openMP* pragma directions to automatically parallelize portions of the implementation. By using this technique, we have increased the efficiency of the encryption algorithm. We achieve an enhancement in terms of public key generation computation

time with respect to previous works. For the large setting we have reduced the requir time by $\approx 65\%$. Also, for the medium setting, a reduction in term of computation time $\approx 55\%$ is attained.

6. Reduced the computation time of encryption and decryption processes compared to previous results. For both encryption and decryption, our implementation have reduced the computation time of them by orders of magnitude.

To sum up, the main purpose of developing the DGHV somewhat homomorphic encryption over the integers scheme is to reduce the complexity of Gentry's scheme. The significance of the development of such scheme is that proved out the existent of different mathematical theories that can be applied to construct similar homomorphic encryption schemes.

Our implementation have enhanced and increased the efficiency of the batch somewhat homomorphic encryption over the integers scheme. This work is an effort towards more practical homomorphic encryption scheme. The fully homomorphic encryption schemes have influenced many researchers to seek for more mathematical methods and procedures to enhance the capabilities and efficiency. Currently, the existing homomorphic encryption schemes still have lots of room for further enhancement before they can be used in the real world.

Bibliography

[AD97] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 284–293. ACM, 1997.

[BEAS09] Elizabeth A. Burroughs, Tyler J. Evans, Thomasenia Lott Adams, and Joanne Snow. mathematical roots: The chinese remainder theorem. *Mathematics Teaching in the Middle School*, 14(7):pp. 436–443, 2009.

[BGH13] Zvika Brakerski, Craig Gentry, and Shai Halevi. Packed ciphertexts in lwe-based homomorphic encryption. In *Public-Key Cryptography–PKC 2013*, pages 1–13. Springer, 2013.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of cryptography*, pages 325–341. Springer, 2005.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 309–325. ACM, 2012.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Advances in Cryptology–CRYPTO 2012*, pages 868–886. Springer, 2012.

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *Proceedings of the 2011 IEEE*

52Nd Annual Symposium on Foundations of Computer Science, FOCS '11, pages 97–106, Washington, DC, USA, 2011. IEEE Computer Society.

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Advances in Cryptology–CRYPTO 2011*, pages 505–524. Springer, 2011.

[BY88] Ernest F Brickell and Yacov Yacobi. On privacy homomorphisms. In *Advances in CryptologyEUROCRYPT87*, pages 117–125. Springer, 1988.

[CCK+13] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrede Lepoint, Mehdi Tibouchi, Aaram Yun, et al. Batch fully homomorphic encryption over the integers. In *EUROCRYPT*, volume 7881, pages 315–335. Springer, 2013.

[CF85] Josh D Cohen and Michael J Fischer. A robust and verifiable cryptographically secure election scheme. In *FOCS*, volume 85, pages 372–382, 1985.

[CH11] Henry Cohn and Nadia Heninger. Approximate common divisors via lattices. *arXiv preprint arXiv:1108.2714*, 2011.

[CLT13] Jean-Sbastien Coron, Tancrde Lepoint, and Mehdi Tibouchi. Batch fully homomorphic encryption over the integers. Cryptology ePrint Archive, Report 2013/036, 2013.

[CLT14] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Scale-invariant fully homomorphic encryption over the integers. In *Public-Key Cryptography–PKC 2014*, pages 311–328. Springer, 2014.

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

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

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

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

[DJN10] Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of pailliers public-key system with applications to electronic voting. *International Journal of Information Security*, 9(6):371–385, 2010.

[DR00] Joan Daemen and Vincent Rijmen. The block cipher rijndael. In *Smart Card Research and Applications*, pages 277–284. Springer, 2000.

[DR02] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2002.

[EJ03] Patrik Ekdahl and Thomas Johansson. A new version of the stream cipher snow. In *Selected Areas in Cryptography*, pages 47–61. Springer, 2003.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology*, pages 10–18. Springer, 1985.

[FIP00] PUB FIPS. 186-2. digital signature standard (dss). *National Institute of Standards and Technology (NIST)*, 2000.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

[GH11a] Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 107–109. IEEE, 2011.

[GH11b] Craig Gentry and Shai Halevi. Implementing gentrys fully-homomorphic encryption scheme. In *Advances in Cryptology–EUROCRYPT 2011*, pages 129–148. Springer, 2011.

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

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

[GHS12c] Craig Gentry, Shai Halevi, and Nigel P Smart. Homomorphic evaluation of the aes circuit. In *Advances in Cryptology–CRYPTO 2012*, pages 850–867. Springer, 2012.

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. A simple bgn-type cryptosystem from lwe. In *Advances in Cryptology–EUROCRYPT 2010*, pages 506–522. Springer, 2010.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.

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

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International Journal of Information Security*, 1(1):36–63, 2001.

[Kah74] David Kahn. *The codebreakers*. Weidenfeld and Nicolson, 1974.

[KLYC13] Jinsu Kim, Moon Sung Lee, Aaram Yun, and Jung Hee Cheon. Crt-based fully homomorphic encryption over the integers. *IACR Cryptology ePrint Archive*, 2013:57, 2013.

[KS92] Y.H. Ku and Xiaoguang Sun. The chinese remainder theorem. *Journal of the Franklin Institute*, 329(1):93 – 97, 1992.

