CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Fully Homomorphic Encryption: A Case Study

Master's Thesis in Computer Systems and Networks

EMELIE WIDEGREN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Master's thesis 2018

Fully Homomorphic Encryption: A Case Study

EMELIE WIDEGREN

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2018

Fully Homomorphic Encryption: A Case Study
EMELIE WIDEGREN

Supervisor: Andrei Sabelfeld, Department of Computer Science and Engineering
Supervisor: Elena Pagnin, Department of Computer Science and Engineering
Examiner: Tomas Olovsson, Department of Computer Science and Engineering

Master's Thesis 2018
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2018

Fully Homomorphic Encryption: A Case Study
EMELIE WIDEGREN
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Fully Homomorphic Encryption (FHE) has been dubbed as cryptography's holy grail. It opens the door to many new capabilities with the goal to solve the IT world's problems of security and trust. After 2009, when Craig Gentry showed that FHE can be realised, research in the area exploded and substantial progress has been made in finding more practical and more efficient schemes.
FHE is a cryptographic primitive that allows one to compute arbitrary functions over encrypted data. Such schemes have numerous applications since it allows users to encrypt their private data locally but still outsource the computation of the encrypted data without risking exposing the actual data.
In 2012, LTV12 published the first multi-key FHE scheme and proved that any Somewhat Homomorphic Encryption (SHE) scheme could be made multi-key. As in the single key setting, a lot of progress been made in the area but no work has been done in implementing the multi-key schemes.
This thesis is survey on FHE and MKFHE, with special attention to the state of the art implementations available as well as three implementations including the first implementation in the multi-key settings to the best of our knowledge.

Keywords: public-key cryptography, fully homomorphic encryption, multi-key FHE, LWE, GSW.

Acknowledgements

First and foremost, I would like to thank my supervisor Elena Pagnin for her endless help and support during the thesis. Elena invested a big amount of time, effort and patience not only into helping me with this project, but also into involving me in academic life by inviting me to talks and meetings and introducing me to lots of people. It would not have been possible to do this without her.
Also a great thank you to Andrei Sabelfeld, Tomas Olovsson, Michal Palka, Carlo Brunetta, Martin Kastebo, Alexandra Back, Emma Westman, Maria Larsson and Amanda Nilsson who provided me with valuable feedback during the process.
Finally, I would like to thank friends and family for their infinite love and support. A special thanks to my mom, brother and boyfriend for never ever doubting me, not even for a second.

Emelie Widegren, Gothenburg, October 2018

Contents

List of Figures

List of Tables

Glossary

BGV A Fully Homomorphic Encryption (FHE) scheme developed by Brakerski, Gentry & Vaikuntanathan [1].

FDMK A multi-key FHE scheme [2].

GSW A FHE scheme developed by Gentry, Sahai & Waters [3].

LTV A multi-key FHE scheme developed by López-Alt, Tromer & Vaikuntanathan based on NTRU [4].

NTRU A public-key cryptosystem that uses lattice-based cryptography [5].

RSA One of the first practical public-key cryptosystems, that is also partially homomorphic [6].

SHMK A Single-Hop Multi-Key FHE scheme by Mukherjee & Wichs [7].

Acronyms

BDD Bounded Distance Decoding Problem.

CVP Closest Vector Problem.

DLWE Decisional Learning With Errors.

FHE Fully Homomorphic Encryption.

GapCVP$_\gamma$ Gap Closest Vector Problem.
GapSVP$_\gamma$ Gap Shortest Vector Problem.

LPN Learning Parity with Noise.
LWE Learning With Errors.

NP Non-deterministic Polynomial Time.
NP-Hard Non-deterministic Polynomial Acceptable Problems.

P Polynomial Time.
PPT Polynomial Time Algorithm.

RLWE Ring Learning With Errors.

SHE Somewhat Homomorphic Encryption.
SIS Short Integer Solutions Problem.
SVP Shortest Vector Problem.

1 Introduction

We live in an era where most people all over the world own more than one digital device with limited local storage. Therefore, there is a growing need for services that let users easily store and access personal files. Data that was previously stored on paper is being converted to a digital file. However, some data needs to be kept secret and is not meant for public consumption. Several cloud services allow one to securely upload private data that is encrypted, but in many cases, once the data is uploaded, the cloud service can decrypt the data. It does this because most encryption schemes no longer produce the correct decrypted value once computations have been performed on the data. This is not the case with homomorphic encryption schemes.

Encryption is a method used for encoding information with the goal to ensure confidentiality so that only authorised parties can access the information. There exists different types of encryption schemes that can be either symmetric or asymmetric. In the symmetric setting, the same key is used for encryption and decryption and it is commonly used when a secure channel is already established. In the asymmetric setting, there exists a public and private key for each party where the public key is used for encryption and the private key for decryption. The public key is shared between parties while the private key is kept secret so that only the holder of the secret key can decrypt the message encrypted under the corresponding public key. In 1978, shortly after the discovery of public key cryptography, Rivest et al. were the first to observe the possibility of manipulating encrypted data in a meaningful way, rather than just storing and retrieving it [8].

A special form of encryption is homomorphic encrytion that has a wide range of applications and supports operations on encrypted data. The outcome, is a ciphertext that, when decrypted, matches the result of the desired computation performed on the plaintext. They raised the question: *Can we do arbitrary computations on data while it remains encrypted, without ever decrypting it?* which asks for the ability to perform computations on encrypted data without being able to "see" the data. This ability allows users to encrypt their private data locally; send it to the cloud service along with the computations they would like to perform on the data; have the cloud service perform these computations and send back the encrypted result; and then decrypt the result with high certainty that the private data was not exposed. The flow of this idea can be seen in Figure 1.1.

A metaphor often used for homomorphic cryptosystems is the one of a jewellery shop [9]. Alice owns a jewellery shop and has raw precious material that she wants

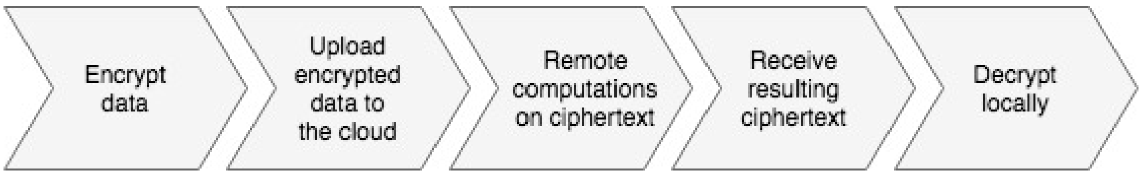
Encrypt data
Upload encrypted data to the cloud
Remote computations on ciphertext
Receive resulting ciphertext
Decrypt locally

Figure 1.1: Ideal flow when working with the cloud.

her workers to assemble into jewellery. The problem is that she distrusts her workers - she is afraid they will steal the material if given the opportunity. She wants her workers to be able to process the materials without actually having access to them. In order to solve this problem, Alice designs a transparent, impenetrable glove box, puts the raw material inside, locks the box with a key that only she has access to and then gives the box to one of the workers. The worker can assemble the jewels inside the box using the gloves without being able to access the materials inside since it is impenetrable. Once the worker is finished, the worker gives the box back to Alice who can unlock it with her key and extract the jewellery. In this metaphor, the data is represented by the material that needs to be processed and the encryption of that data is represented by the box. The special thing about this cryptosystem is that it has gloves allowing the data to be processed without accessing it.

This concept of homomorphic cryptosystems was however long viewed as a fantasy. After some progress over the years where several cryptosystems where found to be partially homomorphic [10, 11, 12, 13], a breakthrough occurred in 2009 when Gentry presented the first FHE scheme [9]. Although Gentry's scheme was not yet useful for practical implementations it presented a solution to achieve privacy homomorphism.

Gentry's FHE scheme is an asymmetric encryption scheme based on ideal lattices. Essentially one generates a secret key and then a number of public keys, each containing "noise", in a way that it is infeasible for an adversary to generate the secret key from the public keys. The problem with this first solution is that the "noise" in the ciphertext grew with each additional computation. This means that at a certain point the ciphertext will no longer decrypt to the original message because the "noise" has grown to large. Encryption schemes with this property are called Somewhat Homomorphic Encryption (SHE) schemes. The term somewhat stresses that the number of homomorphic operations one can perform is limited. In the same work [9] Gentry also provides a generic technique to transform SHE in FHE. Such technique is called Bootstrapping. Bootstrapping solves the problem of not being able to decrypt properly when the "noise" grows too large by homomorphically decrypting the ciphertext, performing a single computation on it, and then recrypting under a different public key. Unfortunately, the bootstrapping procedure is too theoretical and therefore not very efficient. Additionally, bootstrapping requires the non-standard assumption circular security which assumes that it is safe to encrypt the secret key under its own public key [2, 9, 14].

Since Gentry's first FHE scheme in 2009, several others have been developed [15, 16, 17, 18, 19]. The focus of subsequent works has been to simplify the construction of FHE schemes as well as to base the security on more standard assumptions. In

2010, Smart and Vercauteren [16] made the first attempt of implementing FHE and since then a lot of work has been done towards more practical implementations [20, 21, 22].

FHE is only suitable in settings where the computations involve a single user since it requires the input from the users to be encrypted under the same key. Imagine instead a scenario where users, who have uploaded data to the cloud in encrypted form, wish to compute some joint function of their data encrypted under different keys. To handle these multi party situations, in 2012, Lopéz-Alt et al. [4] introduced a multi-key FHE scheme based on the NTRU cryptosystem [5]. This scenario is significantly more complex than the single user setting but even in this area a lot of recent improvements have been done [2, 7, 23]. Contrary to the single user setting, there exist no implementation of multi-key FHE to the best of the author's knowledge.

1.1 Aim

The goal of this thesis is to acquire a deep understanding of (1) the challenges involved in implementing a FHE scheme and (2) the theoretical knowledge required to build secure FHE schemes. This is achieved by performing an extensive study of the design choices and challenges when implementing FHE. More precisely, this is done in two steps;

1. performing a state of the art literature study,
2. studying and implementing three somewhat similar FHE schemes: one by Gentry et al [3] and two multi-key schemes by Mukherjee and Wichs [7] and Perlman and Brakerski [2] that have not previously been implemented.

A consequence of choosing non-trival schemes that have not already been implemented is that the difficulties are in general unpredictable. The challenges include studying and translating the scheme into code, choosing parameters, designing algorithms, choosing the benchmarks and running experiments on them. In addition, the schemes are based on some non-standard assumptions and Perlman and Brakerski state that their scheme is not practical which makes implementing it a big challenge [2]:

"We stress that our scheme is not by itself practical. We use the bootstrapping machinery in a way that introduces fair amounts of overhead into the evaluation process. The goal of this work, rather, is to indicate that the theoretical boundaries of multi-key FHE, and open the door for further optimisations bringing solutions closer to the implementable world."

