NTNU – Trondheim
Norwegian University of
Science and Technology

The Smart-Vercauteren Fully Homomorphic Encryption Scheme

Vidar Klungre

Master of Science in Physics and Mathematics
Submission date: June 2012
Supervisor: Kristian Gjøsteen, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

We give a review of the Smart-Vercauteren fully homomorphic encryption scheme presented in 2010. The scheme follows Craig Gentry's blueprint of first defining a somewhat homomorphic encryption scheme, and prove that it is bootstrappable. This is then used to create the fully homomorphic scheme. Compared to the original paper by Smart and Vercauteren, we give a more comprehensive background, and explains the concepts of the scheme more in detail. This text is therefore well suited for readers who find Smart and Vercauteren's paper too brief.

Samandrag

Vi gir ein utvida presentasjon av Smart og Vercauteren sitt fullstendig homomorfe kryptosystem som vart utgitt i 2010. Kryptosystemet brukar samme mal som Craig Gentry brukte for sitt fullstendig homomorfe kryptosystem, ved å først definere eit kryptosystem som er delvis homomorft, for så å konstruere eit som er fullstendig homomorft. Denne rapporten vil vere meir omfattande enn den orginale artikkelen, derfor vil den vere nyttig for lesarar som synest at Smart og Vercauteren sin tekst er for lite detaljert.

Contents

Chapter 1

Introduction

To explain homomorphic encryption we consider unpadded RSA. Given the public key (m, e) and the encryption algorithm

$$c_i = \mathsf{Encrypt}(M_i) = M_i^e \bmod m,$$

then the following holds:

$$\begin{aligned}\mathsf{Encrypt}(M_1 \cdot M_2) &= M_1^e \cdot M_2^e \bmod m \\ &= (M_1 \cdot M_2)^e \bmod m \\ &= \mathsf{Encrypt}(M_1) \cdot \mathsf{Encrypt}(M_2).\end{aligned}$$

So if one decrypts $\mathsf{Encrypt}(M_1) \cdot \mathsf{Encrypt}(M_2)$, one obtains $M_1 \cdot M_2$. This shows that RSA is homomorphic with respect to multiplication.

In general, an encryption scheme is homomorphic with respect to the binary operator $*$ if there exists a corresponding binary operator $*'$ s.t.

$$\mathsf{Encrypt}(M_1 * M_2) = \mathsf{Encrypt}(M_1) *' \mathsf{Encrypt}(M_2)$$

holds for all messages M_1, M_2 in the plaintext space $\mathcal{P}$ of the scheme.

1.1 Fully Homomorphic Encryption

We will in this paper review an example of what we call a *fully homomorphic encryption scheme* (FHE scheme). This is an encryption scheme where we can do any operation homomorphically, not only one single multiplication, or another binary operator. Indeed we can do as many such operations as we like to with a fully homomorphic scheme

Assume we are working with bits, like we often do. Then addition and multiplication modulo 2 is functionally complete. Recall that an AND gate is the same operation as multiplication modulo 2, and an XOR gate equals addition modulo 2. Hence if we can run any boolean circuit consisting of only AND gates and XOR gates, then we have obtained fully homomorphic encryption. When we say any circuit, we here mean circuits of arbitrary depth, not only circuits up to a given depth. This is an important requirement, since practical circuits tend to be very deep.

Fully homomorphic encryption has many useful applications, e.g in cloud computing. Unfortunately there do not exist any practical implementation today. However, researchers are making progress, and the last few years the number of publications related to FHE has grown drastically.

Craig Gentry's work

In 2009 Craig Gentry made a breakthrough when he presented the first fully homomorphic encryption scheme [4]. He started by creating what he called a *somewhat homomorphic scheme* (SWHE scheme), which is a scheme that can evaluate boolean circuits homomorphically, but only up to a given depth.

Such SWHE schemes are easier to find than fully homomorphic schemes directly, and Gentry's method for constructing a fully homomorphic encryption scheme from a SWHE scheme is well described and easy to adapt to other SWHE schemes.

Gentry used a SWHE scheme based on ideal lattices, and he could therefore prove security by using well-studied lattice problems. Although his work was a huge theoretical discovery, it does not work well in practice when sufficient security is required.

Smart and Vercauteren's work

Gentry's work inspired others to make schemes based on the same idea, like [6] written by N. P. Smart and F. Vercauteren in 2010. They made a different fully homomorphic encryption scheme based on a different SWHE scheme than the one Gentry used.

The main purpose of this paper is to give an extended review of the Smart-Vercauteren-scheme. The scheme is of course explained by the authors themselves in [6], but the paper is short and some details are omitted. We will here give a full review which explains the scheme in detail, in addition to the background needed to understand it.

Like Gentry's scheme, the Smart-Vercauteren-scheme works well in theory, but not in practice. However, the theory is interesting, and the ideas are worth reviewing. For actual performance results see Section 10.

This review is naturally highly influenced by [6], but we will often also refer to Gentry's work [4].

1.2 Sections Overview

The actual SWHE scheme is presented in Section 3, but before that we will give a more complete summary of Gentry's idea in Section 2. This section also contains an overview of notation needed to understand the SWHE scheme.

In Section 4 we explain some of the algebraic number theory needed to understand the SWHE scheme. We choose to present this after the actual scheme, because then it is easier to relate the theory directly to the scheme we present. The actual analysis of the SWHE scheme is given in Section 5, while the security aspects of the SWHE scheme are discussed in Section 6.

In Section 7 we start the construction of the fully homomorphic scheme by redefining the key generation algorithm. Then we construct a circuit $C_{\mathcal{D}}$ which performs decryption in Section 8. In Section 9 this decryption circuit is used to create the algorithm $\mathsf{Recrypt}$ needed in the fully homomorphic scheme. Finally in Section 10 we summarize and give the final conclusion.

Chapter 2

Preliminaries

The goal of this section is to give the background needed before presenting the actual SWHE scheme. First we will review the work done by Gentry, which is needed to understand important concepts of SWHE schemes in general. Then comes a short part with important definitions which should clarify the notation we use when we present the SWHE scheme, but also in the remaining part of this paper.

In order to achieve full understanding of the SWHE scheme we present, we need a more complete review of the algebraic number theory used. This review is not given here, but follows in Section 4 after we have presented the SWHE scheme.

2.1 Public Key Encryption

A conventional public key encryption scheme consists of the three standard algorithms KeyGen, $\mathsf{Encrypt}$ and $\mathsf{Decrypt}$. KeyGen is used to set the plaintext space $\mathcal{P}$ and the ciphertext space $\mathcal{C}$. It also generates and returns the public key PK, and the secret key SK, which will be used for encryption and decryption. It is common to define $\mathcal{P} = \{0, 1\}$, and we will also do this for our scheme. In other words, we are encrypting bits.

The algorithm $\mathsf{Encrypt}$ takes as input a plaintext message $M \in \mathcal{P}$ and the public key PK. It returns a ciphertext $c \in \mathcal{C}$, a valid encryption of M. Since PK is public, anyone can encrypt messages.

$\mathsf{Decrypt}$ takes as input a ciphertext in $\mathcal{C}$, in addition to the secret key SK, and returns the plaintext message M corresponding to the ciphertext. This can only be done by users that possess the secret key SK, which usually is the one which runs KeyGen.

Homomorphic Encryption

A homomorphic encryption scheme has, in addition to the three standard algorithms, a fourth algorithm called $\mathsf{Evaluate}$. This algorithm takes as input the public key PK, a boolean t-input circuit C_t, and a vector containing t ciphertexts $c_1, \ldots, c_i$ where $c_i \xleftarrow{R} \mathsf{Encrypt}(M_i, \mathsf{PK})$. The $\mathsf{Evaluate}$ algorithm returns a ciphertext c such that

$$\mathsf{Decrypt}(c, \mathsf{SK}) = C_t(M_1, \ldots, M_t).$$

In other words, $\mathsf{Evaluate}$ evaluates circuits homomorphically. In the scheme we present here, we have replaced the $\mathsf{Evaluate}$ by two algorithms Add and Mult. They are used to perform respectively addition and multiplication homomorphically, which is sufficient since we are working with bits in $\mathbb{F}_2$. If we need to evaluate larger circuits, we simply use Add and Mult multiple times.

2.2 Gentry's Construction

Our goal is to create a scheme we can use to evaluate boolean circuits of arbitrary length. It has appeared to be very hard to find such schemes directly. Gentry solved this problem by first finding what he calls a somewhat homomorphic encryption scheme (SWHE scheme). A SWHE scheme is much easier to find than a fully homomorphic encryption scheme, and it does not differ too much. The SWHE scheme is then used as basis when the fully homomorphic scheme is constructed.

Noise of Ciphertexts

In the SWHE schemes we work with, each ciphertext has a small error. This error is often called *noise*, and is not critical if it is small. A ciphertext which is a direct result of $\mathsf{Encrypt}$, is what we call a *clean ciphertext*. Such ciphertexts have a very low amount of noise, and decryption of it will always be correct.

Now consider a ciphertext c which is the result of $\mathsf{Evaluate}$. This c typically has a larger noise value than the input ciphertexts of $\mathsf{Evaluate}$. In other words, homomorphic evaluation results in a ciphertext with larger error.

As indicated, the noise of a ciphertext is not a problem before it reaches a given value. But if the error exceeds this value, $\mathsf{Decrypt}$ will fail to return the decryption of the ciphertext.

Since the noise grows as we evaluate homomorphically, we get a problem if we try to evaluate deep circuits. The result will have a too large noise value, and decryption will fail. This is the difference between a SWHE scheme and a FHE scheme; we can only evaluate circuits up to a limited depth.

In the Smart-Vercauteren-scheme, each ciphertext c has a corresponding polynomial $C(x) = \sum_{i=0}^{t-1} c_i x^i$, and the noise equals the absolute value of the largest of the coefficients in $C(x)$. We often denote this value by $\|C(x)\|_\infty$, i.e.

$$\|C(x)\|_\infty = \max_{i=0,\dots,t} |c_i|.$$

Decryption will fail if $\|C(x)\|_\infty$ exceeds a limit, denoted by $\mathsf{r}_{\mathsf{Dec}}$. This can be though of as the largest "radius" $C(x)$ can have. We will later calculate $\mathsf{r}_{\mathsf{Dec}}$ for our scheme.

Recrypt

Since the noise grows as we evaluate circuits homomorphically, it would be useful to have an algorithm which reduces it. Gentry makes such an algorithm, and he calls it Recrypt. The Recrypt algorithm takes as input a ciphertext c with a large amount of noise, and the public key PK, and returns a clean ciphertext c_{new}. Notice that Recrypt does not remove all the error, it just sets the error to a relatively low level.

With this Recrypt algorithm we can evaluate circuits of arbitrarily length by doing one level at a time, and recrypt between each level to prevent the noise from growing above $\mathsf{r}_{\mathsf{Dec}}$.

Gentry proved that if the SWHE scheme is able to evaluate its own decryption algorithm homomorphically, then it is possible to obtain the Recrypt algorithm. He calls such SWHE schemes *bootstrappable*. In Section 5 we show how we should set our parameters to make our SWHE scheme bootstrappable. After than we will construct the Recrypt algorithm which is needed in the FHE scheme.

2.3 Notation

We here give a few important definitions which will be useful later.

Norms and Balls

Definition 2.3.1. *Given a polynomial $g(x) = \sum_{i=0}^{t} g_i x^i \in \mathbb{Z}[x]$ we define the 2-norm and the ∞-norm as*

$$\|g(x)\|_2 = \sqrt{\sum_{i=0}^{t} g_i^2} \text{ and } \|g(x)\|_\infty = \max_{i=0\dots t} |g_i|\,.$$

Definition 2.3.2. *For a positive value r, we define two corresponding types of "ball" centered at the origin:*

$$\mathcal{B}_{2,N}(r) = \left\{ \sum_{i=0}^{N-1} a_i x^i : \sum_{i=0}^{N-1} a_i^2 \leq r^2 \right\}$$

and

$$\mathcal{B}_{\infty,N}(r) = \left\{ \sum_{i=0}^{N-1} a_i x^i : -r \le a_i \le r \right\},$$

where all $a_i \in \mathbb{Z}$.