[Lag85] Jeffrey C Lagarias. The computational complexity of simultaneous diophantine approximation problems. *SIAM Journal on Computing*, 14(1):196–209, 1985.

[LATV12] Adriana López-Alt, Eran Tromer, and Vinod 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*, pages 1219–1234. ACM, 2012.

[LJ87] Hendrik W Lenstra Jr. Factoring integers with elliptic curves. *Annals of mathematics*, pages 649–673, 1987.

[LLJMP90] Arjen K Lenstra, Hendrik W Lenstra Jr, Mark S Manasse, and John M Pollard. The number field sieve. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 564–572. ACM, 1990.

[MGH10] Carlos Aguilar Melchor, Philippe Gaborit, and Javier Herranz. Additively homomorphic encryption with d-operand multiplications. In *Advances in Cryptology–CRYPTO 2010*, pages 138–154. Springer, 2010.

[NS98] David Naccache and Jacques Stern. A new public key cryptosystem based on higher residues. In *Proceedings of the 5th ACM conference on Computer and communications security*, pages 59–66. ACM, 1998.

[OU98] Tatsuaki Okamoto and Shigenori Uchiyama. A new public-key cryptosystem as secure as factoring. In *Advances in CryptologyEUROCRYPT'98*, pages 308–318. Springer, 1998.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in cryptologyEUROCRYPT99*, pages 223–238. Springer, 1999.

[PPP09] Bart Preneel, Christof Paar, and Jan Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer, 2009.

[RAD78] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[Reg04] Oded Regev. New lattice-based cryptographic constructions. *Journal of the ACM (JACM)*, 51(6):899–942, 2004.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

[Ros93] Kenneth H. Rosen. *Elementary number theory and its applications (3. ed.)*. Addison-Wesley, 1993.

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

[Sch07] Bruce Schneier. *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons, 2007.

[Sho15] Victor Shoup. NTL 8.1.2 : A library for doing number theory, 2015. `www.shoup.net/ntl`.

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

[SYY99] Tomas Sander, Adam Young, and Moti Yung. Non-interactive cryptocomputing for nc 1. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 554–566. IEEE, 1999.

[VDGHV10] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology–EUROCRYPT 2010*, pages 24–43. Springer, 2010.

[Ver26] Gilbert S Vernam. Cipher printing telegraph systems: For secret wire and radio telegraphic communications. *AIEE, Journal of the*, 45(2):109–115, 1926.

Appendix A

Additional Information

A.1 Test Results

A.1.1 A. Small

We used a 37-bit messages

$M1 = 1010110101101011010110101101011010110$

$M2 = 1100111001110011100111001110011100111$

- $\lambda = 52$
- $\tau = 661$
- $\rho = 41$
- $\eta = 1558$
- $\ell = 37$
- Public key generation time = 2.71s
- CRT Array NumBits = 38.0332Mb
- Public Key length = 40.2197Mb
- Encryption *time* = 0.47ms
- Decryption *time* = 2.38ms

- Summation Eval time = 3.74ms
- Multiplication Eval time = 40.99ms

A.1.2 B. Medium

We used a 62-bit messages

$M1$ = 10101101011010110101101011010110101101011010110101101011010110
10110101101010110101101011010110101101011010110101101011010110

$M2$ = 11001110011100111001110011100111001111100111001110011100111001
00111001111100111001110011100111001110011111001110011100111001110011

- $\lambda = 62$
- $\tau = 2410$
- $\rho = 56$
- $\eta = 2128$
- $\ell = 138$
- Public key generation time = 30s
- CRT Array NumBits = 683.633Mb
- Public Key length = 723.063Mb
- Encryption time = 13.9ms
- Decryption time = 87.97ms
- Summation Eval time = 96.9ms
- Multiplication Eval time = 1.94s

A.1.3 C. Large

We used a 72-bit messages

$M1$ = 10101101011010110101101011010110101101010110101101011010110
10110101101010110101101011010110101101011010110101011010110101101011010110
10101101011010110101101011010110101101010110101101011010110101101011010110
10101101011010110101101011010110101101010110101101011010110101101010110101
10101101011010110101101011010101101011010110101101011010110101101010110101
10101101011010110101101011010101101011010110101101011010110101011010110101
10101101011010110101011010110101101011010110101101011010110101011010110101
101011010110101101010 10

$M2$ = 11001110011100111001110011100111001111100111001110011100111
00111001111100111001110011100111001110011100111110011100111001110011100111
11001110011100111001110011100111001111100111001110011100111001110011100111
11001110011100111001110011100111001111100111001110011100111001111100111001
11001110011100111001110011111001110011100111001110011100111001111100111001
11001110011100111001110011111001110011100111001111100111001110011100111001
11001110011100111110011100111001110011100111001110011111001110011100111001
1100111001110011111 0011

- $\lambda = 72$
- $\tau = 8713$
- $\rho = 71$
- $\eta = 2698$
- $\ell = 531$
- Public key generation time = 1129s
- CRT Array NumBits = 11945.4Mb
- Public Key length = 12674.8Mb
- Encryption time = 0.22s

- Decryption time = 1.12s
- Summation Eval time = 1.23s
- Multiplication Eval time = 23.8s