The ambition of the thesis is that the findings will serve as a stepping-stone for designing more practical FHE schemes even beyond these particular ones. Therefore, we address the following questions:

- What are the challenges of implementing FHE schemes?

- What implementation techniques can be used to optimise FHE schemes?
- Is it possible to make FHE schemes rely on more standard assumptions?
- How does one choose parameters that guarantee a given level of security?
- How does multi-key FHE compare to single-key FHE in terms of performance?

This thesis work focuses on the theory study of FHE and the challenges of implementing FHE and multi-key FHE. The outcome shall include proof of concept implementations but the aim is not to achieve the most efficient implementations due to the complexity of the task

1.2 Contribution

In this work, we present a state of the art survey on FHE in general as well as the first implementation of multi-key FHE. In more details, we present the following:

- State of the art of FHE.
- Overview of currently existing schemes.
- Implementations of two multi-key FHE schemes [2, 7].
- Overview of existing tools and libraries that can be useful for implementing and optimising FHE.
- Overview of currently public implementations of FHE.

1.3 Outline

This thesis is organised as follows. Chapter 2 describes the required cryptographic and mathematical background needed for FHE. Chapter 3 describes the concept of homomorphic encryption as well as some related work. Chapter 4 presents the Methodology of this work. In Chapter 5 we present the results of the thesis: the benchmarks of the implemented schemes. This is followed by a discussion in Chapter 6 and last but not least, Chapter 7 concludes this thesis and outlines some future work directions.

2

Background

This chapter presents the required mathematical background as well as some fundamental statements on computational theory, cryptography and logic gates. The aim of this chapter is to state the most important definitions for the comprehension of the schemes and definitions for FHE.

2.1 Notation

Common mathematical notations used throughout this report are listed in Table 2.1.

Table 2.1: Mathematical notations used.

Symbol	**Meaning**
$\mathbb{Z}, \mathbb{R}, \mathbb{C}$	The sets of integers, the reals and the complex numbers
$\mathbb{Z}^{n\times m}, \mathbb{R}^{n\times m}, \mathbb{C}^{n\times m}$	The space of n times m matrices with integer, reals, complex and integer entries
$\mathcal{U}, \chi$	Uniform distribution $\mathcal{U}$ and error distribution χ.
$\mathbb{F}_q$	A finite field of q elements, where q is power of a prime
$\mathbb{F}_q^m$	A vector space of dimension m over $\mathbb{F}_q$, for a positive integer m
$\lVert\cdot\rVert, \lVert\cdot\rVert_\infty$	The Euclidean norm and the maximum norm
$\Lambda, \mathcal{L}$	A lattice
$\mathbb{L}(\mathbf{B})$	The lattice with the columns of the matrix $\mathbf{B}$ as basis
γ	Approximation factor in lattice problems
λ	The security parameter
n	The degree of a polynomial
m	Lattice dimension
$\mathbf{v}$	A vector $\mathbf{v} = (v_1, ..., v_n) \in \mathbb{Z}^n$
$\sum_{i=1}^n \lvert v_i\rvert$	The l_1 norm for vector $\mathbf{v} = (v_1, ..., v_n)$
$\mathbf{0}$	A zero vector (of appropriate size if nothing else is specified)
$\mathbf{1}$	A vector of ones (of appropriate size if nothing else is specified)
q, $\lceil q \rfloor$	The rounding of q to the next integer

2.2 Cryptology

The word cryptology is originally derived from the Greek words *kryptós* and *logos* meaning “hidden word”. Generally, cryptology is a science that studies how to hide confidential information. Cryptology is divided into two complementary fields, cryptography and cryptanalysis, where cryptography is the science of designing secure ciphers and cryptanalysis the science of breaking ciphers. In this thesis, we will focus on cryptography and more specifically on encryption schemes.

The goal of cryptography is to make any confidential information inaccessible for unauthorised parties, by providing some among the following properties:

- Confidentiality: The information cannot be understood by anyone for whom it was unintended.
- Integrity: The information cannot be altered in storage or transit between the sender and the intended receivers without the alteration being detected.
- Non-repudiation: The creator/sender of the information cannot deny at a later stage his or her intentions in the creation or transmission of the information.
- Authentication: The sender and the receiver can confirm each other’s identity and the origin/destination of the information.

Modern cryptography is heavily based on mathematical theory and computer science practice [24, 25, 26]. Cryptographic algorithms are designed around computational hardness assumptions, making such algorithms hard to break in practice by any adversary. Procedures and protocols that meet some or all of the above criteria are known as cryptosystems.

In cryptography, encryption is the process of encoding a message or information in such a way that only authorised parties can access it to ensure confidentiality. Imagine Alice wants to send a private message to Bob, but she is afraid that Eve will intercept the message and read the message that is only meant for Bob. In order to solve this issue, Alice encrypts the message in such a way so that only Bob is able to decrypt.

There are two types of encryption schemes: symmetric-key schemes and asymmetric-key schemes, also referred to as private-key and public-key schemes. The main difference between the two is that symmetric cryptosystems uses the same key for encrypting and decrypting while asymmetric cryptosystems uses different keys for encryption and decryption. One of the advantages of using symmetric encryption over asymmetric is that encryption and decryption are faster, but one drawback is that you have to exchange the key in a secure way. In this thesis, we will focus on the public-key setting in order to be able to outsource computations.

2.3 Computational Problems

Before defining complexity classes, it is crucial to understand the fundamental objects of computational complexity theory: A computational problem is a mathematical object representing a collection of questions that computers might want to solve. The input string for a computational problem is referred to as an instance. Respectively the output string is called the solution. A computational problem consists of an infinite amount of tuples which are composed of instances and the according solution. There are two major fields which deal with computational problems. First, there is the field of algorithm research which is the study of methods for solving the problems efficiently. Second, there is the field of complexity theory which explains why a problem is believed to be unsolvable or intractable even if a great deal or infinite computational resources are available. There are different types of computational problems of which the two most common will be further explained:

Search problem: A search problem consists of an infinite set of instances and a concise specification of valid solutions.

Example. Factoring a composite number is a search problem where the instance is a number n and the valid solution is a set of prime numbers $p_1, \ldots, p_n$ which are the prime factors to the number.

Decision problem: A decision problem consists of an infinite set of instances and a concise specification of YES-instances.

Example. Primality testing is a decision problem where the instance is a positive integer n and the problem is to determine if n is a prime number or not.

2.3.1 Complexity Classes

Computational complexity theory focuses on classifying computational problems according to their computational hardness. Similar to the following theory of public key systems, the field of computational complexity theory has developed rapidly in the past three decades, due to the fact that all cryptographic systems rely on the intractability of an infeasible underlying computational problem. The central task of computational complexity theory is whether tasks can be performed efficiently or not.

Intuitively, one can say that the time complexity of a problem is the number of steps that it takes to solve a special instance of the problem using the most efficient algorithm. In order to describe the time efficiency of the function the Big-O notation for an asymptotic upper bound. There are different options available on which resource should be focused on. The resource most often used is time. Another possible computational resource is memory (space). We call a function f:

- constant if $f = O(1)$
- logarithmic if $f = O(\log(x))$
- linear if $f = O(x)$
- polynomial if $f = O(x^c)$ for a $c > 0$
- exponential if $f = O(c^x)$ for a $c > 0$

One of the complexity classes is the complexity class P that is defined as follows:

Definition 1 P *is the class of decisional problems with solutions that run in time $O(n^c)$, for some constant c.*

In cryptography, we care about problems that are considered to *not* have efficient solutions, which means we do not care about problems in P. This is because when constructing an encryption scheme the goal is to prove that if a computational adversary can break the encryption scheme, then that adversary can also provide an efficient solution to the hardness assumption. Because of this reason these problems are well-suited to base encryption schemes on. To describe inefficient problems, we first define the complexity class NP:

Definition 2 NP *is the class of decisional problems with deterministic verifiers that run in time $O(n^c)$, for some constant c.*

NP-Hard are problems that are at least as hard as the hardest problems in NP and it is defined as follows:

Definition 3 *A problem is NP-hard if an algorithm for solving it can be translated into one for solving any NP-problem.*

When a decision problem is both in NP and NP-Hard it is NP-Complete and defined as follows:

Definition 4 *A problem which is both NP and NP-hard.*

What makes NP-Complete so interesting is that if any one of the NP-Complete problems was to be solved quickly, then all NP problems can in fact be solved quickly, i.e, in an efficient (polynomial) time.

The relationship of the complexity classes presented in this section can be seen in Figure 2.1.

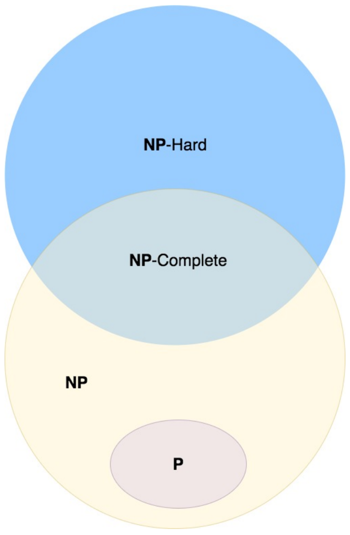

Figure 2.1: The relationships among the complexity classes NP, P, NP-Hard and NP-complete under the assumption that $\mathsf{P} \neq \mathsf{NP}$.

2.4 Public-Key Encryption

Public-key encryption, also known as asymmetric encryption, is a cryptographic primitive that allows users to privately exchange messages without a pre-established shared secret [27, 6]. Public-key encryption requires two separate keys: one public key, used for encryption and one private key for decryption. To ensure confidentiality, the message shall be encrypted with the receiver's public key. The resulting ciphertext shall and can only then be decrypted with the receiver's private key. An important property of the key pair is that the public key generation function takes the corresponding secret key as its input parameter. In other words, there is a mathematical relation between a party's public key and its corresponding secret key. If the scheme is secure it should be computationally unfeasible for an adversary to determine the private key knowing the public key, or to recover the plaintext knowing the public key and the corresponding ciphertext. Because of this, public keys can be freely shared, allowing users an easy and convenient method for encrypting content, and private keys can be kept secret, ensuring only the owners of the private keys can decrypt content.