Resultant and Sylvester Matrix

Definition 2.3.3. *Given two polynomials* $F(x) = \sum_{i=0}^{M} f_i x^i, G(x) = \sum_{i=0}^{N} g_i x^i \in \mathbb{Z}[x]$, *we define the* resultant *of* $F(x)$ *and* $G(x)$ *as*

$$\mathsf{resultant}(F(x), G(x)) = \prod_{F(r)=0} G(r)$$

We denote the Sylvester matrix of $F(x)$ and $G(x)$ by $\mathsf{Syl}(F, G)$ and it is the $(M + N) \times (M + N)$ matrix, where all entries equal the coefficients in either $F(x)$ or $G(x)$, or zero. The first row is $(f_M, f_{M-1}, \dots, f_0, 0, \dots, 0)$, and the next $(N - 1)$ rows are equal to the previous row, just shifted one column to the right. The $(N + 1)$th row is $(g_N, g_{N-1}, \dots, g_0, 0, \dots, 0)$, and the next $(M - 1)$ rows are equal to the previous one, just shifted one column to the right. We then end up with the matrix

$$\mathsf{Syl}(F,G) = \begin{pmatrix} f_M & f_{M-1} & \cdots & f_0 & & & & \\ & f_M & f_{M-1} & \cdots & f_0 & & & \\ & & \ddots & \ddots & & \ddots & & \\ & & & f_M & f_{M-1} & \cdots & f_0 & \\ & & & & f_M & f_{M-1} & \cdots & f_0 \\ g_N & g_{N-1} & \cdots & g_0 & & & & \\ & g_N & g_{N-1} & \cdots & g_0 & & & \\ & & \ddots & \ddots & & \ddots & & \\ & & & g_N & g_{N-1} & \cdots & g_0 & \\ & & & & g_N & g_{N-1} & \cdots & g_0 \end{pmatrix}$$

where all empty cells are zero entries.

$\mathsf{Syl}(F, G)$ can be used to calculate useful properties of $F(x)$ and $G(x)$, like the resultant. In general we have that $\mathsf{det}(\mathsf{Syl}(F, G)) = \mathsf{resultant}(F, G)$.

Miscellaneous

All reductions modulo an odd integer n is defined to result in a value in the range

$$\left[-\frac{n-1}{2}, \frac{n-1}{2}\right]$$

unless otherwise stated.

For a real number a, we will use $\lfloor a \rceil$ to denote the integer closest to a, and for a polynomial $p(x) = p_0 + p_1 x + \cdots + p_k x^k \in \mathbb{R}[x]$ we will let $\lfloor p(x) \rceil$ denote the polynomial $\bar{p}(x)$ in $\mathbb{Z}[x]$ satisfying $\bar{p}(x) = \lfloor p_0 \rceil + \lfloor p_1 \rceil x + \cdots + \lfloor p_k \rceil x^k$. In other words, $\lfloor p(x) \rceil$ rounds the coefficients of $p(x)$.

Chapter 3

The Somewhat Homomorphic Scheme

We now present the Smart-Vercauteren scheme, and we denote it by Π, i.e.

$$\Pi = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Add}, \mathsf{Mult}).$$

Π is controlled by the triple of parameters (N,η, μ). A typically set of parameters would be $(N, 2^{\sqrt{N}}, \sqrt{N})$. Later we will see how these parameters should be chosen, and how they affect the performance of the scheme.

$\mathsf{KeyGen}()$:
- $\mathcal{P} = \{0, 1\}$.
- Pick a monic irreducible polynomial $F(x) \in \mathbb{Z}[x]$ of degree N.
- Do:
  - $S(x) \xleftarrow{R} \mathcal{B}_{\infty,N}(\eta/2)$.
  - $G(x) \leftarrow 1 + 2 \cdot S(x)$.
  - $p \leftarrow \mathsf{resultant}(G(x), F(x))$.
- Until p is prime.

- $D(x) \leftarrow \mathsf{gcd}(G(x), F(x))$ over $\mathbb{F}_p[x]$.
- Let $\alpha \in \mathbb{F}_p$ be the unique root of $D(x)$.
- Apply XGCD over $\mathbb{Q}[x]$ to obtain $Z(x) = \sum_{i=0}^{N-1} z_i x^i$ s.t.

$$Z(x) \cdot G(x) = p \mod F(x).$$

- $B \leftarrow z_0 \mod 2p$.
- Return $\mathsf{PK} = (p, \alpha)$ and $\mathsf{SK} = (p, B)$.

$\mathsf{Encrypt}(M, \mathsf{PK})$:
- If $M \notin \{0,1\}$ then abort.
- $R(x) \xleftarrow{R} \mathcal{B}_{\infty,N}(\mu/2)$.
- $C(x) \leftarrow M + 2 \cdot R(x)$.
- $c \leftarrow C(\alpha) \bmod p$.
- Return c.

$\mathsf{Decrypt}(c, \mathsf{SK})$:
- Return $M \leftarrow (c - \lfloor c \cdot B/p \rceil) \bmod 2$.

$\mathsf{Add}(c_1, c_2, \mathsf{PK})$:
- Return $(c_1 + c_2) \bmod p$.

$\mathsf{Mult}(c_1, c_2, \mathsf{PK})$:
- Return $(c_1 \cdot c_2) \bmod p$.

In KeyGen, $\mathcal{P}$ is set to be $\{0,1\}$. This means that we are encrypting bits only. After that $F(x)$ is defined, a monic polynomial irreducible over $\mathbb{Z}$ with degree N. $F(x)$ is not chosen randomly from the set of monic irreducible polynomials, so the one who runs KeyGen can choose a suitable $F(x)$. A typical choice for $F(x)$ is $x^N + 1$, where $N = 2^n$ for some n.

Next we pick $S(x) \xleftarrow{R} \mathcal{B}_{\infty,N}(\eta/2)$ and set $G(x) \leftarrow 1 + 2 \cdot S(x)$ and repeat this until $\mathsf{resultant}(G(x), F(x))$ is a prime number. At this point we have defined $F(x)$, $G(x)$ and a prime $p = \mathsf{resultant}(G(x), F(x))$. p will define the size of the ciphertext space $\mathcal{C}$, as we will see.

Further we set $D(x) \leftarrow \mathsf{gcd}(G(x), F(x))$ over $\mathbb{F}_p[x]$. $D(x)$ will have exactly one root in $\mathbb{F}_p$, and we denote this by α. This α will be used in the public key PK.

Now we find $Z(x) = \sum_{i=0}^{N-1} z_i x^i$ s.t. $Z(x) \cdot G(x) = p \mod F(x)$. From this we define $B \leftarrow z_0 \mod 2p$. Finally the KeyGen algorithm returns $\mathsf{PK} = (p, \alpha)$ and $\mathsf{SK} = (p, B)$.

The Encrypt algorithm only accepts plaintext messages $M \in \{0,1\}$. To encrypt the plaintext message we first add the message to two times a random polynomial $R(x) \in \mathbb{Z}[x]$ to obtain the polynomial $C(x)$. Then we evaluate $C(x)$ in α and reduce modulo p. This results in a ciphertext $c \in \mathbb{F}_p$.

The Decrypt algorithm is used to decrypt a ciphertext message $c \in \mathbb{F}_p$. We first multiply it by B/p, and then round the result to the nearest integer. Then the result is subtracted from c, before we reduce it modulo 2. This results in a message $M \in \{0,1\}$.

Add and Mult are the algorithms used to respectively add and multiply ciphertext messages homomorphically. They both take two ciphertexts $c_1 \xleftarrow{R} \mathsf{Encrypt}(M_1, \mathsf{PK})$ and $c_2 \xleftarrow{R} \mathsf{Encrypt}(M_2, \mathsf{PK})$ as input, together with the public key PK. They output encryptions of $M_1 + M_2 \bmod 2$ and $M_1 \cdot M_2 \bmod 2$ respectively.

Chapter 4

Algebraic Number Theory Background

Before we start analysing the somewhat homomorphic encryption scheme Π, we need some algebraic number theory background. We will focus on theory which is needed to do the analysis in later sections, however more distant subjects will also be reviewed for completeness or to achieve a better understanding.

Most of the theory stated here can be found in either [1], [2] or [5]. If statements are given here without proofs, these sources should give them.

4.1 The AKLB-setup

We start by defining the so-called *AKLB-setup*, named after the four rings it consists of; A, K, L and B (K and L are actually fields.). More precisely, we will discuss the special case where $A = \mathbb{Z}$, $K = \mathbb{Q}$, and L is the number field $\mathbb{Q}(\theta)$ where θ is a root in $F(x)$. The AKLB-setup is a common structure in the field of algebraic number theory, and described in e.g [1]. We will adapt this setup to make it suitable for our purposes, i.e. we will relate it directly to our SWHE scheme Π.

Consider $\mathbb{Z}$, the ring of integers, which is also an integral domain. This is the first of the four rings in the AKLB-setup. The second ring we consider is the field of fractions of $\mathbb{Z}$, which is $\mathbb{Q}$, the rational numbers. Both $\mathbb{Z}$ and $\mathbb{Q}$ should be well-known rings for to reader.

A polynomial $p(x) \in \mathbb{Z}$ is called primitive if the greatest common divisor of its coefficients is 1. Since $F(x)$ from Π is monic, it must also be primitive. A primitive polynomial is reducible over $\mathbb{Q}$ if and only if it is reducible over $\mathbb{Z}$. Since $F(x)$ is irreducible over $\mathbb{Z}$, this implies that $F(x)$ is irreducible over $\mathbb{Q}$ as well.

Now let θ be a root of $F(x)$. It is irrelevant which root of $F(x)$ we choose, since the arguments we present later will work for all of them. However, this θ must be fixed, since the structures we soon will present is based on θ. Notice that θ is not used in any of the algorithms in Π, so it is actually a hidden parameter for the users of Π. We will use θ to create a field extension of $\mathbb{Q}$, which will be the third of the rings in our AKLB-setup.

The Field Extension $\mathbb{Q}(\theta)$

Since $\mathbb{Q}$ is a subfield of $\mathbb{C}$, there exists a smallest intermediate field extension of $\mathbb{Q}$ which contains θ. We call this field $L = \mathbb{Q}(\theta)$, and it is the field generated by $\mathbb{Q}$ and θ. This is a simple field extension, since it is generated by the adjunction of only one element; θ.

According to [2] we have that $\mathbb{Q}(\theta) = \mathbb{Q}[\theta]$, where $\mathbb{Q}[\theta]$ is the ring of all polynomials in θ with rational coefficients, i.e.

$$\mathbb{Q}(\theta) = \mathbb{Q}[\theta] = \{q_0 + q_1\theta + \cdots + q_m\theta^m \mid q_0 + q_1x + \cdots + q_mx^m \in \mathbb{Q}[x]\}.$$

The set $\{1, \theta, \ldots, \theta^{N-1}\}$ forms a basis of $\mathbb{Q}(\theta)$ over $\mathbb{Q}$. This means that each element $\tau \in \mathbb{Q}(\theta)$ can be written uniquely as

$$\tau = q_0 + q_1\theta + \cdots + q_{N-1}\theta^{N-1},$$

where $q_i \in \mathbb{Q}$. $\mathbb{Q}(\theta)$ can therefore be considered a $\mathbb{Q}$-vector space of degree N. The dimension of this vector space is the degree of the field extension $\mathbb{Q}(\theta)/\mathbb{Q}$. This equals N, and since N is finite, $\mathbb{Q}(\theta)/\mathbb{Q}$ is a finite (algebraic) extension of $\mathbb{Q}$.