Definition 5 *A public-key encryption scheme consists of the following 3 algorithms [27]:*

Key Generation. *KeyGen(1^λ) $\to$ (pk, sk): Outputs a pair of keys consisting of a public key* ***pk*** *and a secret key* ***sk*** *on the input security parameter λ.*

Encryption. *Enc(pk, μ) $\to$ c: Using the public key* ***pk****, encrypts a message $\mu \in \mathcal{M}$ into a ciphertext c, where the message space $\mathcal{M}$ is defined by the key space.*

Decryption. *Dec(sk, c) $\to$ μ: Using the secret key* ***sk****, decrypts a ciphertext c to recover the message $\mu \in \mathcal{M}$.*

As already stated, cryptographic algorithms are designed around computational hardness assumptions, making such algorithms hard to break in practice by any adversary with certain knowledge. The security provided by the use of a specific encryption is dependent on its average-case hardness, rather than the effort it would take to solve the worst-case. The hardness which is desired is either a proof, that the minimum steps needed for finding the solution is a very high number and so infeasible for an attacker to break, or a reduction to a NP-Hard problem under the assumption that P $\neq$ NP.

In the history of public-key encryption, the security assumptions of most of the cryptosystems have been relying on the integer factorisation problem and the discrete logarithm problem. The fact that it is easy to multiply together two large primes but it is hard to factor a large integer is what we call the integer factorisation problem. An algorithm that efficiently factors an arbitrary integer would break all cryptographic schemes that rely on this assumption. The discrete logarithm problem refers to the difficulty of computing x given only g and g^x, where x is an integer and g any element of a group G. The hardness of finding discrete logarithms depends on the groups. As with the integer factorisation problem, the inverse problem of the discrete exponentiation is not difficult to compute.

We call a cryptosystem computationally secure, if the practical side of attacking the cryptographic system is infeasible, while all its algorithms (KeyGen, Enc, Dec) are efficiently computable. Otherwise we could have a scheme that is super secure, nut it is impossible to encrypt/decrypt as it would require to solve NP-hard problems. In the setting of provable security, breaking a system's security is proven to be as difficult as solving a well-known problem which is thought to be hard.

2.5 Homomorphic Encryption

Traditional encryption requires a receiver to decrypt a message in order to perform operations on it. In contrast, homomorphic encryption additionally allows one to directly perform computations on the ciphertext. When decrypting this ciphertext the resulting plaintext will match the result of performing the computation on the corresponding plaintext.

The most common definition is the following: Let $\mathcal{M}$ denote the set of the plaintexts and $\mathcal{C}$ denote the set of the ciphertexts. Let $\odot_M$ and $\odot_C$ denote the operations in the plaintext and ciphertext spaces respectively. An encryption scheme is said to be homomorphic if for any given encryption key k the encryption function E satisfies

$$\forall m_1, m_2 \in M,\ E(m_1 \odot_M m_2) \leftarrow E(m_1) \odot_C E(m_2)$$

for some operators $\odot_M$ in M and $\odot_C$ in C, where $\leftarrow$ means "can be directly computed from", that is, without any intermediate decryption [28].

Definition 6 *A homomorphic public-key encryption scheme* ***HE = (KeyGen, Enc, Dec, Eval)*** *with message space $\mathcal{M}$ consists of the following 4 Polynomial Time Algorithm (PPT) algorithms:*

- ***KeyGen(1^λ) $\rightarrow$ (pk, sk)****: Outputs a public encryption key* ***pk*** *and a secret decryption key* ***sk****.*
- ***Enc(pk, μ) $\rightarrow$ c****: Using the public key* ***pk****, encrypts a message $\mu \in \mathcal{M}$ into a ciphertext c.*
- ***Dec(sk, c) $\rightarrow$*** *μ: Using the secret key* ***sk****, decrypts a ciphertext c to recover the message $\mu \in \mathcal{M}$.*
- ***Eval(C,($c_1, ..., c_l$),pk) $\rightarrow$ ĉ****: Using the public key* ***pk****, applies a circuit C: $\mathcal{C}^l \rightarrow \mathcal{C}$ to $c_1, ..., c_l$, and outputs a ciphertext $\hat{c}$.*

Homomorphic cryptosystems are classified into two main categories:

- Partially homomorphic cryptosystem: An encryption scheme that is either multiplicative or additive homomorphic.
- Somewhat homomorphic cryptosystem: An encryption scheme that support only a limited number of homomorphic operations, e.g., only additive homomorphism, or multiplicative homomorphism up to a certain degree.
- Fully homomorphic cryptosystem: An encryption scheme that supports arbitrary computation.

In the rest of the section we analyse different types of partially HE, we refer to Chapter 3 for a deep explanation of FHE.

A cryptosystem is considered partially homomorphic if it supports either additive or multiplicative homomorphism, but not both. Clearly, this is enough for some systems and in addition the efficiency of some homomorphic encryption schemes is even high enough for practical applications. However, some situations require more operations than only one type of operation, for instance if one wishes to compute a point on a parabola, $f(x) = ax^2 + bx + c$.

A cryptosystem that supports arbitrary computations on ciphertext is far more powerful than one that supports only addition or multiplication. These more powerful cryptosystems are known as FHE [9]. FHE would allow one to compute a point on a parabola among many other constructions of programs without ever having to decrypt at all in theory. In practice, we are not there yet [19].

2.5.1 Partially Homomorphic Encryption

There are two possible homomorphisms, namely the multiplicative and additive homomorphism. This implies that there exists a group structure, that is preserved by the encryption and decryption. If an encryption scheme allows either one of these operations, but only one, it is called a partially homomorphic encryption scheme. A selection of some partially homomorphic cryptosystems can be seen in Figure 2.2 and we will now show the homomorphic properties for Unpadded RSA and ElGamal.

Partially Homomorphic Cryptosystem:

- Unpadded RSA [6]
- ElGamal [29]
- Goldwasser-Micali [12]
- Benaloh [30]
- Paillier [11]
- Okamoto-Uchiyama [31]
- Naccache-Stern [32]
- Damgård-Jurik [33]
- Boneh-Goh-Nissim [10]
- Ishai-Paskin [34]

Figure 2.2: Some partially homomorphic cryposystems.

Definition 7 *The RSA [6, 27] scheme consists of three algorithm (KeyGen, Enc, Dec), defined as follows:*

- ***KeyGen(λ) → (pk, sk):***
  1. *Generate two distinct λ-bit primes p and q, compute $N = pq$ and $\phi(N)$.*
  2. *Choose an integer $e \xrightarrow{R} \mathbb{Z}_{\phi(N)}$ such that $GCD(e, \phi(N)) = 1$ and compute its (modular) inverse $d = e^{-1} \mod \phi(N)$.*
  3. *Set:* $\mathbf{pk} = (N, e)$ *and* $\mathbf{sk} = (N, d)$
- ***Enc(pk, m) → c :*** *Compute* $c = m^e \mod N$.
- ***Dec(sk, m) → m :*** *Compute* $m = c^d \mod N$.

RSA has the multiplicative homomorphic property which means we we can multiply two ciphertexts in order to receive multiplication of the underlying plaintexts.

Let us encrypt two messages m_1 and m_2 with the same key:

$$\mathbf{Enc}(m_1) = m_1^e \mod N \tag{2.1}$$

$$\mathbf{Enc}(m_2) = m_2^e \mod N \tag{2.2}$$

The homomorphic property is then defined as follows:

Definition 8 *(RSA homomorphic property)*

$$\mathbf{Enc}(m_1) \cdot \mathbf{Enc}(m_2) = m_1^e m_2^e \mod N = (m_1 m_2)^e \mod N = \mathbf{Enc}(m_1 \cdot m_2) \tag{2.3}$$

Definition 9 *The ElGamal [13, 27] scheme consists of three algorithm (KeyGen, Enc, Dec), defined as follows:*

- ***KeyGen(λ) $\rightarrow$ (pk, sk):***
  1. *Generate the description of a cyclic group $G =< g >$ of order q (where q is a λ-bit long integer).*
  2. *Choose a random value $x \overset{R}{\leftarrow} \{1, 2, \cdots, q-1\}$ and compute $h = g^x \in G$.*
  3. *Set $\mathbf{sk} = x$ and $\mathbf{pk} = (G, g, q, h)$.*
- ***Enc(pk, m) $\rightarrow c = (c_1, c_2)$:*** *Generate a random value $r \overset{R}{\leftarrow} \{1, 2, \cdots, q-1\}$ and compute $c_1 = g^r \in G$ and $c_2 = mh^r \in G$.*
- ***Dec(sk, m) $\rightarrow$ m :*** *First compute $k = c_1^x \in G$ and then $m = c_2 k^{-1} \in G$.*

As RSA, ElGamal also has the multiplicative homomorphic property. Let us encrypt two messages m_1 and m_2 with the same key:

$$\mathbf{Enc}(m_1) = (g^{r_1}, m_1 \cdot h^{r_1}) \tag{2.4}$$

$$\mathbf{Enc}(m_2) = (g^{r_2}, m_2 \cdot h^{r_2}) \tag{2.5}$$

The Homomorphic property is then defined as follows [27]:

Definition 10 *(ElGamal homomorphic property)*

$$\begin{aligned}\mathbf{Enc}(m_1) \cdot \mathbf{Enc}(m_2) &= (g^{r_1}, m_1 \cdot h^{r_1})(g^{r_2}, m_2 \cdot h^{r_2}) \\ &= (g^{r_1+r_2}, (m_1 \cdot m_2)h^{r_1+r_2}) = \mathbf{Enc}(m_1 \cdot m_2)\end{aligned} \tag{2.6}$$

2.5.2 Somewhat Homomorphic Encryption

A homomorphic encryption scheme can also be categorised into somewhat homomorphic or fully homomorphic based on the range of functions it can be applied to.

A somewhat homomorphic encryption scheme, is a scheme that support only a limited number of homomorphic operations, e.g., only additive homomorphism, or multiplicative homomorphism up to a certain degree. Fully homomorphic encryption schemes do not have any constraint regarding the circuit depth and can evaluate arbitrary functions [9].

When considering this property of fully homomorphic encryption it should make it a powerful tool in constructing various privacy preserving systems but in reality it comes with a heavy overhead which makes it unpractical [35]. With regard to the efficiency aspect, somewhat homomorphic encryption is far ahead of fully homomorphic encryption but with the possible drawback of supporting only a limited number of homomorphic operations [36].

2.5.3 Circuits

The construction of any cryptographic primitive including FHE relies on the fact that you can represent any function with a circuit. A circuit consists of a set of basic logic gates which operate on the inputs to give a certain output. In order to evaluate a function Y, express Y as a circuit and topologically arrange its gates into levels which will be executed sequentially.

The complexity of a circuit is expressed as its depth. A circuit can be viewed as a directed acyclic graph in which each node represents a gate. The circuit depth is then defined as the maximum length from any input to any output.

Definition 11 *The size of a circuit C is the number of its non-input gates. The depth of a circuit C is the length of its longest path, from an input gate to the output gate, of its underlying directed graph.*

Example. Assume the function Y outputs the expression $A \cdot B + B \cdot C \cdot (B + C)$ on input (A, B, C). Then the following circuit represents the function Y, with the logic gates AND and OR which can be seen in Figure 2.3.

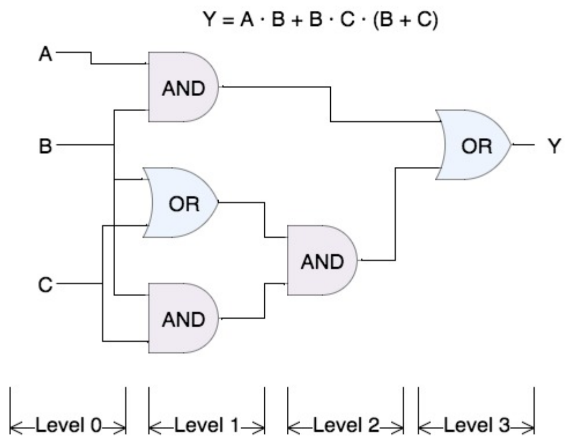

Figure 2.3: The function Y represented as a circuit.

Furthermore, by being able to perform addition and multiplication, we can represent basic logical operations as described in Table 2.2, where a and b represents one bit.

Table 2.2: Implementation of logical operations.

operation	symbol	arithmetic implementation
and	$a \wedge b$	ab
or	$a \vee b$	$a + b - ab$
not	$\neg a$	$1 - a$
nand	$a \bar{\wedge} b$	$1 - ab$
xor	$a \oplus b$	$a + b - 2ab$

It is a well known fact that it is possible to construct any circuit by only using NAND gates which means that if we can construct a NAND gate we can evaluate any circuit and thus compute any function.

Table 2.3: Truth table for NAND gate.

a	b	$a \bar{\wedge} b$
0	0	1
0	1	1
1	0	1
1	1	0

2.6 Quantum Era

Today, the security of many encryption schemes is endangered by the prospect of quantum computing. In 1994, Shor developed one quantum algorithm that can be used to factor integers and solve the discrete logarithm problem [37]. This means that all cryptographic schemes based on these assumptions will not reach the same level of security with the presence of quantum computers. Among the affected schemes there are: RSA and ElGamal. Breaking these would have potentially severe consequences for privacy and security as they are used to protect many types of sensitive data. Therefore cryptographers need to design new protocols based on entirely different ideas and assumptions. Possible research directions that are believed to be quantum secure are:

- Hash-based signatures,
- Code-based cryptography,
- Multivariate cryptography,
- Lattice-based cryptography.

In Table 2.4 some classical cryptosystems are listed along with their current status of security in relation to quantum computers. In this thesis, we study lattice-based cryptography that is believed to be resistant against quantum computers which means that nobody has figured out a way to attack it yet.

Table 2.4: Status of security of some classical cryptosystems in relation to quantum computers.

Cryptosystem	Broken by Quantum Algorithms?
RSA [6]	Broken
Diffie-Hellman [38]	Broken
Elliptic curve [39, 40]	Broken
McEliece [41]	Not broken yet
NTRU [5]	Not broken yet
Lattice-based [42]	Not broken yet

2.7 Lattice-based Cryptography

Lattice-based cryptography appears to be one of the most promising candidates for post-quantum cryptography. There are mainly two reasons for the confidence in the long-lasting security of lattice-based cryptography. Firstly, many problems in lattice-theory are proven to be NP-Hard [43]. Secondly, the security of many lattice problems have a worst-case to average-case reduction [42]. This means that picking any random instance of the problem will be as hard as solving the worst case.

2.7.1 Lattices

A lattice is a set of points in n-dimensional space with a periodic structure [44, 45]. Let $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ be a set of n linearly independent vectors in $\mathbb{R}^m$ [45]. The lattice generated by $\mathbf{B}$ will then be the set of all integer linear combinations of the vectors in $\mathbf{B} : \mathcal{L}(\mathbf{B}) = \{\sum_{i=1}^{n} x_i \mathbf{b}_i | x_i \in \mathbb{Z}\}$ [44, 46, 47, 45]. This gives the definition [47]:

$$\mathcal{L}(B) = \{\vec{x} \times \mathbf{B} : \vec{x} \in \mathbb{Z}^n\} \tag{2.7}$$

A full-rank lattice basis $\mathbf{B}$ is defined as a set of n linearly independent vectors in a vector space of dimension n. A lattice is full-rank lattice if $n = m$ [45]. This gives the definition [47]:

$$\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}, \quad \mathbf{b}_i \in \mathbb{R}^n \tag{2.8}$$

An example of a lattice is shown in Figure 2.4, that represents a lattice in $\mathbb{R}^2$.

Ideal lattices, as Gentry's scheme [9] was based on, are lattices with some additional structure than traditional lattices, namely the structure of an ideal. While lattices have a group structure, ideal lattices have an ideal structure as the term suggests.

In lattice-based cryptography, some problems are easy to solve using bases of a particular structure. We refer to bad basis as those in which it is generally no easier than a random basis to solve a particular lattice problem. Good bases are referred as those in which a given problem is easy to solve. For Lattice based FHE schemes the public key is a "bad" basis while the secret key is a "good" basis for the same

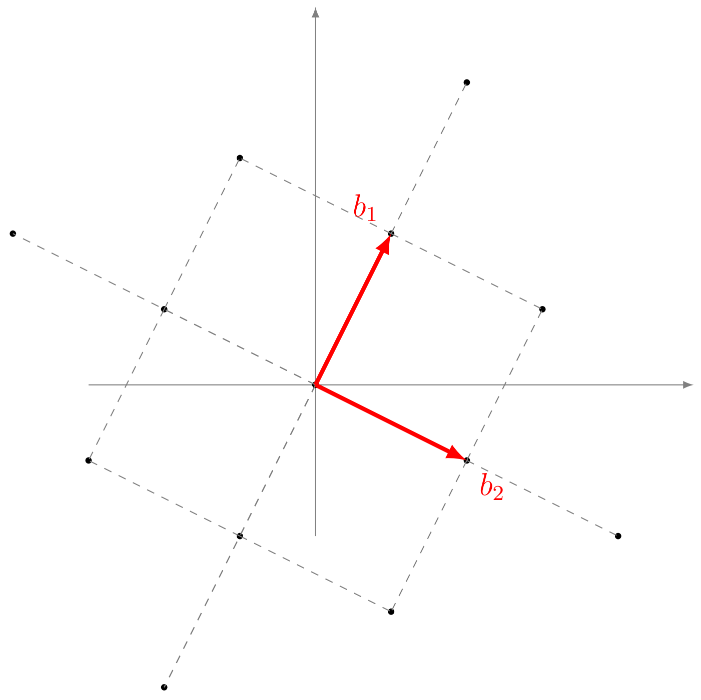

Figure 2.4: A lattice in $\mathbb{R}^2$.

lattice [48]. Typically, a good basis consists of vectors that are short and close to orthogonal [49, 48].

2.7.2 Lattice Problems

This section reviews some common hard problems that are used as the foundation for a number of lattice-based cryptographic schemes. The problems deal with finding short vectors and the closest vectors in a lattice.

2.7.2.1 SVP & CVP

The Shortest Vector Problem has been studied intensively, and appears to be intractable in general, even including quantum algorithms.

The Shortest Vector Problem (SVP) asks for a nonzero vector whose Euclidean norm is minimal among all other nonzero lattice vectors, i.e a short or the shortest vector. This is considered easy in a 2-dimensional lattice but it becomes hard to solve for several dimensions.

Definition 12 *(SVP) Given an arbitrary basis* $\mathbf{B}$ *for an* n*-dimensional lattice* $\mathcal{L} = \mathcal{L}(\mathbf{B})$*, compute a non-zero vector* $\boldsymbol{v} \in \mathcal{L}$*, such that* $\|\mathbf{v}\| = \lambda_1$*. [50, 51, 46]*

The closest vector problem is a generalisation of the shortest vector problem. It has been shown that Closest Vector Problem (CVP) is at least as hard as SVP. [52]

CVP asks for a vector that is not too far from a specific target point, it does not necessarily have to be the closest one.

Definition 13 *(Closest Vector Problem (CVP)) Given an arbitrary lattice basis* $\mathbf{B}$ *for an* n*-dimensional lattice and some target point* $\mathbf{t} \in \mathbb{R}^n$*, compute* $\mathbf{v} \in \mathcal{L}$ *such that* $\|\mathbf{t} - \mathbf{v}\|$ *is minimal. [50, 26]*

There are two main differences between SVP and CVP. One of the differences is that SVP asks for a lattice point near zero, while CVP asks for a lattice point close to an arbitrary point in the space. The other one is that in CVP the solution can be the all zero vector while in SVP it must not. This means it is not possible to use CVP to find the shortest vector close to the origin in order to solve SVP since it would return the zero vector.

The hardness of solving SVP and CVP has led scientists to consider approximation versions of these problems, which is particularly applicable to cryptography. Approximation algorithms return solutions that are only guaranteed to be within some specified factor γ from the optimum.

In the γ-approximation SVP (SVP$_\gamma$), this means finding a non-zero lattice vector at distance of at most $\gamma\lambda_1(\mathcal{L})$, for $\gamma = \gamma(n) \geq 1$.

Definition 14 *(Approximate Shortest Vector Problem (SVP$_\gamma$)) Fix* $\gamma > 1$*. Given an arbitrary basis* $\mathbf{B}$ *for an* n*-dimensional lattice* $\mathcal{L}$*, compute a non-zero vector* $\boldsymbol{v} \in \mathcal{L}$ *such that* $\|\mathbf{v}\| \leq \gamma \cdot \lambda_1$*. [50, 46]*

In the γ-approximation CVP (CVP$_\gamma$), this means finding a lattice vector at distance of at most γ, for $\gamma = \gamma(n) \geq 1$.

Definition 15 *(Approximate Closest Vector Problem (CVP$_\gamma$)) Fix* $\gamma > 1$*. Given an arbitrary basis* $\mathbf{B}$ *for an* n*-dimensional lattice* $\mathcal{L} = \mathcal{L}(\mathbf{B})$ *and some target point* $\mathbf{t} \in \mathbb{R}^n$*, compute* $\mathbf{v} \in \mathcal{L}$ *such that* $\|\mathbf{t} - \mathbf{v}\| \leq \gamma\|\mathbf{t} - \mathbf{xB}\|$ *for all* $\mathbf{x} \in \mathbb{Z}^m$*. [50]*

The following problem is the decision variant of approximating the shortest vector in a given lattice within a factor γ.