In the setup we have made so far, we know that $\mathbb{Q}(\theta) \cong \mathbb{Q}[x]/(F(x))$. This is a true statement also for all other roots of $F(x)$, θ_i for $i = 1, \ldots, N-1$. This implies that

$$\mathbb{Q}(\theta) \cong \mathbb{Q}(\theta_1) \cong \mathbb{Q}(\theta_2) \cong \ldots \cong \mathbb{Q}(\theta_{N-1}).$$

Since all of these field extensions are isomorphic, it may seem to be irrelevant which root we choose. That is true, but it is important to use the same root through the whole setup, since $\mathbb{Q}(\theta_i) \neq \mathbb{Q}(\theta)$ for $i = 1, \ldots, N-1$.

We have now introduced three of the four rings in our AKLB-setup; $A = \mathbb{Z}$, $K = \mathbb{Q}$ and $L = \mathbb{Q}(\theta) = \mathbb{Q}[\theta]$, for the chosen root θ of $F(x)$. This information is summarized in Figure 4.1.

The Ring of Integers $\mathcal{O}_L$

Since $L = \mathbb{Q}(\theta)$ is an algebraic extension of $\mathbb{Q}$, we know that each element $\tau \in L$ satisfies the equation

$$a_0 + a_1\tau + \cdots + a_m\tau^m = 0$$

for some coefficients $a_0, a_1, \ldots, a_m \in \mathbb{Q}$ and some m. If τ satisfies the same equation with coefficients in $\mathbb{Z}$, for some m, we call τ an *algebraic integer*. The collection of

$$\begin{tikzcd}
K = \mathbb{Q}[\theta] = \mathbb{Q}(\theta) \arrow[d, dash] & \\
\mathbb{Q} \arrow[r, dash] & \mathbb{Z}
\end{tikzcd}$$

Figure 4.1: Diagram showing three of the four rings in the AKLB-setup Π is based on.

all algebraic integers contained in L form a ring and we denote it by $\mathcal{O}_L$. We say that $\mathcal{O}_L$ is the integral closure of $\mathbb{Z}$ in L, and we will call $\mathcal{O}_L$ the ring of integers. Since L is algebraic over $\mathbb{Q}$ we also know that L is the field of fractions of $\mathcal{O}_L$.

Dedekind Domains

We will now show some additional properties of the ring $\mathcal{O}_L$, in particular, we will show that it is what we call a *Dedekind domain*.

Definition 4.1.1. *A Dedekind domain is an integral domain A satisfying the following three conditions:*

1. *A is a Noetherian ring.*
2. *A is integrally closed.*
3. *Every nonzero prime ideal of A is maximal.*

$\mathbb{Z}$ is a Dedekind domain, because it is a principal ideal domain. This implies that $\mathcal{O}_L$ also has to be a Dedekind domain according to [1]. The fact that $\mathcal{O}_L$ is a Dedekind domain will be useful later.

Now consider $\mathbb{Z}[\theta]$, the ring of all polynomials in θ with coefficients in $\mathbb{Z}$. Since $F(\theta) = 0$, we can replace all terms of degree higher than $N-1$ with lower degree terms, defined by the equation $F(\theta) = 0$. Therefore the following set will define $\mathbb{Z}[\theta]$ completely:

$$\mathbb{Z}[\theta] = \{z_0 + z_1\theta + \cdots + z_{N-1}\theta^{N-1} | z_i \in \mathbb{Z}\}.$$

Take an element $\tau = z_0 + z_1\theta + \cdots + z_{N-1}\theta^{N-1} \in \mathbb{Z}[\theta]$. Here z_i is a root in the polynomial $x - z_i \in \mathbb{Z}[x]$ and θ is a root in $F(x) \in \mathbb{Z}[x]$, hence $z_i \in \mathcal{O}_L$ and $\theta \in \mathcal{O}_L$. We earlier established that $\mathcal{O}_L$ is a ring, hence it must also include all sums of products of these elements, including τ. This proves that

$$\mathbb{Z}[\theta] \subseteq \mathcal{O}_L.$$

For the parameter choices we typically use, we often have the case where $\mathbb{Z}[\theta] = \mathcal{O}_L$. The rules set in KeyGen does not ensure this, so it is not true in general. However our scheme works with ideals in $\mathbb{Z}[\theta]$ that are assumed coprime with the index $[\mathcal{O}_L : \mathbb{Z}[\theta]]$, so we may as well assume that $\mathbb{Z}[\theta] = \mathcal{O}_L$. From this point we will denote this ring by $\mathbb{Z}[\theta]$, and just remind the reader occasionally that $\mathbb{Z}[\theta] = \mathcal{O}_L$.

We have now set up the complete AKLB-setup of four rings $A = \mathbb{Z}$, $K = \mathbb{Q}$, $L = \mathbb{Q}(\theta) = \mathbb{Q}[\theta]$ and $B = \mathbb{Z}[\theta] = \mathcal{O}_L$. $\mathbb{Z}$ and $\mathbb{Z}[\theta]$ are Dedekind domains, while $\mathbb{Q}$ and $\mathbb{Q}(\theta)$ are fields. The rings satisfy

$$\mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{Q}(\theta) \text{ and } \mathbb{Z} \subseteq \mathcal{O}_L \subseteq \mathbb{Q}(\theta).$$

In addition we know that $\mathbb{Q}$ and $\mathbb{Q}(\theta)$ are the fields of fractions of $\mathbb{Z}$ and $\mathbb{Z}[\theta]$ respectively.

We can now add $\mathcal{O}_L$ to the diagram in Figure 4.1, and we end up with the diagram in Figure 4.2 which summarizes the AKLB-setup.

$$\begin{tikzcd}
K = \mathbb{Q}[\theta] = \mathbb{Q}(\theta) \arrow[r, dash] \arrow[d, dash] & \mathcal{O}_L = \mathbb{Z}[\theta] \arrow[d, dash] \\
\mathbb{Q} \arrow[r, dash] & \mathbb{Z}
\end{tikzcd}$$

Figure 4.2: Diagram showing the complete AKLB-setup.

We will continue to work with the AKLB-setup, mostly with the Dedekind domains $\mathbb{Z}$ and $\mathcal{O}_L$, and eventually we will see how all this can be related to our SWHE scheme Π.

4.2 Ideals and Norms

Consider ideals in the Dedekind domain $\mathbb{Z}[\theta] = \mathcal{O}_L$. These ideals have the property that they can be factorized uniquely as a product of prime ideals. Given an ideal $\mathfrak{I} \subseteq \mathbb{Z}[\theta]$, we can express it as

$$\mathfrak{I} = \mathfrak{p}_1^{m_1} \cdot \mathfrak{p}_2^{m_1} \cdot \dots \cdot \mathfrak{p}_m^{m_1},$$

where the $\mathfrak{p}_i$ are prime ideals and $m_i \in \mathbb{Z}^+$. We will take a closer look at the ideals in $\mathbb{Z}[\theta]$, and especially the principal ideal generated by $\gamma = G(\theta)$, where θ is the root of $F(x)$ we chose earlier, and $G(x)$ is the polynomial defined in KeyGen. This ideal will be important later when we show correctness of Π. But before that we need to define the norm of elements in $\mathbb{Q}(\theta)$ and $\mathbb{Z}[\theta]$.

Norm of Elements in Fields and Rings

For any pair of rings (or fields) $A \subset B$, such that B is a free A-rank module of rank n, we have that $\beta \in B$ defines an A-linear transformation given by

$$x \mapsto \beta x : B \to B.$$

We define the determinant of this linear transformation to be the norm of the element β in the extension B/A. Thus if $\{e_1, e_2, \ldots, e_n\}$ is a basis for B over A, and $\beta \cdot e_i = \sum a_{ij} e_j$, then

$$\mathcal{N}_{B/A}(\beta) = \mathsf{det}(a_{ij}).$$

If it is obvious which extension we are working with, we may write $\mathcal{N}$ instead of $\mathcal{N}_{B/A}$.

Since all $a_{ij} \in A$, we know that $\mathcal{N}(\beta) = \mathsf{det}(a_{ij}) \in A$, because the determinant is calculated by only summing products of elements of A.

It can be proven that the norm map preserves multiplication, that is:

$$\mathcal{N}(a) \cdot \mathcal{N}(b) = \mathcal{N}(a \cdot b),$$

for all elements $a, b \in B$.

The definition above is valid for the field extension $\mathbb{Q}(\theta)/\mathbb{Q}$, so the norm of an element $\tau \in \mathbb{Q}(\theta)$ is a rational number. Since $\mathbb{Z}[\theta] \subseteq \mathbb{Q}(\theta)$, we can use the same definition also for elements in $\mathbb{Z}[\theta]$, just restricted to $\mathbb{Z}[\theta]$. Since $\mathbb{Z}[\theta] = \mathcal{O}_L$ is the ring of all integral elements of $\mathbb{Q}(\theta)$, we know that each element $\phi \in \mathbb{Z}[\theta]$, can be written as

$$\phi = z_0 + z_1\theta^1 + \cdots + z_{N-1}\theta^{N-1},$$

with $z_i \in \mathbb{Z}$. To find the norm of ϕ, we must first write $\phi \cdot \theta^i$ as a linear combination of the basis $\{1, \theta, \ldots, \theta^{N-1}\}$. When multiplying ϕ with θ^i, we get some terms of the form $z_k \cdot \theta^k$ where $z_k \in \mathbb{Z}$ and $k \geq N$. We can easily reduce these terms by using the equation $F(\theta) = 0$ ($F(x)$ is monic.), and what remains is just terms of the form $z_i \cdot \theta^i$ for $i < N$. We have now written $\phi \cdot \theta^i$ as a linear combination of the basis, with coefficients in $\mathbb{Z}$. The norm of ϕ is based only on these coefficients, hence $\mathcal{N}(\phi) \in \mathbb{Z}$.

Since $\mathbb{Q}$ has characteristic 0, the field extension $\mathbb{Q}(\theta)/\mathbb{Q}$ is a separable extension. From this fact and [1] we get a new simplified definition of the norm of an element in $\mathbb{Q}(\theta)$.

$$\begin{tikzcd} \phi \arrow[d, mapsto] & \in & \mathbb{Q}(\theta) \arrow[d] \\ \mathcal{N}(\phi) & \in & \mathbb{Q} \end{tikzcd} \qquad\qquad \begin{tikzcd} \phi \arrow[d, mapsto] & \in & \mathbb{Z}[\theta] \arrow[d] \\ \mathcal{N}(\phi) & \in & \mathbb{Z} \end{tikzcd}$$

Figure 4.3: Diagram explaining the norm map in the field extension $\mathbb{Q}(\theta)/\mathbb{Q}$ to the left and the ring extension $\mathbb{Z}[\theta]/\mathbb{Z}$ to the right.

Definition 4.2.1. (Norm of field elements) *In the field extension $\mathbb{Q}(\theta)/\mathbb{Q}$ we define the norm $\mathcal{N}$ of an element $\tau \in \mathbb{Q}(\theta)$ to be:*

$$\mathcal{N}(\tau) = \prod_{\sigma \in Gal} \sigma(\tau)$$

where Gal is the Galois group of $\mathbb{Q}(\theta)$ over $\mathbb{Q}$, consisting of all automorphisms of $\mathbb{Q}(\theta)$ which fix the base field $\mathbb{Q}$.

Again, this definition may be restricted to elements in $\mathbb{Z}[\theta]$.

Norm of Ideals

So far the norm map is defined only for elements, but we want to extend the norm map to ideals. In particular, we will define the norm of ideals in $\mathbb{Z}[\theta]$.

Definition 4.2.2. *In the ring $\mathbb{Z}[\theta]$ we define the norm $\mathcal{N}$ of an non-zero ideal $\mathfrak{I} \subseteq \mathbb{Z}[\theta]$ to be*

$$\mathcal{N}(\mathfrak{I}) = |\mathbb{Z}[\theta]/\mathfrak{I}|.$$

For principal ideals we get a more specific formula given by the following theorem.