Definition 16 *(Gap Shortest Vector Problem (GapSVP$_\gamma$)) Given* $\mathbf{B}$ *and* r*, decide whether* $\lambda(\mathbf{B}) \leq r$*, or if* $\lambda(\mathbf{B}) \geq \gamma \cdot r$*. (Instances where* $r < \lambda(\mathbf{B}) < \gamma \cdot r$ *are not considered.) [46]*

The following problem is the decision variant of approximating the closest vector in a given lattice within a factor γ.

Definition 17 *(Gap Closest Vector Problem (GapCVP$_\gamma$)) Given* $\mathbf{B}$*,* $x \in \mathbb{R}^n$ *and* r*, decide whether* $dist(x, L) \leq r$*, or if* $dist(x, L) \geq \gamma \cdot r$*. (Again, instances in the middle of* r *and* $\gamma \cdot r$ *are not considered.) [46]*

Variants of this approximate problem are typically used to prove the security of cryptosystems. Micciancio proved that SVP is NP-Hard to solve even approximately, for any approximation factor up to $\sqrt{2}$ [53]. Furthermore, the decision problem associated to CVP is NP-complete which means no algorithm can solve CVP in deterministic polynomial time, provided that $\mathsf{P} \neq \mathsf{NP}$.

2.7.2.2 Other Lattice Problems

The Bounded Distance Decoding Problem is a problem similar to CVP.

Definition 18 *(Bounded Distance Decoding Problem (BDD)) Given an arbitrary basis* $\mathbf{B}$ *for an* n*-dimensional lattice* $\mathcal{L} = \mathcal{L}(\mathbf{B})$ *and a target point* $\mathbf{t} \in \mathbb{R}^n$ *with the guarantee that* $\exists \mathbf{v} \in \mathcal{L}$ *such that* $\|\mathbf{v} - \mathbf{t}\| < d = \lambda_1(\mathcal{L}/2)$*, find* $\mathbf{v}$*. [46]*

Ajtai showed that if it is possible to efficiently solve the Short Integer Solutions Problem (SIS) it is also possible to efficiently find short vectors in every n-dimensional lattice [42]. Due to this fact, SIS became an important building block for lattice-based cryptography.

Definition 19 *(Short Integer Solutions Problem (SIS)) Given* $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$*, find* $\mathbf{e} \in \mathbb{Z}^m$ *satisfying* $\mathbf{A}\mathbf{e} = 0 \mod q$ *and* $\|\mathbf{e}\|_2 \leq \alpha$*.*

2.7.2.3 Learning With Errors

In 2005, Regev defined the Learning With Errors (LWE) problem [24], a generalisation of the Learning Parity with Noise (LPN) problem. The LPN problem is equivalent to the problem of decoding random linear codes. It is an extensively studied problem that is believed to be hard [24]. Regev showed that his public-key cryptosystem based on the hardness of LWE was much more efficient than other proposed public-key cryptosystems based on unique-SVP, a special case of SVP. He also proves the hardness of LWE with a quantum reduction from worst-case lattice problem SVP, where a quantum reduction is a reduction algorithm which uses quantum computing. It has therefore become an important building block in modern cryptographic systems and a popular topic in present-day research. The problem with LWE is that it is very inefficient due to an inherent quadratic overhead. Its public key size is $O(mn \log q) = \tilde{O}(n^2)$ and it additionally increases the size of the message by a factor of $O(n \log q = \tilde{O}(n)$ by every encryption [24].

Definition 20 *(Decisional Learning With Errors (DLWE)). Let* n *and* q *be positive integers, and* χ *an error distribution over* $\mathbb{Z}$*. Let* $\mathbf{s}$ *be a uniformly random vector in* $\mathbb{Z}_q^n$*. The DLWE is to distinguish* $A_{s,\chi}$ *from the uniform distribution* U *from* m *independent samples* $(\mathbf{a_i}, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ *where every sample is distributed according to either:* $A_{s,\chi}$ *or the uniform distribution [46, 24].*

Definition 21 *(Search Learning With Errors). Let* n *and* q *be positive integers, and* χ *an error distribution over* $\mathbb{Z}$*. Let* $\mathbf{s}$ *be a uniformly random vector in* $\mathbb{Z}_q^n$*. The search LWE is to find* $\mathbf{s}$ *from* $\mathbf{m}$ *independent samples* $(\mathbf{a_i}, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ *drawn from* $A_{s,\chi}$ *[46, 24].*

2.7.2.4 Ring Learning With Errors

To deal with the inefficiency of LWE, Lyubashevsky, Peikert and Regev introduced the Ring Learning With Errors (RLWE) problem [54]. In most cases n noisy LWE equations can be replace with only one noisy RLWE equation which obviously improves in terms of efficiency.

RLWE is an algebraic variant of LWE, i.e. LWE over ideal lattices which are more structured than random lattices. Mathematically speaking, it can be interpreted as replacing the group $\mathbb{Z}_q^n$ with the ring $\mathbb{Z}_q[x]/\langle x^n + 1\rangle$ [24, 54].

Definition 22 *(RLWE). Consider the ring $R = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$ with n as power of 2 and an error distribution χ over R. Let $\mathbf{s}$ be uniformly random sampled from $\mathbb{R}_q$. The decision RLWE is to distinguish $A_{s,\chi}$ from the uniform distribution $\mathbf{R}$ from m independent samples $(\mathbf{a_i}, b_i) \in \mathbb{R}_q \times \mathbb{R}_q$ where every sample is distributed according to either: $A_{s,\chi}$ or the uniform distribution [46, 54].*

3

Fully Homomorphic Encryption

This chapter presents the concepts of FHE in the single and multi key setting as well as details of the implemented schemes.

3.1 Single-key Fully Homomorphic Encryption

If we are satisfied with an encryption scheme that supports homomorphic operations of some specific operation one might question that homomorphic encryption seems easy to realise. However, some situations require more operations than only one type. If we again consider the case of cloud computing one typical reason for outsourcing computations is because it tend to be heavy and complex. This is obviously not achieved by only one specific operation which is why we need homomorphic encryption schemes that support arbitrary operations.

As already stated, a cryptosystem that supports arbitrary computation on ciphertexts in a meaningful way is known as FHE [9].

Definition 23 *A FHE (public-key) encryption scheme* ***HE = (KeyGen, Enc, Dec, Eval)*** *with message space $\mathcal{M}$ consists of the following 4 PPT algorithms [18]:*

- ***KeyGen(1^λ) → (pk, sk)****: Outputs a public encryption key* **pk** *and a secret decryption key* **sk***.*
- ***Enc(pk, μ) → c****: Using the public key* **pk***, encrypts a message $\mu \in \mathcal{M}$ into a ciphertext c.*
- ***Dec(sk, c) → μ****: Using the secret key* **sk***, decrypts a ciphertext c to recover the message $\mu \in \mathcal{M}$.*
- ***Eval(C,($c_1, ..., c_l$),pk) → ĉ****: Using the public key* **pk***, applies a circuit C: $\mathcal{C}^l \to \mathcal{C}$ to $c_1, ..., c_l$, and outputs a ciphertext $\hat{c}$, where $\mathcal{C}$ can have any kind of gates.*

FHE is like encryption where we have an extra algorithm **Eval**, which enables any third party holding the public key to run an unlimited number of operations on the encrypted data.

Definition 24 *(Correctness) The scheme* ***HE = (KeyGen, Enc, Dec, Eval)*** *is correct for a given l-input circuit $\mathcal{C}$ if, for any key-pair* ***(pk, sk)*** *output by* ***KeyGen(1^λ)****, any l plaintexts $m_1, \cdots, m_l$ and any ciphertexts $\boldsymbol{c} = (c_1, \cdots, c_l)$ with*

*$c_i \leftarrow$ **Enc(pk,** m_i**)**, it holds that [15]:*

$$\textbf{\textit{Dec(sk, Eval(}}\mathcal{C}\textbf{\textit{,c,pk))}} = \mathcal{C}(m_1, \cdots, m_l) \tag{3.1}$$

A selection of some fully homomorphic cryptosystems can be seen in Figure 3.1 and the one relevant for this thesis [3] will be explained further in the following section.

Fully Homomorphic Cryptosystem:

- Gentry's [9]
- FHE over the integers [15]
- Brakerski-Gentry-Vaikuntanathan (BGV) [1]
- Brakerski's scale-invariant [55]
- NTRU-based (LTV) [4]
- Gentry-Sahai-Waters (GSW) [3]

Figure 3.1: Some fully homomorphic cryptosystems.

3.1.1 GSW

Imagine that we have two matrices $\mathbf{C_1}$ and $\mathbf{C_2}$ that have the same eigenvector $\mathbf{s}$. Would it possible to create an encryption scheme where $\mathbf{s}$ represents the secret key, $\mathbf{C}$ the ciphertext and m the message in form of an eigenvalue that corresponds to the eigenvector? If the eigenvector is kept secret, the message should be hidden and when we know the eigenvector it should possible to recover the message. Intuitively, the answer would be no as finding an eigenvalue can be solved by Gaussian elimination in polynomial time. However, Gentry, Sahai and Waters [3] took this idea further and instead of using eigenvectors as secret keys they used approximate eigenvectors. As long as the noise vector has a lower norm than the modulus we work with the ciphertext will be decryptable. By relaxing the condition, it becomes a hard problem to solve where the hardness derives from the LWE problem. Gentry, Sahai and Waters were able to create an encryption scheme from a simple idea that has been the base for several others [7, 2, 20, 21].

The scheme also uses a special gadget matrix $\mathbf{G}$ where $\mathbf{G}^{-1}(\cdot)$ is a function. A gadget matrix is a matrix with powers of two in the diagonal [3].

The scheme consists of the following algorithms, that is directly taken from [3, 7]:

- **params $\leftarrow$ GSW.SetUp(1^λ, 1^d):** Choose a lattice dimension parameters $n = n(\lambda, d)$ and B_χ-bounded error distribution $\chi = \chi(\lambda, d)$ and a modulus q of size $q = B_\chi 2^{\omega(d\lambda \log \lambda)}$ such that $\text{LWE}_{n-1,q,\chi,B_\chi}$ holds. Choose $m = n\log(q) + \omega(\log \lambda)$. Finally choose a random matrix $\mathbf{B} \in \mathbb{Z}_q^{n-1\times m}$. Output **params** $:= (q, n, m, \chi, B_\chi, \mathbf{B})$.
- $(sk, pk) \leftarrow$ **GSW.KeyGen(params):**
  - Sample $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^{n-1}$. Output $sk = \mathbf{t} = (-\mathbf{s}, 1) \in \mathbb{Z}_q^n$.
  - Sample $\mathbf{e} \leftarrow \chi^m$. Set $\mathbf{b} := \mathbf{sB} + \mathbf{e} \in \mathbb{Z}_q^m$. Output $pk = \mathbf{A}$ where, $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ is defined as $\mathbf{A} := \begin{bmatrix}\mathbf{B}\\ \mathbf{b}\end{bmatrix}$.

- $\mathbf{C} \leftarrow \mathbf{GSW}.\mathbf{Encrypt}(pk, \mu)$: Choose a short random matrix as the randomness $\mathbf{R} \overset{\$}{\leftarrow} \{0,1\}^{m \times m}$. Then output the encryption of message $\mu \in \{0,1\}$ as $\mathbf{C} \in \mathbb{Z}_q^{n \times m}$ where,

$$\mathbf{C} := \mathbf{AR} + \mu\mathbf{G}$$

- $\mu' \leftarrow \mathbf{GSW}.\mathbf{Decrypt}(sk, \mathbf{C})$: Let $\mathbf{t} := sk$. Define a vector $\mathbf{w} \in \mathbb{Z}_q^n$ as follows:

$$\mathbf{w} = [0, \ldots, 0, \lceil q/2 \rceil]$$

  Then compute $v = \mathbf{t}\mathbf{C}\mathbf{G}^{-1}(\mathbf{w}^T) \in \mathbb{Z}_q^m$. Finally output $\mu' = |\lfloor \frac{v}{q/2} \rceil|$ as the decrypted message.

- On input two ciphertexts $\mathbf{C}_1, \mathbf{C}_2 \in \mathbb{Z}_q^{n \times m}$ we can define homomorphic addition, multiplication:
  - $\mathbf{GSW}.\mathbf{Add}(\mathbf{C}_1, \mathbf{C}_2)$: Output $\mathbf{C}_1 + \mathbf{C}_2 \in \mathbb{Z}_q^{n \times m}$.
  - $\mathbf{GSW}.\mathbf{Mul}(\mathbf{C}_1, \mathbf{C}_2)$: Output the matrix product $\mathbf{C}_1\mathbf{G}^{-1}(\mathbf{C}_2) \in \mathbb{Z}_q^{n \times m}$.

  This also allows us to compute a homomorphic NAND gate by outputting $\mathbf{G} - \mathbf{C}_1\mathbf{G}^{-1}(\mathbf{C}_2)$.

3.2 Multi-key Fully Homomorphic Encryption

A motivation for FHE is the ability to be able to encrypt data locally but still outsource the computation of the encrypted data without risking exposing the actual data. FHE can only handle this in a single user setting where the ciphertexts are encrypted under the same key. In order to be able to compute a function on ciphertexts encrypted under different public keys multi-key FHE was introduced by Lopéz-Alt et al. [4]. All the secret keys of the parties involved are needed in order to decrypt the ciphertext after the computation [4, 56, 23, 2, 7]. In other words, the parties involved need to jointly decrypt the ciphertext to obtain the output.

3.2.1 Multi-key GSW

The GSW scheme does not support multi-key by nature but it is possible to convert the GSW FHE into a multi-key FHE. In 2014, Clear and McGoldrick showed how to extend the GSW scheme into multi-key FHE resulting in the first multi-key FHE based on LWE [56]. In 2016, this work was simplified and improved further by Mukherjee and Wichs [7] resulting in a Single-Hop Multi-Key (SHMK) FHE scheme. The property of single-hop in this scheme means that all relevant keys must be known at the start of the homomorphic computation and the output cannot be combined with ciphertexts encrypted under other keys in a useful way without a bootstrapping step being performed. This means that in both of these works all input needed to be known in advance before the computation starts. In 2016, this requirement was removed by Brakerski and Perlman [2] when they showed how to extend the prior work to support an unbounded number of homomorphic operations for an unbounded number of parties. In their work input from new parties can be introduced into the computation dynamically. Additionally, they also improved the length of the ciphertexts and the space complexity of an atomic operation. This scheme is called Fully Dynamic Multi-Key (FDMK) FHE scheme. The fact that input from new parties can be introduced into the computation dynamically is what makes the scheme dynamic. This is achieved via bootstrapping that was introduced by Gentry [9].

Single-Hop Multi-Key (SHMK) FHE:

The scheme consists of the following algorithms, directly taken from [7]:

- $\mathbf{SHMK}.\mathbf{Keygen}(\mathbf{params})$: Run the key-generation algorithm of GSW to generate:

$$(sk, pk) \leftarrow \mathbf{GSW}.\mathbf{KeyGen}(\mathbf{params})$$

- $\mathbf{SHMK}.\mathbf{Encrypt}(pk, \mu)$: Execute the following steps:
  - $(\mathcal{U}, \mathbf{C}) \leftarrow \mathbf{UniEnc}(\mu, pk)$. On input a message $\mu \in \{0,1\}$ and a GSW public key pk it generates a pair $(\mathcal{U}, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{n\times m}$ and $\mathcal{U} \in \{0,1\}^*$
  - Output the pair $c := (\mathcal{U}, \mathbf{C})$ as the ciphertext for μ [7]
- $\mathbf{SHMK}.\mathbf{Expand}((pk_1, \ldots, pk_N), i, c)$: On receiving a sequence of public-keys $(pk_1, \ldots, pk_N)$ and a fresh ciphertext $c = (\mathcal{U}, \mathbf{C})$ under the public key pk_i run the Extend algorithm for all pk_j where $i \neq j$.
  - For $j \in \{pk_1, \ldots, pk_N\} \backslash \{i\}$, compute $\mathbf{X}_j \leftarrow \mathbf{Extend}(\mathcal{U}, pk_i, pk_j)$.
  - Then define a matrix $\mathbf{C}_b \in \mathbb{Z}_q^{nN\times mN}$ as a concatenation of N^2 sub-matrices where each sub-matrix $\mathbf{C}_{a,b} \in \mathbb{Z}_q^{n\times m}$ for $a, b \in [N]$ is defined as:

$$C_{a,b} := \begin{cases} \mathbf{C} & \text{when } a = b \\ \mathbf{X}_j & \text{when } a = i \neq j \text{ and } b = j \\ \mathbf{0}^{n\times m} & \text{otherwise} \end{cases}$$

  Finally output $\hat{c} := \hat{\mathbf{C}}$ as the new expanded ciphertext.
- $\mathbf{SHMK}.\mathbf{Eval}(\mathbf{params}, \mathcal{C}, (\hat{c}_1, \ldots, \hat{c}_l))$: On input l expanded ciphertexts simply use the GSW homomorphic evaluation algorithms namely **GSW.Add** and **GSW.Mult**, albeit with expanded dimensions $n' = nN$ and $m' = mN$ and the expanded $\hat{\mathbf{G}}_N, \hat{\mathbf{G}}_N^{-1}$ (in place of n, m and $\mathbf{G}, \mathbf{G}^{-1}$).
- $\mathbf{SHMK}.\mathbf{Decrypt}(\mathbf{params}, (sk_1, \ldots, sk_N), c)$: On input a ciphertext $c = \hat{\mathbf{C}}$ and the sequence of secret keys $(sk_1, \ldots, sk_N)$ parse each $\mathbf{t}_i := sk_i$ and then construct the joint secret key by horizontally appending all the secret-keys in sequence $\hat{\mathbf{t}} = [\hat{\mathbf{t}}_1 \hat{\mathbf{t}}_2 \cdots \hat{\mathbf{t}}_N] \in \mathbb{Z}_q^{nN}$. Then run the GSW decryption algorithm albeit with expanded dimensions $n' = nN$ and $m' = mN$ and the expanded $\hat{\mathbf{G}}_N, \hat{\mathbf{G}}_N^{-1}$ (in place of n, m and $\mathbf{G}, \mathbf{G}^{-1}$)

The new property of the SHMK scheme is the **Expand** algorithm that makes it possible to convert single-key GSW into multi-key which means we can perform meaningful operations on ciphertexts encrypted under different public keys. The two algorithms (**UniEnc**, **Expand**) act as a masking scheme for GSW. What **UniEnc** essentially does is to create helper information by encrypting each entry of the randomness matrix as a separate GSW ciphertext.

Fully Dynamic Multi-Key (FDMK) FHE:

In comparison with the prior presented multi-key FHE schemes this scheme supports an unbounded number of homomorphic encryptions for an unbounded number of parties. This is interesting since it means that inputs from new parties can be introduced into the computation dynamically whereas in previous schemes the final set of parties needed to be known in advance.

The scheme consists of the following algorithms, that is directly taken from [2]:

- **FDMK.Keygen(params):** Generate a secret key as in the SHMK scheme - sample uniformly at random $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^{n-1}$, set and output:

  $$sk := \mathbf{t} = (-\mathbf{s}, 1) \in \mathbb{Z}_q^n$$

  Then, to generate the public key - sample a noise vector $\mathbf{e} \xleftarrow{\$} \chi^m$ and compute:

  $$\mathbf{b} := \mathbf{sB} + \mathbf{e} \in \mathbb{Z}_q^m \quad \mathbf{A} := \binom{\mathbf{B}}{\mathbf{b}} \in \mathbb{Z}_q^{n\times m}$$

  Next, encrypt the secret key bit-by-bit according to the SHMK scheme. Let $\mathrm{Bit}_i(sk)$ denote the ith bit of sk. For every $i \in [n \cdot l_q]$ compute:

  $$\overrightarrow{\mathcal{S}}[i] \leftarrow \textbf{SHMK.Enc}(\mathbf{A}, \mathrm{Bit}_i(sk))$$

  Set the public key to:

  $$pk := (\mathbf{A}, \overrightarrow{\mathcal{S}}).$$

  We note that the $\overrightarrow{\mathcal{S}}$ part is only used for homomorphic evaluation and not for encryption.

- **FDMK.Enc**(pk, μ): Sample uniformly at random $\mathbf{r} \xleftarrow{\$} \{0,1\}^m$. Set and output the encryption:

  $$\mathbf{c} := \mathbf{Ar} + \mu 2^{l_q - 1}\mathbf{u}_n \in \mathbb{Z}_q^n$$

  where $\mathbf{u}_i$ is the ith standard basis vector.