Theorem 4.2.1. *For a principal ideal $\mathfrak{I} = a\mathbb{Z}[\theta] \subseteq \mathbb{Z}[\theta]$, the norm is given by*

$$\mathcal{N}(\mathfrak{I}) = |\mathcal{N}(a)|$$

Proof. Proof is given in Chapter 4.2 in [1]. □

The norm map for ideals follows many of the same properties as the norm of elements. The following properties hold for ideals $\mathfrak{a}, \mathfrak{b} \subseteq \mathbb{Z}[\theta]$:

$$\mathcal{N}(\mathfrak{a} \cdot \mathfrak{b}) = \mathcal{N}(\mathfrak{a}) \cdot \mathcal{N}(\mathfrak{b})$$

$$\mathcal{N}(\mathfrak{a}) = 1 \Leftrightarrow \mathfrak{a} = \mathbb{Z}[\theta].$$

Now that we have the AKLB-setup, and a good definition of the norm map, we are ready to look at some of the ideals that are created in KeyGen.

4.3 The Ideal $\mathfrak{p}$ created in KeyGen

Consider the ideal $\mathfrak{p}$ generated by $\gamma = G(\theta)$, i.e. $\mathfrak{p} = \gamma\mathbb{Z}[\theta]$. We will calculate the norm of $\mathfrak{p}$, and show that it is prime. After that we will show that $\mathfrak{p}$ equals the ideal $p\mathbb{Z}[\theta] + (\theta - \alpha)\mathbb{Z}[\theta]$.

Theorem 4.3.1. *Given $F(x), G(x)$ and p described in* KeyGen*, θ a root of $F(x)$ and $\gamma = G(\theta)$, then $\mathcal{N}(\gamma\mathbb{Z}[\theta]) = p$, and $\gamma\mathbb{Z}[\theta]$ is a prime ideal.*

Proof. By Definition 4.2.1, the norm of the principal ideal $\gamma\mathbb{Z}[\theta] \in \mathbb{Z}[\theta]$ is equal to the absolute value of the norm of γ, hence we must first calculate the norm of the element γ:

$$\begin{aligned}
\mathcal{N}(\gamma) &= \prod_{\sigma \in Gal} \sigma(\gamma) = \prod_{\sigma \in Gal} \sigma(G(\theta)) \\
&= \prod_{\sigma \in Gal} \sigma(g_0 + g_1\theta + \cdots + g_{N-1}\theta^{N-1}) \\
&= \prod_{\sigma \in Gal} \sigma(g_0) + \sigma(g_1)\sigma(\theta) + \cdots + \sigma(g_{N-1})\sigma(\theta^{N-1}) \\
&= \prod_{\sigma \in Gal} g_0 + g_1\sigma(\theta) + \cdots + g_{N-1}\sigma(\theta)^{N-1} \\
&= \prod_{\sigma \in Gal} G(\sigma(\theta)) = \prod_{F(\hat{\theta})=0} G(\hat{\theta}) \\
&= \mathsf{resultant}(G(x), F(x)) = p.
\end{aligned}$$

And by the definition we have of the norm of principal ideals we get:

$$\mathcal{N}(\gamma\mathbb{Z}[\theta]) = |\mathcal{N}(\gamma)| = p.$$

This is valid because of the fact that the automorphisms in Gal fix all elements in the base field, while the roots of $F(x)$ are permuted, and because $F(x)$ has no multiple roots.

To show that $\mathfrak{p} = \gamma\mathbb{Z}[\theta]$ is a prime ideal, suppose $\mathfrak{p}$ is a product of two ideals $\mathfrak{i}_1$ and $\mathfrak{i}_2$, i.e. $\mathfrak{p} = \mathfrak{i}_1 \cdot \mathfrak{i}_2$. Since the norm map is multiplicative, this implies that $p = \mathcal{N}(\mathfrak{p}) = \mathcal{N}(\mathfrak{i}_1) \cdot \mathcal{N}(\mathfrak{i}_2) \Rightarrow \mathcal{N}(\mathfrak{i}_1) = 1$ or $\mathcal{N}(\mathfrak{i}_2) = 1$. But this means that $\mathfrak{i}_1 = \mathbb{Z}[\theta]$ or $\mathfrak{i}_2 = \mathbb{Z}[\theta]$. Therefore, the prime factorization of $\mathfrak{p}$ is $\mathfrak{p}$ itself, in other words, $\mathfrak{p}$ is prime.

□

So we have now established that $\mathfrak{p} = \gamma\mathbb{Z}[\theta]$ is a prime ideal of norm p. We continue to prove that this ideal also equals $p\mathbb{Z}[\theta] + (\theta - \alpha)\mathbb{Z}[\theta]$.

Theorem 4.3.2. *If $F(x), G(x), \alpha$ and p are defined like described in* KeyGen, *θ is a root of $F(x)$ and $\gamma = G(\theta)$, then $\mathfrak{p} = \gamma\mathbb{Z}[\theta] = p\mathbb{Z}[\theta] + (\theta - \alpha)\mathbb{Z}[\theta]$.*

Proof. Since $D(x)|G(x) \mod p$, we also know that $D(\theta)|\gamma \mod p$. This means that there exist $f_1(\theta) \in \mathbb{Z}[\theta]$ s.t. $\gamma = D(\theta) \cdot f_1(\theta) \mod p$. This again implies that there exist an $f_2(\theta) \in \mathbb{Z}[\theta]$ s.t. $\gamma - D(\theta) \cdot f_1(\theta) = p \cdot f_2(\theta)$. Hence $\gamma = p \cdot f_2(\theta) + D(\theta) \cdot f_1(\theta) \in p\mathbb{Z}[\theta] + (\theta - \alpha)\mathbb{Z}[\theta]$. Therefore $\gamma\mathbb{Z}[\theta] \subseteq p\mathbb{Z}[\theta] + (\theta - \alpha)\mathbb{Z}[\theta]$.

By Theorem 4.3.1 we know that $\gamma\mathbb{Z}[\theta]$ is a prime ideal. Since $\mathbb{Z}[\theta] = \mathcal{O}_L$ is a Dedekind domain, we know that all nonzero prime ideals in $\mathbb{Z}[\theta]$ are maximal. Hence $\gamma\mathbb{Z}[\theta]$ has to be maximal. Now since $\gamma\mathbb{Z}[\theta] \subseteq p\mathbb{Z}[\theta] + (\theta - \alpha)\mathbb{Z}[\theta]$, and $p\mathbb{Z}[\theta] + (\theta - \alpha)\mathbb{Z}[\theta] \neq \mathbb{Z}[\theta]$, we know that $p\mathbb{Z}[\theta] + (\theta - \alpha)\mathbb{Z}[\theta] = \gamma\mathbb{Z}[\theta]$. □

The form $\mathfrak{p} = p\mathbb{Z}[\theta] + (\theta - \alpha)\mathbb{Z}[\theta]$ is called the two element representation of the ideal $\mathfrak{p}$. This is because $\mathfrak{p}$ is represented by two elements, namely p and $(\theta - \alpha)$. $\mathfrak{p}$ is indeed generated by p and $(\theta - \alpha)$ in the ring $\mathbb{Z}[\theta] = \mathcal{O}_L$.

In general each ideal in $\mathbb{Z}[\theta]$ can be represented in two different ways. The first way to represent an ideal is as a two element $\mathbb{Z}[\theta]$-basis like the one above, where we are given two elements $\delta_1, \delta_2 \in \mathbb{Z}[\theta]$, and the ideal equals

$$\delta_1 \cdot \mathbb{Z}[\theta] + \delta_2 \cdot \mathbb{Z}[\theta].$$

So $\mathfrak{p}$ is an ideal with $\delta_1 = p$ and $\delta_2 = (\theta - \alpha)$.

The other way of representing an ideal in $\mathbb{Z}[\theta]$ is as a N dimensional $\mathbb{Z}$-basis. Then we give N elements $\gamma_1, \ldots, \gamma_N \in \mathbb{Z}[\theta]$, and every element of the ideal is represented by the $\mathbb{Z}$-module generated by $\gamma_1, \ldots, \gamma_N$, i.e. each element τ can be written in the form

$$z_1 \cdot \gamma_1 + \cdots + z_N \cdot \gamma_N$$

where $z_i \in \mathbb{Z}$. It is common practice to represent this basis as an $N \times N$-matrix $(\gamma_{i,j})$ where we write γ_i in canonical form $\gamma_i = \sum_{j=1}^{N} \gamma_{i,j} \cdot \theta^{j-1}$. The rows of $(\gamma_{i,j})$ therefore represents the N elements in the basis.

If we now take the Hermite Normal Form (HNF), an analogue of reduced echelon form for matrices over the integers $\mathbb{Z}$, of $(\gamma_{i,j})$, we get a lower triangular matrix H. Given the ideal $\mathfrak{p}$, the corresponding HNF representation H is very simple to construct, and closely related to the two elements p and $(\theta - \alpha)$:

$$H = \begin{pmatrix} p & & \cdots & & 0 \\ -\alpha & 1 & & & \\ -\alpha^2 & & 1 & & \vdots \\ \vdots & & & \ddots & \\ -\alpha^{N-1} & & 0 & \cdots & 1 \end{pmatrix}.$$

We will use this matrix later in the security section.

Chapter 5

Correctness of the Scheme

In this section we will analyse the algorithms used in the somewhat homomorphic encryption scheme Π. We have already seen what is going on in the background when KeyGen is applied, so in this section we will focus more on $\mathsf{Encrypt}$ and $\mathsf{Decrypt}$. At the end we will also review the algorithms Add and Mult.

In particular we will find out how the parameters N, η and μ affect the behavior of the algorithms, and when they are correct or not. This will give us estimates of $\mathsf{r}_{\mathsf{Dec}}$ and d, the depth of the circuits we are able to evaluate homomorphically with Π.

5.1 Definitions

Given an encryption scheme, we call it *correct* if the $\mathsf{Decrypt}$ algorithm can be used to decrypt ciphertexts, that is for each message $M \in \mathcal{P}$ we need the following to be true.

$$\mathsf{Decrypt}(\mathsf{Encrypt}(M, \mathsf{PK}), \mathsf{SK}) = M$$

This is the most important property, because it ensures correct decryption of clean ciphertexts.

The second correctness property is related to homomorphic encryption. We must check that $\mathsf{Decrypt}$ correctly decrypts the ciphertexts we get after evaluating a circuit homomorphically. Given a plaintext message vector $\vec{M} = (M_1, M_2, \ldots, M_t)$ with $c_i \stackrel{R}{\leftarrow} \mathsf{Encrypt}(\mathsf{PK}, M_i)$, $\vec{c} = (c_1, c_2, \ldots, c_t)$ and a t-input boolean circuit C_t we must prove that

$$\mathsf{Decrypt}(\mathsf{Evaluate}(C_t, \vec{c})) = C_t(\vec{M}). \tag{5.1.1}$$

holds for circuits of depth lower than a boundary d. The requirement that the circuit must have depth at most d is only needed for SWHE schemes. For FHE schemes, Equation 5.1.1 should hold for circuits of any depth.

In Π, $\mathsf{Evaluate}$ is replaced with the two algorithms Add and Mult. So all we need to check is that these two algorithms satisfy the homomorphism property, i.e. for $M_1, M_2 \in \mathcal{P}$ and $c_i \overset{R}{\leftarrow} \mathsf{Encrypt}(M_i, \mathsf{PK})$, we need to show that

$$\mathsf{Decrypt}(\mathsf{Add}(c_1, c_2, \mathsf{PK})) = M_1 + M_2$$

and

$$\mathsf{Decrypt}(\mathsf{Mult}(c_1, c_2, \mathsf{PK})) = M_1 \cdot M_2.$$

5.2 Encrypt

We will now see what is going on when $\mathsf{Encrypt}(M, \mathsf{PK})$ is applied. First the polynomial $C(x) = M + 2 \cdot R(x)$ is created. This polynomial is then evaluated in α and reduced modulo 2 to obtain c, which is returned. Notice how the ciphertext c is closely related to the polynomial $C(x)$. We will later call this the ciphertext polynomial corresponding to c. Now we will show an important property of clean ciphertexts.

Theorem 5.2.1. *Given $F(x), p$ and α as described in* KeyGen*, θ a root in $F(x)$, $\mathfrak{p} = p\mathbb{Z}[\theta] + (\theta - \alpha)\mathbb{Z}[\theta]$ and $C(x), c$ as described in* $\mathsf{Encrypt}$*, then*

$$c = C(\alpha) \bmod p = C(\theta) \bmod \mathfrak{p}$$

Proof. Since $\theta - \alpha \in \mathfrak{p}$, we get that $\alpha = \theta \bmod \mathfrak{p}$, which again implies that

$$C(\theta) \bmod \mathfrak{p} = C(\alpha) \bmod \mathfrak{p}.$$

Both $C(\alpha) \bmod p$ and $C(\alpha) \bmod \mathfrak{p}$ returns a number in $\mathbb{F}_p$, and since p is included in $\mathfrak{p}$, it is clear that $C(\alpha) \bmod \mathfrak{p} = C(\alpha) \bmod p$.

□

An interesting consequence of Theorem 5.2.1 is that $C(\theta) - c \in \mathfrak{p}$. It appears to be the case that this holds also for all other ciphertexts.

Consider two ciphertexts c_1 and c_2 such that $c_i = C_i(\alpha) \bmod p$, and $C_i(\theta) - c_i \in \mathfrak{p}$. Now we define $c = \mathsf{Add}(c_1, c_2, \mathsf{PK}) = c_1 + c_2 \bmod p = C_1(\alpha) + C_2(\alpha) \bmod p$, which means that the polynomial corresponding to c is $C(x) = C_1(x) + C_2(x)$. Now it follows directly that $C(\theta) - c = C_1(\theta) + C_2(\theta) - c_1 - c_2 \in \mathfrak{p}$ since both $C_1(\theta) - c_1$ and $C_2(\theta) - c_2$ are elements of $\mathfrak{p}$.

A similar argument can be done for multiplication, where $C(x) = C_1(x) \cdot C_2(x)$. Since all ciphertexts we are working with are either clean ciphertexts, or a result of Add or Mult, we know that

$$C(\theta) - c \in \mathfrak{p}$$

holds for all ciphertexts $c = C(\alpha) \bmod p$.

5.3 Decrypt

Now consider Decrypt, the decryption algorithm. Ideally this algorithm takes any ciphertext and returns the decryption of it. However it will not be that easy because of the noise attached to the ciphertext. We will here calculate a bound $\mathsf{r}_{\mathsf{Dec}}$, such that we can ensure that $\mathsf{Decrypt}(c, \mathsf{SK})$ returns the correct decryption if $\|C(x)\|_\infty < \mathsf{r}_{\mathsf{Dec}}$, where $C(x)$ is the polynomial corresponding to c.

Earlier we established that

$$C(\theta) - c \in \mathfrak{p} = \gamma\mathbb{Z}[\theta] \tag{5.3.1}$$

holds for all ciphertexts c. In other words, we know there exist a $q(\theta) \in \mathbb{Z}[\theta]$ such that

$$C(\theta) - c = q(\theta) \cdot \gamma. \tag{5.3.2}$$

If we can obtain the element $C(\theta)$, we can easily retrieve M, because $M = C(\theta) \mod 2$. To find this $C(\theta)$ we need to use the polynomial $Z(x)$ from KeyGen satisfying

$$G(x) \cdot Z(x) = p \mod F(x). \tag{5.3.3}$$

We evaluate (5.3.3) in θ to get $\gamma^{-1} = Z(\theta)/p$. We multiply this with (5.3.2), and get

$$-c \cdot \frac{Z(\theta)}{p} = q(\theta) - \frac{C(\theta) \cdot Z(\theta)}{p}. \tag{5.3.4}$$

We will now round both sides of (5.3.4). For polynomials in $\mathbb{Q}[\theta]$ rounding is done by rounding each coefficient of the polynomial to its nearest integer. Notice that $q(\theta) \in \mathbb{Z}[\theta]$, so rounding this polynomial has no effect. The other term on the right hand side is $(C(\theta) \cdot Z(\theta))/p \in \mathbb{Q}[\theta]$, and rounding of this polynomial will change it.

Now assume that

$$\left\| \frac{C(\theta) \cdot Z(\theta))}{p} \right\|_\infty < \frac{1}{2} \tag{5.3.5}$$

holds. Then, rounding of $(C(\theta) \cdot Z(\theta))/p$ will result in the zero polynomial, and we simply get:

$$\left\lfloor \frac{-c \cdot Z(\theta)}{p} \right\rceil = \left\lfloor q(\theta) - \frac{C(\theta) \cdot Z(\theta)}{p} \right\rceil = q(\theta) - \left\lfloor \frac{C(\theta) \cdot Z(\theta))}{p} \right\rceil = q(\theta). \tag{5.3.6}$$

We insert this result into (5.3.2) and get:

$$C(\theta) = c + q(\theta) \cdot \gamma = c - \left\lfloor \frac{c \cdot Z(\theta)}{p} \right\rceil \cdot \gamma.$$

Now we can retrieve M by reducing $C(\theta)$ modulo 2. This gives us the following equation:

$$M = C(\theta) \bmod 2 = c - \left\lfloor \frac{c \cdot Z(\theta)}{p} \right\rceil \cdot \gamma \bmod 2. \tag{5.3.7}$$

Since $\gamma = G(\theta) = 1 + 2 \cdot S(\theta)$, we know that $\gamma \equiv 1 \bmod 2$. Hence we can remove γ from (5.3.7).

$$M = c - \left\lfloor \frac{c \cdot Z(\theta)}{p} \right\rceil \bmod 2. \tag{5.3.8}$$

The right hand side of (5.3.8) is a polynomial in θ, because $Z(\theta)$ is a polynomial in θ. All the other parts of the right hand side are independent of θ. But the left hand side is a binary number, since it is a plaintext. Hence we know that all terms of $Z(\theta)$ will disappear after rounding, and we can therefore ignore all these terms. We are then left with only the constant term of $Z(\theta)$, which is B. By replacing $Z(\theta)$ with B, we finally get the procedure we have in Decrypt:

$$M = c - \left\lfloor \frac{c \cdot B}{p} \right\rceil \bmod 2 \tag{5.3.9}$$

Notice that we do not claim that $Z(\theta) = B$, which this is not true in general. The only thing we can say for sure is that $\lfloor c \cdot B/p \rceil \equiv \lfloor c \cdot Z(\theta)/p \rceil \mod 2$.

In the argument above we used $B = z_0$, and not $B = z_0 \bmod 2p$ as it is defined in KeyGen. But this is indeed the same, since for any $k \in \mathbb{Z}$ we know that the following holds,

$$c - \left\lfloor \frac{c \cdot (B + 2pk)}{p} \right\rceil \equiv c - \left\lfloor \frac{c \cdot B}{p} + c \cdot 2k \right\rceil \equiv c - \left\lfloor \frac{c \cdot B}{p} \right\rceil + 2kc \equiv c - \left\lfloor \frac{c \cdot B}{p} \right\rceil \bmod 2.$$

In KeyGen, B is reduced modulo $2p$ since this gives a smaller integer to work with.

The correctness of the decryption algorithm relies on Equation 5.3.5. Since $Z(\theta)$ and p are fixed after KeyGen is applied, the only factor which can vary is $C(\theta)$, which is related to the encryption.

As indicated earlier the error of a ciphertext is related to the largest coefficient of $C(x)$. Now we see that this also agrees with what is needed for decryption. Decryption will only work if $\|C(x)\|_\infty$ is less than a given bound, $\mathsf{r}_{\mathsf{Dec}}$. We will continue to find this bound given what we have in Section 5.3.5. For this we need the following theorem.

Theorem 5.3.1. *Let $F(x), G(x) \in \mathbb{Z}[x]$ with $F(x)$ monic, $deg(F) = N$ and $deg(G) = M < N$ and $\mathsf{resultant}(F,G) = p$, then there exist a polynomial $Z(x) \in \mathbb{Z}[x]$ with $Z(x) \cdot G(x) = p \mod F(x)$ and*

$$\|Z(x)\|_\infty \leq \|G(x)\|_2^{N-1} \cdot \|F(x)\|_2^M$$

Proof. Since $\mathsf{resultant}(F,G) = p$, we know that $\gcd(F,G) = 1$, which means that there exist $Z(x) = \sum_{i=0}^{N-1} z_i x^i \in \mathbb{Q}[\theta]$ and $R(x) = \sum_{i=0}^{M-1} r_i x^i \in \mathbb{Q}[\theta]$ s.t.

$$Z(x) \cdot G(x) + R(x) \cdot F(x) = 1.$$

The z_i and r_i are then given by the following matrix equation:

$$\mathsf{Syl}(F,G)^T \cdot \begin{pmatrix} r_{M-1} \\ \vdots \\ r_0 \\ z_{N-1} \\ \vdots \\ z_0 \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ \\ \vdots \\ 0 \\ 1 \end{pmatrix}.$$

By Cramer's rule we find and explicit expression for the coefficients z_i, given by

$$z_i = \frac{\det(S_i)}{\det(\mathsf{Syl}(F,G))} = \frac{\det(S_i)}{p},$$

where S_i is the matrix we get by replacing the $(M+N-i)$th column by $(0,\ldots,0,1)^T$. So S_i will contain exactly M columns with coefficients from $F(x)$, $N-1$ columns with coefficients from $G(x)$, and one column equal to $(0,\ldots,0,1)^T$, when $i < N$. Now we use Hadamard's inequality to give a bound on z_i:

$$|z_i| = \left|\frac{\det(S_i)}{p}\right| \leq \|G(x)\|_2^{N-1} \cdot \|F(x)\|_2^M$$

Since this holds for all z_i, we finally get

$$\|Z(x)\|_\infty \leq \|G(x)\|_2^{N-1} \cdot \|F(x)\|_2^M$$

□

The leading coefficient in $G(x)$ is chosen randomly, so it is very unlikely that it equals zero. We will therefore use $M = N-1$, and we get

$$\|Z(x)\|_\infty \leq \|G(x)\|_2^{N-1} \cdot \|F(x)\|_2^{N-1}.$$

This relation will be used to find an expression for $\mathsf{r}_{\mathsf{Dec}}$ given Equation 5.3.5, but first we need another theorem.

Theorem 5.3.2. *Given $F(x)$, the polynomial from* KeyGen *and δ_∞ defined as*

$$\delta_\infty = sup \left\{ \frac{\|g(x) \cdot h(x) \bmod F(x)\|_\infty}{\|g(x)\|_\infty \cdot \|h(x)\|_\infty} | \, \mathsf{deg}(g), \mathsf{deg}(h) < N \right\},$$

then we have that

$$\|g(\theta) \cdot h(\theta)\|_\infty \leq \delta_\infty \cdot \|g(x)\|_\infty \cdot \|g(x)\|_\infty$$

for all polynomials $h(x)$ and $g(x)$ where $\mathsf{deg}(g), \mathsf{deg}(h) < N$.

Proof. Since δ_∞ is the greatest possible value of $\frac{\|g(x) \cdot h(x) \bmod F(x)\|_\infty}{\|g(x)\|_\infty \cdot \|h(x)\|_\infty}$, we get that

$$\|g(x) \cdot h(x) \bmod F(x)\|_\infty \leq \delta_\infty \cdot \|g(x)\|_\infty \cdot \|h(x)\|_\infty .$$

But $\|g(x) \cdot h(x) \bmod F(x)\|_\infty = \|g(\theta) \cdot h(\theta)\|_\infty$, since reduction modulo $F(x)$ is the same as evaluating in θ where all polynomials is reduced by the equation $F(\theta) = 0$. The result follows directly:

$$\|g(\theta) \cdot h(\theta)\|_\infty = \|g(x) \cdot h(x) \bmod F(x)\|_\infty \leq \delta_\infty \cdot \|g(x)\|_\infty \cdot \|h(x)\|_\infty .$$

□

With help of Theorem 5.3.1 and Theorem 5.3.2 we can now modify $\|C(\theta) \cdot Z(\theta)/p\|_\infty$ and find a bound on $C(x)$,

$$\begin{aligned} \left\| \frac{C(\theta) \cdot Z(\theta)}{p} \right\|_\infty &= \frac{\|C(\theta) \cdot Z(\theta)\|_\infty}{p} \\ &< \frac{\delta_\infty \cdot \|C(x)\|_\infty \cdot \|Z(x)\|_\infty}{p} \\ &= \frac{\delta_\infty \cdot \|C(x)\|_\infty \cdot \|G(x)\|_2^{N-1} \cdot \|F(x)\|_2^{N-1}}{p}. \end{aligned}$$

So decryption will work as long as this is less than $1/2$, i.e. when

$$\|C(x)\|_\infty < \frac{p}{2 \cdot \delta_\infty \cdot \|G(x)\|_2^{N-1} \cdot \|F(x)\|_2^{N-1}} := \mathsf{r}_{\mathsf{Dec}}.$$

This definition of $\mathsf{r}_{\mathsf{Dec}}$ is pretty complicated, so we do a simplification to make it more usable, in particular we will use the fact that $p \simeq \|G(x)\|_2^N \cdot \|F(x)\|_2^{N-1}$. We then get

$$\mathsf{r}_{\mathsf{Dec}} \simeq \frac{\|G(x)\|_2}{2 \cdot \delta_\infty} \simeq \frac{\sqrt{N} \cdot \eta}{2 \cdot \delta_\infty},$$

where $\|G(x)\|_2$ are estimated to be equal to $\sqrt{N} \cdot \eta$, because each coefficient of $G(x)$ has size approximately η.