- **FDMK.Dec**$((sk_i, \ldots, sk_N)), c)$: Parse each secret key as $\mathbf{t}_i := sk_i$ for every $i \in [N]$. Concatenate the secret keys and set $\hat{\mathbf{t}} = [\mathbf{t}_1, \ldots, \mathbf{t}_N] \in \mathbb{Z}_q^{nN}$. Compute and output

  $$\mu' := \mathsf{Treshold}(\hat{\mathbf{t}}, c),$$

  where $\mathsf{Treshold}(.,.)$ is defined as follows:

  $\mathsf{Treshold}(\mathbf{t}, \mathbf{c}) = \arg\min_{\mu\in\{0,1\}} \mathsf{noise}_{t,\mu}(\mathbf{c})$. *Then if* $\mathsf{noise}_{t,\mu}(\mathbf{c}) < q/8$ *for some* μ, *then* $\mathsf{Treshold}(\mathbf{t}, \mathbf{c}) = \mu$.

Note that FDMK is leveled by default. In order to remove the dependency on the multiplicative depth of the circuit the authors introduce bootstrapping and assume circular security which is the assumption that it is safe to encrypt the secret key under its own public key [2, 9, 14].

4 Methodology

The project consists mainly of three different phases: literature study, implementation phase and evaluation phase. In addition, these phases consists of several sub tasks that are described more in detail in this chapter.

4.1 Choosing the schemes to implement

The schemes to be implemented are chosen by reviewing the latest publications in the area with a special focus on already implemented schemes. Due to the fact that a lot of improvements have been done on the GSW scheme [3, 20, 21, 57] and there exist no previous implementation of multi-key FHE I decided to implement the SHMK scheme presented by Mukherjee et al. [7] and the FDMK scheme by Perlman et al. [2], described in section 3.2. Additionally, I decided to implement the single-key GSW scheme in order to be able to make the comparison that is needed to answer the research question:

- How does multi-key FHE compare to single-key FHE in terms of performance?

4.2 Implementation

Following the literature study comes the implementation phase that is divided into the following steps:

- System design: choosing tools, programming language and high performance number theory libraries as well as familiarising with the libraries.
- Implementation: generating the parameters for the LWE problem and implementing the algorithms of the schemes.
- Evaluation/Improvement: choosing the benchmarks and running experiments on them and investigate if any optimisation techniques can be used.

4.2.1 Programming Language and Libraries

Due to the complexity of the project it is advantageous to be able to rely on fast libraries. We decided to perform the implementation in C++ because there exist publicly available fast library that we could use avoiding implementing functions like matrix multiplication from scratch. Furthermore, most of the schemes [3, 20, 21] have been implemented in C++.

Regarding number theory libraries, there are mainly two options:

- **NTL**: Number Theory library that supports data structures and algorithms for vectors, matrices, and polynomials (`http://www.shoup.net/ntl/`) [58].
- **FLINT**: Fast Library for Number Theory that provides functionality for numbers, polynomials, power series and matrices over many base rings. (`http://flintlib.org/`) [59].

NTL is more well established and used in for example the HElib [22] implementation which is currently the most well developed implementation of FHE. HElib is based on the BGV [1] scheme along with many different optimisation techniques [60].

Some other useful libraries when implementing FHE are the following:

- **GMP**: GNUMultiple Precision Arithmetic Library used for arbitrary precision arithmetic on integers, rational numbers and floating-point numbers (`https://gmplib.org/`) [61].
- **FFTW**: C subroutine library for computing the discrete Fourier transform in one or more dimensions (`http://www.fftw.org/`) [62]. One reason for using FFT is that it increases the number of operations.

A disadvantage of choosing NTL is that it is not thread safe and not compatible with FFTW. On the other hand, FLINT has a lot of dependencies which makes it more complex to work with. The decision of which number theory library to use was mainly based on the outcomes of the compiled benchmarks that compare the relative performance of NTL and FLINT on some fundamental benchmarks where the decision was to go with NTL [63].

4.3 Choice of Parameters

In order to choose parameters correctly, in such a way to ensure a good security, we used the lwe-generator [1] and lwe-estimator [2] modules developed by Albrecht et al. [25]. The modules were designed with the intent of giving developers an easy way to choose parameters resisting known attacks attacks with respect to the current state of the art. Furthermore, we also compare the parameters with parameters chosen in other public implementations.

The parameters are:

- dimension n
- modulus q (e.g $q \approx n^2$)
- noise size α (e.g $\alpha q \approx \sqrt{n}$)

[1]`https://bitbucket.org/malb/lwe-generator`

[2]`https://bitbucket.org/malb/lwe-estimator`

- number of samples m
- Elements of $\mathbf{A}, \mathbf{s}, \mathbf{e}, \mathbf{c}$ are in $\mathbb{Z}_q$
- $\mathbf{e}$ is sampled from a discrete Gaussian with width $\sigma = \frac{\alpha q}{\sqrt{2\pi}}$

4.4 Evaluation

The objective of this section is to describe how the evaluation of performance and correctness of the implemented schemes is done. This phase aims to answer the following research questions:

- How does multi-key FHE compare to single-key FHE in terms of performance?

Optimisation techniques as those proposed in [20, 21] will be considered in the improvement phase. For example, Ducas [20] presents: a new method to homomorphically compute the NAND of two LWE encryptions, and a new technique to implement and speed up bootstrapping.

Chillotti et al. [21] take this optimisation even further to obtain a speed up from less than 1 second to less than 0.1 seconds.

The implementation will be evaluated in terms of performance, implementation complexity. More precisely, the runtime for the different algorithms will be measured with respect to different choices of parameters. Finally, the benchmarks of the different schemes will be compared.

4.4.1 Metrics

In order to evaluate FHE schemes, we used two measures of efficiency:

1. Space complexity: the size of the public key and the ciphertexts.
2. Time complexity: the running time of performing encryptions, decryptions, and homomorphic evaluations.

In theory, these metrics are asymptotic measures of efficiency in terms of the security parameter.

In implementation, these metrics are concrete statistics: space complexity is measured in bytes, and time complexity is measured in seconds per operation (relative to some specified computer).

In order to test the correctness of the implementations, the following is verified:

- The encryption of a message decrypts to the initial plaintext message (which corresponds to the correctness of the enc scheme).
- The decryption works after homomorphic operations are performed on the ciphertext (which corresponds to the correctness of the evaluation procedure).

5

Results and Comparasion

In this section we present the benchmarks of the different functions of the implemented schemes. We ran the experiments on an ordinary laptop with a 1.3 GHz Intel Core m7 processor, with 4MB L3-cache and 8GB memory.

In tables 5.1, 5.2 and 5.3 the average runtimes out of 100 runs are shown for the respective choices parameters for the different schemes. In tables 5.4, 5.5 and 5.6 we present the best runtimes followed by the worst ones in tables 5.7, 5.8 and 5.9.

Table 5.1: Average runtimes of the GSW scheme.

λ	q	$\log q$	n	m	Generate sk	Generate pk	Encrypt	Decrypt
10	1024	10	10	100	240ms	626ms	12078ms	473ms
16	65536	16	16	256	327ms	2824ms	98007ms	2711ms

Table 5.2: Average runtimes of the SHMK scheme.

λ	q	$\log q$	n	m	Setup	KeyGen	Encrypt	Expand
6	64	6	6	36	3ms	204ms	783285ms	14256ms
10	1024	10	10	100	4ms	677ms	58196616ms	976628ms

Table 5.3: Average runtimes of the FDMK scheme.

λ	q	$\log q$	n	m	Setup	KeyGen	Encrypt	Decrypt
10	1024	10	10	100	18ms	687ms	112ms	5ms
16	65536	16	16	256	14ms	2675ms	406ms	6ms

Table 5.4: Best runtimes of the GSW scheme.

λ	q	$\log q$	n	m	Generate sk	Generate pk	Encrypt	Decrypt
10	1024	10	10	100	145ms	522ms	10165ms	340ms
16	65536	16	16	256	178ms	1696ms	69462ms	1565ms

Table 5.5: Best runtimes of the SHMK scheme.

λ	q	$\log q$	n	m	Setup	KeyGen	Encrypt	Expand
6	64	6	6	36	1ms	100ms	585652ms	8218ms
10	1024	10	10	100	1ms	399ms	44459692ms	335155ms

Table 5.6: Best runtimes of the FDMK scheme.

λ	q	$\log q$	n	m	Setup	KeyGen	Encrypt	Decrypt
10	1024	10	10	100	8ms	367ms	68ms	2ms
16	65536	16	16	256	8ms	1587ms	246ms	4ms

Table 5.7: Worst runtimes of the GSW scheme.

λ	q	$\log q$	n	m	Generate sk	Generate pk	Encrypt	Decrypt
10	1024	10	10	100	2738ms	1573ms	20870ms	1636ms
16	65536	16	16	256	2219ms	13843ms	363969ms	9756ms

Table 5.8: Worst runtimes of the SHMK scheme.

λ	q	$\log q$	n	m	Setup	KeyGen	Encrypt	Expand
6	64	6	6	36	53ms	1962ms	2926817ms	141884ms
10	1024	10	10	100	16ms	1458ms	85805621ms	1803781ms

Table 5.9: Worst runtimes of the FDMK scheme.

λ	q	$\log q$	n	m	Setup	KeyGen	Encrypt	Decrypt
10	1024	10	10	100	1252ms	3216ms	712ms	29ms
16	65536	16	16	256	37ms	5256ms	884ms	30ms

6 Discussion

In this chapter the results from Chapter 5 and the research questions will be discussed and answered.

6.1 Literature Study

What implementation techniques can be used to optimise FHE schemes?

Most of the optimisation techniques that can be used for FHE schemes are theoretical improvements. This makes theoretical boundaries the bottleneck for FHE at the current state of the art.

Is it possible to make the schemes I implemented rely on more standard assumptions?

As long as we need bootstrapping to achieve FHE we need non-standard assumptions as circular security. Being able to get FHE without the circular security assumption is an open question. One way to achieve this would be if we could develop schemes that are not "noisy". This is still an open research question since all of the FHE schemes known so far are based on noise [64].

How does one choose parameters that guarantee a given level of security?

Choosing parameters for schemes based on the LWE problem is an intensively researched topic that is very hard [25]. Lattice algorithms and attacks on latties are not very well understood as runtimes are given asymptotically, algorithms are better in practice than the theoretical bounds and there are many heuristic assumptions [65]. Additionally, LWE is still a very new and complicated problem due to different parameters, instantiations and use cases. At the moment, there are no clear standards on how to choose secure parameters but tools as the lwe-generator and lwe-estimator [25] exist to make this process easier.

6.2 Implementation

What are the challenges of implementing FHE?

The main challenge when implementing FHE is that many papers use theoretical concepts that do not work in practice. For example, even though it is possible to represent any function as a circuit to perform it in practice is a rather non-trivial task.

Another challenge is that there exist very few examples on how to implement the bootstrapping procedure. The fact that IBM released their bootstrapping procedure 3 years after the rest of their implementation indicates it is not a trivial task [22]. The other implementations of bootstrapping are done by Ducas and Chillotti [20, 21] who implemented bootstrapping only for a very specific case considering only one gate.

Furthermore, the fact that it is very inefficient to perform the schemes functions with secure parameters makes testing and running benchmarks harder. Additionally, a magnitude of gigabytes is needed for the keys [20].

6.3 Evaluation

How does multi-key FHE compare to single-key FHE in terms of performance?

As can be seen in tables 5.1, 5.2 and 5.3 the performance does not differ much for the average runtimes of the Setup and KeyGen functions. This is due to the fact that the functions of the multi-key schemes are the same as in the single-key scheme. There is a difference in the performance because of the error sampling that varies in size as can be seen in the best and worst runtimes in tables 5.7, 5.8, 5.9, 5.4, 5.5 and 5.6. Furthermore, we see that the main bottleneck for the multi-key schemes is the expansion algorithm that allows the single-key scheme to be transformed into a multi-key one. The fact that the decryption function performs better in the multi-key setting is because the schemes were implemented exactly as they are described when they were first presented [3, 2, 7]. This means that the decryption function of the GSW implementation could be optimised as well.

7
Conclusion

In this thesis, we present a state of the art survey of FHE with a focus on multi-key schemes, and schemes that have public implementations. Such schemes enable a potentially untrusted party to run computation on data encrypted (encrypted by a single or by multiple users) in such a way that the outcome corresponds to the encrypted value of quested function evaluated on the corresponding ciphertexts. Moreover, we present three implementations of SWFHE: the single-key scheme GSW by Gentry et al. [3] and the multi-key schemes SHMK by Mukherjee et al. [7] and FDMK by Perlman et al. [2]. This was done in order to evaluate the difficulties of implementing FHE schemes. In addition, an evaluation of the implemented schemes is done with respect to efficiency and performance. As many of the functions of the multi-key schemes are the same as in the single-key scheme the performance does not differ much. The main problem for the multi-key schemes to become efficient is the expansion function that increases the size of the ciphertext by every party in the computation.

FHE is theoretically challenging and difficult. Therefore adding a multi key feature renders the design of such scheme even more complex. Additionally, the fact that FHE is heavily based on theoretical assumptions makes implementing it even more challenging.

In order to get FHE closer to the practical world new concepts need to be developed and tested. It is important to remember that the first secure FHE scheme is only 8 years old, and related concepts, assumptions and questions have been only after 2009 when Gentry first proposed his construction. There is still much work to be done before we will have a practical implementation of FHE and we can expect to see continuing focus on constructing new efficient schemes.

Bibliography

[1] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, “Fully homomorphic encryption without bootstrapping.” Cryptology ePrint Archive, Report 2011/277, 2011. `http://eprint.iacr.org/2011/277`.

[2] Z. Brakerski and R. Perlman, “Lattice-based fully dynamic multi-key fhe with short ciphertexts,” in *Annual Cryptology Conference*, pp. 190–213, Springer, 2016.

[3] C. Gentry, A. Sahai, and B. Waters, “Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based,” in *Advances in Cryptology–CRYPTO 2013*, pp. 75–92, Springer, 2013.

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

[5] J. Hoffstein, J. Pipher, and J. H. Silverman, “Ntru: A ring-based public key cryptosystem,” in *International Algorithmic Number Theory Symposium*, pp. 267–288, Springer, 1998.

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

[7] P. Mukherjee and D. Wichs, “Two round multiparty computation via multi-key fhe,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 735–763, Springer, 2016.

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

[9] C. Gentry, *A fully homomorphic encryption scheme.* PhD thesis, Stanford University, 2009.

[10] D. Boneh, E.-J. Goh, and K. Nissim, “Evaluating 2-dnf formulas on ciphertexts,” in *Theory of Cryptography Conference*, pp. 325–341, Springer, 2005.

[11] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 223–238, Springer, 1999.

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

[13] T. ElGamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” in *Workshop on the Theory and Application of Cryptographic Techniques*, pp. 10–18, Springer, 1984.

[14] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, “Fully homomorphic encryption without bootstrapping.” Cryptology ePrint Archive, Report 2011/277, 2011. `http://eprint.iacr.org/2011/277`.

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

[16] N. Smart and F. Vercauteren, “Fully homomorphic encryption with relatively small key and ciphertext sizes.” Cryptology ePrint Archive, Report 2009/571, 2009. `http://eprint.iacr.org/2009/571`.

[17] N. P. Smart and F. Vercauteren, “Fully homomorphic simd operations,” *Des. Codes Cryptography*, vol. 71, pp. 57–81, Apr. 2014.

[18] Z. Brakerski and V. Vaikuntanathan, “Efficient fully homomorphic encryption from (standard) lwe,” *SIAM Journal on Computing*, vol. 43, no. 2, pp. 831–871, 2014.

[19] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, “(leveled) fully homomorphic encryption without bootstrapping,” in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS ’12, (New York, NY, USA), pp. 309–325, ACM, 2012.

[20] L. Ducas and D. Micciancio, “Fhew: Bootstrapping homomorphic encryption in less than a second,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 617–640, Springer, 2015.

[21] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, “Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds.” Cryptology ePrint Archive, Report 2016/870, 2016. `http://eprint.iacr.org/2016/870`.

[22] S. Halevi and V. Shoup, “Helib.” `https://github.com/shaih/HElib`. Accessed: 2017-01-17.

[23] C. Peikert and S. Shiehian, “Multi-key fhe from lwe, revisited,” in *Theory of Cryptography Conference*, pp. 217–238, Springer, 2016.

[24] O. Regev, “On lattices, learning with errors, random linear codes, and cryptography,” *Journal of the ACM (JACM)*, vol. 56, no. 6, p. 34, 2009.

[25] M. R. Albrecht, R. Player, and S. Scott, “On the concrete hardness of learning with errors.” Cryptology ePrint Archive, Report 2015/046, 2015. `http://eprint.iacr.org/2015/046`.

[26] D. Micciancio, “The hardness of the closest vector problem with preprocessing,” *IEEE Transactions on Information Theory*, vol. 47, no. 3, pp. 1212–1215, 2001.

[27] Y. Lindell and J. Katz, *Introduction to modern cryptography.* Chapman and Hall/CRC, 2014.

[28] C. Fontaine and F. Galand, “A survey of homomorphic encryption for nonspecialists,” *EURASIP Journal on Information Security*, vol. 2007, no. 1, pp. 1–10, 2007.

[29] T. ElGamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

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

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

[32] D. Naccache and J. Stern, “A new public-key cryptosystem,” in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 27–36, Springer, 1997.

[33] I. Damgård and M. Jurik, “A generalisation, a simpli. cation and some applications of paillier’s probabilistic public-key system,” in *International Workshop on Public Key Cryptography*, pp. 119–136, Springer, 2001.

[34] Y. Ishai and A. Paskin, “Evaluating branching programs on encrypted data,” in *Theory of Cryptography Conference*, pp. 575–594, Springer, 2007.

[35] V. V. Kristin Lauter, Michael Naehrig, “Can homomorphic encryption be practical?,” tech. rep., May 2011.

[36] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, “Multiparty computation from somewhat homomorphic encryption,” in *Advances in Cryptology–CRYPTO 2012*, pp. 643–662, Springer, 2012.

[37] P. W. Shor, “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer,” *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[38] W. Diffie and M. Hellman, “New directions in cryptography,” *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[39] N. Koblitz, “Elliptic curve cryptosystems,” *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[40] V. S. Miller, “Use of elliptic curves in cryptography,” in *Conference on the Theory and Application of Cryptographic Techniques*, pp. 417–426, Springer, 1985.

[41] R. J. McEliece, “A public-key cryptosystem based on algebraic,” *Coding Thv*, vol. 4244, pp. 114–116, 1978.

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

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

[44] O. Regev, "Lattice-based cryptography," in *Annual International Cryptology Conference*, pp. 131–141, Springer, 2006.

[45] D. Micciancio, "Lattices in cryptography and cryptanalysis." Lecture notes of a course given in UC San Diego, 2002. `http://cseweb.ucsd.edu/~daniele/CSE207C/`.

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

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

[48] D. Micciancio, "Lattice-based cryptography," in *Encyclopedia of Cryptography and Security*, pp. 713–715, Springer, 2011.

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

[50] S. D. Galbraith, *Mathematics of public key cryptography*. Cambridge University Press, 2012.

[51] D. Micciancio, "Shortest vector problem," in *Encyclopedia of Cryptography and Security*, pp. 1196–1197, Springer, 2011.

[52] D. Micciancio, "Closest vector problem," in *Encyclopedia of Cryptography and Security*, pp. 212–214, Springer, 2011.

[53] D. Micciancio, "The shortest vector in a lattice is hard to approximate to within some constant," *SIAM journal on Computing*, vol. 30, no. 6, pp. 2008–2035, 2001.

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

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

[56] M. Clear and C. McGoldrick, "Multi-identity and multi-key leveled fhe from learning with errors." Cryptology ePrint Archive, Report 2014/798, 2014. `http://eprint.iacr.org/2014/798`.

[57] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Improving tfhe: faster packed homomorphic operations and efficient circuit bootstrapping." Cryptol-

ogy ePrint Archive, Report 2017/430, 2017. `http://eprint.iacr.org/2017/430`.

[58] V. SHOUP, “Ntl : A library for doing number theory,” *www.shoup.net/ntl/*.

[59] W. Hart, “Flint: Fast library for number theory.”

[60] S. Halevi and V. Shoup, “Algorithms in helib.” Cryptology ePrint Archive, Report 2014/106, 2014. `https://eprint.iacr.org/2014/106`.

[61] T. Granlund and the GMP development team, “The gnu multiple precision arithmetic library.”

[62] M. Frigo and S. G. Johnson, “The design and implementation of FFTW3,” *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005. Special issue on “Program Generation, Optimization, and Platform Adaptation”.

[63] V. Shoup, “Ntl vs flint,” 2016.

[64] K. Nuida, “Towards constructing fully homomorphic encryption without ciphertext noise from group theory.” Cryptology ePrint Archive, Report 2014/097, 2014. `https://eprint.iacr.org/2014/097`.

[65] N. Gama and P. Nguyen, “Predicting lattice reduction,” *Advances in Cryptology–EUROCRYPT 2008*, pp. 31–51, 2008.