5.4 Mult and Add

To evaluate larger boolean circuits than one simple addition or multiplication modulo 2, we will apply a large sequence of Add's and Mult's. As indicated earlier, this affects the error of the ciphertexts we are working with, and eventually it becomes so large that decryption fails. We will here do an analysis of this error propagation.

Consider two ciphertexts $c_1 = C_1(\alpha) \bmod p = M_1 + N_1(\alpha) \bmod p$ and $c_2 = C_2(\alpha) \bmod p = M_2 + N_2(\alpha) \bmod p$, where $N_i(x) = 2 \cdot R_i(x)$ is the error term. Notice that the messages M_i is indeed the right encryption of c_i, so all the error related to the ciphertext polynomial $C_i(x)$ comes from $N_i(x)$. N_i is therefore called the *error term*, or *noise term*. We will further assume that $N_i(x) \in \mathcal{B}_{\infty,N}(r_i - 1) \Rightarrow C_i(x) \in \mathcal{B}_{\infty,N}(r_i)$.

We first analyse addition of ciphertexts. We let

$$\begin{aligned} C_3(x) &= C_1(x) + C_2(x) \text{ mod } p \\ &= (M_1 + M_2) \text{ mod } 2 + 2 \cdot M_1 \cdot M_2 + N_1(x) + N_2(x) \text{ mod } p \\ &= M_3 + N_3(x) \text{ mod } p, \end{aligned}$$

where $M_3 = M_1 + M_2$ mod 2 is the desired message corresponding to $C_3(x)$, and $N_3(x) = 2 \cdot M_1 \cdot M_2 + N_1(x) + N_2(x)$ is the noise corresponding to $C_3(x)$. Ignoring all negligible terms, then

$$C_3(x) \in \mathcal{B}_{\infty,N}(r_1 + r_2).$$

In other words, if we add two ciphertexts homomorphically, the noise of the new ciphertext equals the sum of the noise of the two original ciphertexts.

Now we do the same for multiplication of ciphertexts. We get

$$\begin{aligned} C_4(x) &= C_1(x) \cdot C_2(x) \text{ mod } p \\ &= (M_1 + N_1(x))(M_2 + N_2(x)) \text{ mod } p \\ &= M_1 M_2 + M_1 N_2(x) + M_2 N_1(x) + N_1(x) N_2(x) \text{ mod } p \\ &= M_4 + N_4(x) \text{ mod } p. \end{aligned}$$

which shows that

$$C_4(x) \in \mathcal{B}_{\infty,N}(\delta_\infty \cdot r_1 \cdot r_2 + r_1 + r_2).$$

As expected, Mult generates much more noise than Add. We will now give an estimate of the depth d a circuit can have before decryption fails, i.e. before the noise becomes larger than $\mathsf{r}_{\mathsf{Dec}}$.

Assume we initially start with clean ciphertexts $C(x)$ lying in $\mathcal{B}_{\infty,N}(\mu+1)$. After executing a circuit with multiplicative depth d, we expect the resulting ciphertext c to have a polynomial $C(x)$ lying in a ball $\mathcal{B}_{\infty,N}(r)$ with radius

$$r \approx (\delta_\infty \cdot \mu)^{2^d}.$$

Decryption will only work when $r \leq \mathsf{r}_{\mathsf{Dec}}$, i.e. when

$$d \cdot \log 2 \leq \log\log \mathsf{r}_{\mathsf{Dec}} - \log(\delta_\infty \cdot \mu) \approx \log\log\left(\frac{\sqrt{N}\cdot\eta}{2\cdot\delta_\infty}\right) - \log\log(\delta_\infty\cdot\mu). \quad (5.4.1)$$

Notice that when $\mathsf{r}_{\mathsf{Dec}}$ is large, then d is large as well. This is naturally since a large $\mathsf{r}_{\mathsf{Dec}}$ should result in deeper circuits.

5.5 Choice of Parameters

The estimates for $\mathsf{r}_{\mathsf{Dec}}$ and d are of course not very accurate. However, the results should be reasonable, and overall they give us an idea of how the different parameters influence the performance of the SWHE scheme Π.

First, we see that our main parameter N appears in the numerator of $\mathsf{r}_{\mathsf{Dec}}$, which means that for large choices of N, Π perform better, i.e. both d and $\mathsf{r}_{\mathsf{Dec}}$ increase. But N also affects the parameter δ_∞, which appears in the denominator. Typically δ_∞ is proportional to N, which reduce $\mathsf{r}_{\mathsf{Dec}}$ for large N. The last parameter η is often expressed as a function of N, and typically η is growing exponentially as a function of N, which means that a large N makes $\mathsf{r}_{\mathsf{Dec}}$ larger.

To give an idea of how the δ_∞ behaves we compute it for different choices of $F(x)$.

Theorem 5.5.1. *Let $F_1(x) = x^N - a$ and $F_2(x) = x^N - ax^{N-1}$ then*

$$\delta_\infty(F_1(x)) \leq |a|N \text{ and } \delta_\infty(F_2(x)) \leq |a|^{N-1}N..$$

Proof. Let $g = \sum_{i=0}^{N-1} g_i x^i$ and $h = \sum_{i=0}^{N-1} h_i x^i$, then

$$g \cdot h \bmod F_1(x) = \sum_{i=1}^{N-1}\left(\sum_{0\leq k\leq k} g_i h_{k-i} + a\sum_{k<i<N} g_i h_{N+k-i}\right)x^k,$$

Where the first sum inside the parenthesis is the original product of terms of low degree, and the second sum is of additional terms of high degree that are reduced modulo $F_1(x)$.

Let $\tilde{g}$ and $\tilde{h}$ be the greatest coefficients of $g(x)$ and $h(x)$ respectively. Then $\|g(x)\|_\infty \cdot \|h(x)\|_\infty = \tilde{g}\cdot\tilde{h}$. To make the $\delta_\infty(F_1(x))$ as large as possible, we choose $g_i = \tilde{g}$ and $h_i = \tilde{h}$. Then the kth coefficient of $g\cdot h \bmod F_1(x)$ equals $(k+1)\tilde{g}\tilde{h}+a(N-k-1)\tilde{g}\tilde{h}$. This is largest when k is small, and by setting $k=0$ we get

$$\delta_\infty = \frac{\tilde{g} \cdot \tilde{h}(1 + (N-1)|a|)}{\tilde{g} \cdot \tilde{h}} \leq N|a|.$$

For $F_2(x)$ we write $g \cdot h = \sum_{k=0}^{2N-2} c_k x^k$, where each $c_k \leq N \cdot \|g\|_\infty \cdot \|h\|_\infty$. Then $g \cdot h \bmod F_2(x) = \sum_{k=0}^{N-1} d_k x^k$ with $d_k = c_k$ for $k = 0, \ldots, N-2$, and

$$d_{N-1} = \sum_{i=0}^{N-1} a^i \cdot c_{N-1+i}$$

d_{N-1} is the largest of the coefficients, so

$$\delta_\infty = \frac{\left|\sum_{i=0}^{N-1} a^i \cdot c_{N-1+i}\right|}{\|g\|_\infty \cdot \|h\|_\infty} \leq N \left|\sum_{i=0}^{N-1} a^i\right| \leq \left|a^{N-1}\right| \cdot N.$$

□

As Theorem 5.5.1 shows, δ_∞ takes different values for different $F(x)$. The intension of choosing $F_2(x) = x^N - ax^{N-1}$ was to show a polynomial which results in a fairly large δ_∞, while we choose $F_1(x) = x^N - a$ to show a polynomial with a low δ_∞.

To obtain large $\mathsf{r}_{\mathsf{Dec}}$ and a large multiplicative depth of the circuits we can evaluate with Π, we should choose $F(x)$ such that δ_∞ becomes as small as possible. $F(x) = x^N - 1$ is therefore a good choice. To make $F(x)$ irreducible, we choose $N = 2^n$. $F(x) = x^{2^n} - 1$ will therefore be a good choice, and this will be our canonical example in the remainder of the text.

Let us see how μ affects $\mathsf{r}_{\mathsf{Dec}}$. We see that a large μ gives a larger $\mathsf{r}_{\mathsf{Dec}}$. However it reduces the security, since μ is the parameter which bounds $C(x)$. With a small μ the possible choices of $C(x)$ corresponding to a ciphertext c is reduced, which then reduce the security. The lowest μ we can choose is $\mu = 2$, because $\mathsf{Encrypt}$ set $R(x) \in \mathcal{B}_{\infty,N}(\mu/2)$, which results in $R(x)$ being the zero polynomial each time if $\mu < 2$.

We see that a large η increase $\mathsf{r}_{\mathsf{Dec}}$ and enables us to evaluate deeper circuits homomorphically with Π, so a large η is desired. However if it becomes too large, we get problems with the security as we will see in Section 6.

To do a practical example we calculate d when $F(x) = x^N - 1$, $\mu = 2$, $\eta = 2^N$ with $N = 2^n$. We have chosen both $F(x)$ and μ to make d as large as possible. The results are not exactly promising; to get a positive value of d we must choose $n > 7$, and to get $d > 5$ we need to choose $n = 19 \Rightarrow N = 524288$. This results in $\log_2 p = 379625062$. A ciphertext is about the size of p, so each encrypted bit will need about 46MB of memory, which is very impractical.

Chapter 6

Security Analysis

We will now prove security of the SWHE scheme. In particular we will look at three security goals; onewayness, key recovery and semantic security. Semantic security alone is sufficient, but the other two are included as well.

Notice that we here only prove security for the SWHE scheme, and that we consider security of the FHE scheme later.

This section will be relatively brief, so for additional information about the security of our scheme we refer to the original work done by Smart and Vercauteren [6], and by Gu Chunsheng's cryptanalysis of our scheme [3].

6.1 Onewayness of Encryption

We now consider the problem of recovering a message M, given its ciphertext $c \xleftarrow{R} \mathsf{Encrypt}(M, \mathsf{PK})$ and the public key $\mathsf{PK} = (p, \alpha)$. We know that c is the result of $C(\alpha) \bmod p$ for some $C(x) = M + 2 \cdot R(x) \in \mathbb{Z}[x]$, and if we can find this $C(x)$, we can retrieve the message M as

$$M = C(x) \bmod 2.$$

Hence an adversary wins if he can find the integer coefficients of $C(x)$, i.e. if he can find $x_i \in \mathbb{Z}$ for $i = 0, \ldots, N-1$, such that

$$C(\alpha) \bmod p = c \Rightarrow \sum_{i=0}^{N-1} x_i \cdot \alpha^i = c - k \cdot p$$

where $|x_i| \leq \mu := \mathsf{r}_{\mathsf{Enc}}$, for some integer value of k.

Now consider the lattice generated by the columns in the matrix H presented in Section 4, that is all vectors in $\mathbb{R}^N$ of the form $\sum_{i=1}^{N} a_i \cdot c_i$ where c_i are the columns in H. Let us now take a look at the lattice vector

$$\vec{v} = (k, -x_1, \ldots, -x_n) \cdot H = (c - x_0, -x_1, \ldots, -x_n).$$

This is a lattice vector which is close in distance to the non-lattice vector $\vec{c} = (c, 0, \ldots, 0)$. By close we mean that the distance between the two vectors is less than $\mathsf{r_{Enc}}$ if we are using the ∞-norm or $\sqrt{N} \cdot \mathsf{r_{Enc}}$ if the 2-norm is used.

Now since c is public, we can find the vector $\vec{c}$, and search for lattice vectors near it to find $\vec{v}$. Hence determining the underlying plaintext given its encryption is an instance of the closest vector problem.

Definition 6.1.1. (The closest vector problem (CVP)) *Given a lattice $L \subseteq \mathbb{R}^n$, a norm N, and a vector $\vec{v} \in \mathbb{R}^n$, find the vector in L closest to $\vec{v}$ using the norm N.*

Notice that we do not reduce CVP to the problem of finding M, therefore we have not exactly proven that retrieving the message is at least as hard as solving CVP. However, if we can solve CVP, then can also find M given c and PK. The two problems are therefore similar, which indicates that finding M is likely to be as hard as CVP.

Gentry has also done analysis for this problem in [4]. Although one should bear in mind that Gentry's analysis is for a general lattice, and not for the specific one in our case. The best known attack on Gentry's scheme is one of lattice reduction, related to the bounded distance decoding problem (BDDP). In particular it is related to finding short/closest vectors within a multiplicative factor of $\mathsf{r_{Dec}}/\mathsf{r_{Enc}}$ in a lattice of dimension N. If we set

$$2^{\epsilon} = \frac{\mathsf{r_{Dec}}}{\mathsf{r_{Enc}}} = \frac{\sqrt{N} \cdot \eta}{2 \cdot \delta_{\infty} \cdot \mu}$$

then it is believed that solving BDDP has difficulty $2^{N/\epsilon}$. We shall refer to the value $2^{N/\epsilon}$ as the security level of our SHWE-scheme Π.

6.2 Key Recovery

Here we will see what an adversary must do to recover the secret key. Recall that we have $\mathsf{PK} = (p, \alpha)$ and $\mathsf{SK} = (p, B)$, hence, since p is public, the adversary wins if he can obtain B. This B can also be found if we have $Z(x)$, since $B = Z(0) \bmod 2p$.

This $Z(x)$ is the inverse of $G(x) \bmod F(x)$. Smart and Vercauteren [6] therefore claim that we can recover the key if we can find $\gamma = G(\theta)$. γ is what we call a small generator, since it is small compared to p, hence finding the secret key is an instance of the Small Principal Ideal Problem (SPIP).

Definition 6.2.1. (Small Principal Ideal Problem) *Given a principal ideal π in either two element or HNF representation, compute a small generator of the ideal.*

This problem is well-studied and considered hard. Smart and Vercauteren gives two approaches to solve this problem in [6].

6.3 Semantic Security

We will now prove that our SWHE scheme Π is semantically secure (SS). This can be done by showing that a hard problem can be reduced to the problem of breaking semantic security. In particular we will use the problem called the Polynomial Coset Problem (PCP). This problem will be presented after we have explained the standard game we use for proving SS.

Definition 6.3.1. *An encryption scheme is semantically secure if each adversary $\mathcal{A}$ only has negligible advantage against a challenger $\mathcal{C}$ in the following game.*

1. *$\mathcal{C}$ runs $\mathsf{PK}, \mathsf{SK} \xleftarrow{R} \mathsf{KeyGen}()$.*
2. *$\mathcal{C}$ sends PK to $\mathcal{A}$.*
3. *$\mathcal{A}$ choose two distinct messages M_0 and M_1 from $\mathcal{P}$.*
4. *$\mathcal{A}$ sends M_0 and M_1 to $\mathcal{C}$.*
5. *$\mathcal{C}$ pick $\beta \xleftarrow{R} \{0,1\}$.*
6. *$\mathcal{C}$ calculates $c \xleftarrow{R} \mathsf{Encrypt}(M_\beta, \mathsf{PK})$.*
7. *$\mathcal{C}$ sends c to $\mathcal{A}$.*
8. *$\mathcal{A}$ makes a guess β', and sends it back to $\mathcal{C}$.*

$\mathcal{A}$ wins the game if $\beta = \beta'$.

For our scheme where $\mathcal{P} = \{0,1\}$, $\mathcal{A}$ will always pick 0 and 1 as M_0 and M_2 in step 3. The problem for $\mathcal{A}$ in step 8 is therefore to find out if c is an encryption of 0 or 1.

Now we define the Polynomial Coset Problem.

Definition 6.3.2. *An adversary $\mathcal{A}$ solves the Polynomial Coset Problem if he has non-negligible advantage against a challenger $\mathcal{C}$ in the following game.*

1. *$\mathcal{C}$ runs $\mathsf{PK}, \mathsf{SK} \xleftarrow{R} \mathsf{KeyGen}()$.*
2. *$\mathcal{C}$ selects $b \xleftarrow{R} \{0,1\}$.*
3. *If $b = 0$, then $\mathcal{C}$ performs:*

   - $R(x) \xleftarrow{R} \mathcal{B}_{\infty,N}(\mathsf{r}_{\mathsf{Enc}}/2)$.
   - $r \leftarrow R(\alpha) \bmod p$.

   Whilst if $b = 1$, then $\mathcal{C}$ performs:

   - $r \xleftarrow{R} \mathbb{F}_p$.
4. *$\mathcal{C}$ sends (r, PK) to $\mathcal{A}$.*
5. *$\mathcal{A}$ makes a guess b', and sends it back to $\mathcal{C}$.*

$\mathcal{A}$ wins the game if $b = b'$.

We call this the Polynomial Coset Problem because of its similarities with Gentry's Ideal Coset Problem [4]. The problem is basically to determine if r is the evaluation of a small polynomial in α or a random positive integer less than p.

We will now prove that PCP can be reduced to SS. Hence breaking semantic security for Π is at least as hard as solving PCP.

Theorem 6.3.1. *Suppose there is an algorithm $\mathcal{A}$ which breaks semantic security of Π, with advantage ϵ. Then there is an algorithm $\mathcal{B}$, running in about the same time as $\mathcal{A}$, which solves the PCP with advantage $\epsilon/2$.*

Proof. The algorithm $\mathcal{B}$ receives the pair (r, PK) from the challenger $\mathcal{C}$, and sends PK directly to $\mathcal{A}$. $\mathcal{A}$ then returns M_0 and M_1. Now $\mathcal{B}$ picks a random bit $\beta \xleftarrow{R} \{0, 1\}$, and creates a challenge ciphertext for algorithm $\mathcal{A}$ from its own challenge (r, PK) by setting

$$c \leftarrow (M_\beta + 2 \cdot r) \bmod p,$$

$\mathcal{A}$ sends back a guess β' for β and $\mathcal{B}$ sends $b' = \beta \oplus \beta'$ to $\mathcal{C}$.

When $b = 0$ in the PCP, then c becomes a valid encryption of β, which means that $\mathcal{A}$ returns $\beta' = \beta$ with advantage ϵ. Now since $\beta' = \beta$, $\mathcal{B}$ will return the right answer, $b' = \beta' \oplus \beta = 0$ with advantage ϵ.

When $b = 1$, r is uniformly random modulo p and since p is odd, $2r$ is uniformly random modulo p and therefore so is c. Hence, the advantage of $\mathcal{A}$ is 0, which implies that $\mathcal{B}$'s overall advantage is $\epsilon/2$. $\square$

Since PCP is a reduction of SS, we now know that breaking semantic security for our scheme is at least as hard as solving PCP. PCP is considered to be hard, though it is not the most studied problem in cryptography.

Chapter 7

Fully Homomorphic Encryption

We have now proved that Π is a SWHE scheme; it consists of all the algorithms needed, and we have calculated $\mathsf{r_{Dec}}$, the greatest possible radius of ciphertexts we can decrypt. This $\mathsf{r_{Dec}}$ defines a set C_Π, all boolean circuits we are able to evaluate homomorphically with Π. This set is of course defined by our choice of parameters.

The next step to a fully homomorphic scheme is to show that our SWHE scheme Π is bootstrappable, i.e. show that Π can evaluate its own decryption algorithm homomorphically. Gentry proved that if a SWHE scheme is bootstrappable, then it can be extended to a fully homomorphic scheme. Since Decrypt is not a boolean circuit, we must create $C_\mathcal{D}$, a boolean circuit of finite depth which calculates

$$c - \left\lfloor \frac{c \cdot B}{p} \right\rceil \bmod 2$$

given c and SK. This $C_\mathcal{D}$ should be as shallow as possible, or at least shallow enough. By that we mean that $C_\mathcal{D} \in C_\Pi$. Eventually this $C_\mathcal{D}$ can be used to create the Recrypt algorithm needed in the FHE scheme.

7.1 Fully Homomorphic Key Generation

The decryption procedure

$$M = c - \lfloor c \cdot B/p \rceil$$

consists of a subtraction, a rounding, a multiplication and a division. While subtraction and rounding are relatively easy operations, multiplication and division are operations which require larger circuits. Therefore we want to avoid them in $C_\mathcal{D}$. The idea is to add a vector of integers to PK, and let the secret integer B equal the sum of a subset of these values.

To add more information to PK, we extend the KeyGen algorithm, parameterized by two integers s_1 and $s_2 \leq s_1$. This results in a new key generation algorithm called $\mathsf{KeyGen}_{\mathrm{FHE}}$. As the name indicates, this will also be the key generation algorithm for our fully homomorphic scheme.

$\mathsf{KeyGen}_{\mathrm{FHE}}()$:
- Run KeyGen to obtain p and α.
- Generate s_1 random integers $B_i \in [-p, p]$ such that there exists a subset S of s_2 elements satisfying

$$\textstyle\sum_{j \in S} B_j = B.$$

- Define $\mathsf{sk}_i = 1$ if $i \in S$ and 0 otherwise.
- Encrypt the bits sk_i under Π to obtain $\mathfrak{c}_i \xleftarrow{R} \mathsf{Encrypt}(\mathsf{sk}_i, \mathsf{PK})$.
- Finally return $\mathsf{SK} = (p, B)$ from KeyGen and:

$$\mathsf{PK} = (p, \alpha, s_1, s_2, \{\mathfrak{c}_i, B_i\}_{i=1}^{s_1}).$$

The integers s_1 and s_2 should be chosen such that that $s_1 \geq s_2$, and such that $\binom{s_1}{s_2}$ is large enough. Typically $s_1 \approx \log p$ and $s_2 \ll s_1$. In Section 7.2 we look at the security of the FHE scheme, and there we will discuss the choice of s_1 and s_2 in detail.

The integers B_i are chosen randomly from the interval $[-p, p]$, with only one requirement, namely that there exists a subset containing s_2 of the B_i, which sum up to B. The indices of these B_i are the elements of S, we therefore get $B = \sum_{j \in S} B_j$.

For each $i = 1, \ldots, s_1$ we set $\mathsf{sk}_i = 1$ if $i \in S$, and $\mathsf{sk}_i = 0$ if $i \notin S$. The vector (sk_i) will therefore contain information about which of the B_i we need to sum up to obtain B. In other words we know that

$$\sum_{i=1}^{s_1} \mathsf{sk}_i \cdot B_i = B.$$

Notice that exactly s_2 of the elements in this vector equals 1, while the $(s_1 - s_2)$ other elements equal 0. s_2 is small compared to s_1, so typically most of the entries in (sk_i) equals 0. In $\mathsf{KeyGen}_{\mathrm{FHE}}$, (sk_i) is encrypted under Π and we get the vector $\mathfrak{c}_i$.

In the last step we extend PK by adding s_1, s_2, and the list of pairs $\{\mathfrak{c}_i, B_i\}$ for $i = 1, \ldots s_1$. All the B_i are public, in addition to all the $\mathfrak{c}_i$. SK is also returned, but this is unchanged compared to the SK we had for Π.

Notice that the secret vector (sk_i) is not given anywhere, not even in the secret key. In $C_{\mathcal{D}}$ which we soon will present, the vector (sk_i) is required, which means that no one can use $C_{\mathcal{D}}$ directly to decrypt ciphertexts. However, since the encryption of (sk_i) is given as $(\mathfrak{c}_i)$ in PK, we are still able to evaluate $C_{\mathcal{D}}$ homomorphically, which leads us to the $\mathsf{Recrypt}$ algorithm.

7.2 Security of the Fully Homomorphic Scheme

In the extended key generation algorithm $\mathsf{KeyGen}_{\text{FHE}}$, we add some extra relevant information to the public key, which may make the scheme vulnerable. The two parameters s_1 and s_2 should therefore be chosen such that the secret integer B is hard to obtain. If s_1 is small, one could easily check all possible combinations of summing elements in (B_i), therefore s_1 must be large. We assume that we are not in a low density sum, i.e. $s_1 > \log p$. In this case solving SSSP takes at least $\sqrt{\binom{s_1}{s_2}}$ steps to solve. If we choose s_1 to be slightly greater than $\log p$, we need to select s_2 s.t.

$$\sqrt{\binom{s_1}{s_2}} > 2^{N/\epsilon},$$

where $2^{N/\epsilon}$ is the security we have for BDDP. We have to make sure that SSSP is at least as hard as BDDP.

7.3 Precision and Rounding

Assume we somehow got access to (sk_i). Then with the PK from $\mathsf{KeyGen}_{\text{FHE}}$ we can obtain the decryption of a ciphertext c by first calculating

$$B = \sum_{i=1}^{s_1} \mathsf{sk}_i \cdot B_i,$$

and then use B to calculate $M = c - \lfloor c \cdot B/p \rceil \bmod 2$. We combine these two steps and get

$$M = c - \left\lfloor \sum_{i=1}^{s_1} \mathsf{sk}_i \cdot \frac{c \cdot B_i}{p} \right\rceil \bmod 2 = c - \left\lfloor \sum_{i=1}^{s_1} \mathsf{sk}_i \cdot \frac{(c \cdot B_i \bmod 2p)}{p} \right\rceil \bmod 2$$

Notice that each of the elements in the sum are floating point numbers in the interval $[0, 2)$, which means that we can represent each $(c \cdot B_i \bmod 2p)/p$ uniquely as

$$e_0 \cdot 2^0 + e_1 \cdot 2^{-1} + \cdots + e_{t-1} \cdot 2^{-(t-1)},$$

where the e_i are bits and t is the chosen precision. This t should be as low as possible, but still large enough for the addition to give a reasonable answer without too much error. With t bits of precision, the maximal error of each term is $2^{-(t-1)}$, and because only s_2 of the s_1 terms are non-zero, the total error is at most $s_2 \cdot 2^{-(t-1)}$. Decryption will only work if this total error is less than $1/2$, which gives

$$s_2 \cdot 2^{-(t-1)} \leq 1/2 \Rightarrow t \geq \lfloor \log_2 s_2 \rfloor + 2$$

Let s denote the number of bits needed to represent all integers up to s_2, i.e. $s = \lfloor \log_2 s_2 \rfloor + 1$. Both t and s will be used when we construct $C_{\mathcal{D}}$.

To make the rounding in the decryption procedure easier, we can choose to divide $\mathsf{r}_{\mathsf{Dec}}$ by a factor of 2. This ensures that $c \cdot B/p$ is within the 1/4 of an integer, i.e.

$$c \cdot B/p \in (x - 1/4, x + 1/4)$$

for some integer x. Then there are only 4 different valid possibilities of e_0, e_1 and e_2, and all of them are shown in Table 7.1.

Now we can easily calculate the rounding of $c \cdot B/p = e_0 + e_1 \cdot 2^{-1} + e_2 \cdot 2^{-2} + \dots$ as $e_0 + e_1 \cdot e_2$, which leads to

$$\lfloor c \cdot B/p \rceil \bmod 2 = e_0 + e_1 \cdot e_2 \bmod 2.$$

e_0	e_1	e_2	$c \cdot B/p$	$\lfloor c \cdot B/p \rceil \bmod 2$
0	0	0	$[0, 0.25)$	0
0	1	1	$[0.75, 1)$	1
1	0	0	$[1, 1.25)$	1
1	1	1	$[1.75, 2)$	0

Table 7.1: Possible values of $c \cdot B/p = e_0 + e_1 \cdot 2^{-1} + e_2 \cdot 2^{-2} + \dots$, when we require that it should be within 1/4 of an integer.

Chapter 8

The Decryption Circuit

We now create $C_{\mathcal{D}}$, the circuit representing Decrypt. To be more precise, we create an algorithm that only uses bits and the two boolean gates XOR and AND to decrypt a ciphertext c. If $C_{\mathcal{D}}$ is shallow enough, then it can be evaluated homomorphically, which results in the Recrypt algorithm which we will show later.

$C_{\mathcal{D}}$ takes as input a ciphertext c and the vector (sk_i). It returns the decryption of c.

$C_{\mathcal{D}}(c, (\mathsf{sk}_i))$:

1. Write down the first t bits of the s_1 floating point numbers $(c \cdot B_i \bmod 2p)/p$ as an $s_1 \times t$ matrix T_1.
2. Multiply the ith row of T_1 by sk_i to obtain the $s_1 \times t$ matrix T_2, where $(T_2)_{i,j} = (T_1)_{i,j} \cdot \mathsf{sk}_i$
3. Compute the hamming weight (number of non-zero entries) of each column in T_2, and create the matrix T_3, where $(T_3)_{i,j}$ is the jth bit of the hamming weight of the ith column of T_2. The bits of T_3 can be found directly from T_2 by using symmetric polynomials.
4. Form the $t \times t$ matrix T_4 with $(T_4)_{i,j} = (T_3)_{i,j-i+s}$ whenever the right hand side is defined and zero otherwise.
5. Merge the rows of the matrix T_4, to obtain an $s \times t$ matrix T_5 such that the sum of the rows of T_5 equals the sum of the rows of T_4.
6. Apply carry-save-adders to reduce the 3 first rows to 2 rows. Repeat this procedure until we have a $2 \times t$ matrix T_6.
7. Perform the final addition of the two rows in T_6 to obtain the matrix T_7. Then compute the bit we get by rounding this result and reducing modulo 2. Finally subtract the result to $c \bmod 2$and output the resulting bit.

A toy example is shown in Figure 8.1 where $p = 17$, $B = 15$, $s_1 = 8$, $s_2 = 5$ and $c = 3$. In addition we have set

$$(B_i) = (-9, 13, 5, 10, 11, 3, 7, -6)$$

$$(\mathsf{sk}_i) = (1, 1, 0, 1, 0, 0, 1, 1).$$

This example is created to show how the decryption circuit works, not to give a practical scheme. Most of the parameters are too small to ensure sufficient security, but with e.g $s_1 > 8$ it would be hard to display the matrices in one page.
We will now go through each step of $C_{\mathcal{D}}$ carefully.

Step 1

In the first step we create the matrix T_1 where the i'th row is the bit representation of $(c \cdot B_i \bmod 2p)/p$. There are s_1 of the B_i in total, which each gives origin to a floating point number $c \cdot B_i \bmod 2p/p \in [0, 2)$. If these floating points are expressed with t bits of precision, i.e. each of them can be written on the form $\sum_{i=0}^{t-1} e_i \cdot 2^{-i}$, then T_1 will become a $s_1 \times t$ matrix containing bits which satisfies

$$\left\lfloor \frac{c \cdot B_i \bmod 2p}{p} \right\rceil = \left\lfloor \sum_{j=1}^{t} (T_1)_{i,j} \cdot 2^{-(j-1)} \bmod 2 \right\rceil .$$

The leftmost bit is the bit corresponding to the binary weight $2^0 = 1$, and as we go to the right in the matrix the corresponding binary weight is reduced by a factor of 2.

Notice that we can not write

$$\frac{c \cdot B_i \bmod 2p}{p} = \sum_{j=1}^{t} (T_1)_{i,j} \cdot 2^{-(j-1)},$$

because of the error we get by only using t bits of precision. Also observe that given a floating point number f, it is irrelevant which order we do rounding and reduction modulo 2, i.e.

$$\lfloor f \rceil \bmod 2 = \lfloor f \bmod 2 \rceil .$$

Both variants will be used in the remainder of the text.

Step 2

In step 2 we multiply the ith row with sk_i to obtain the matrix $(T_2)_{i,j} = ((T_1)_{i,j} \cdot \mathsf{sk}_i)$. Now the ith row contains the bit representation of $\mathsf{sk}_i \cdot (c \cdot B_i \bmod 2p)/p$ (given with t bit precision), hence if we sum the rows of T_2, we obtain the bit representation

Step 1

$$T_1 = \begin{array}{c} t \\ \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix} \end{array} s_1 \qquad (\mathsf{sk}_i) = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} s_1$$

Step 2

$$T_2 = \begin{array}{c} t \\ \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix} \end{array} s_1$$

Step 3

$$T_3 = \begin{array}{c} s \\ \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \end{array} t \quad \xrightarrow{\text{Step 4}} \quad T_4 = \begin{array}{c} t \\ \begin{pmatrix} 0 & & & & \\ 1 & 0 & & & \\ 1 & 0 & 0 & & \\ & 0 & 1 & 1 & \\ & & 0 & 1 & 0 \end{pmatrix} \end{array} t$$

Step 5

$$T_5 = \begin{array}{c} t \\ \begin{pmatrix} 0 & 0 & 0 & & \\ 1 & 0 & 1 & 1 & \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} s$$

Step 6

$$T_6 = \begin{array}{c} t \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & \end{pmatrix} \end{array} 2$$

Step 7

$$T_7 = \begin{array}{c} t \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Figure 8.1: A toy example showing how the decryption circuit works. Cells without any number in them are zero entries.

of $c \cdot B/p \bmod 2$ for the given precision. Hence if we round the sum, we obtain the same number as we would get by rounding $c \cdot B/p \bmod 2$, i.e.

$$\left\lfloor \frac{c \cdot B}{p} \bmod 2 \right\rceil = \left\lfloor \sum_{j=1}^{t} 2^{-(j-1)} \sum_{i=1}^{s_1} (T_2)_{i,j} \right\rceil \bmod 2. \tag{8.0.1}$$

But $\sum_{i=1}^{s_1} (T_2)_{i,j}$ is simply the hamming weight of row i, which we will use further in the next step.

Step 3

In step 3 we compute the hamming weight

$$\sum_{i=1}^{s_1} (T_2)_{i,j}$$

of each column of T_2, and represent the result with the matrix T_3. We are using bits, so T_3 is a $t \times s$ matrix where $(T_3)_{i,j}$ denotes the jth bit of the hamming weight of the ith column of T_2. The entries of T_3 can be calculated directly from the entries of T_2 by using symmetric polynomials, without calculating the actual hamming weight as an integer.

We use the term $\mathsf{SymPol}_i(b_1, b_2, \ldots, b_m)$ to denote the ith symmetric polynomial in the variables $b_1, b_2, \ldots, b_m$. This is the symmetric polynomial, consisting of $\binom{m}{i}$ terms, where all the terms are unique. Each of them is the product of i different bits, where each of these bits are chosen from the b_k, and none of them are equal. A few examples are shown below to make things more clear.

$$\begin{aligned}
\mathsf{SymPol}_1(b_1, b_2, \ldots, b_m) &= b_1 + b_2 + \cdots + b_m \\
\mathsf{SymPol}_2(b_1, b_2, \ldots, b_m) &= b_1 b_2 + b_1 b_3 + \cdots + b_i b_j + \cdots + b_{m-1} b_m \\
\mathsf{SymPol}_m(b_1, b_2, \ldots, b_m) &= b_1 b_2 \cdots b_m.
\end{aligned}$$

Now we can calculate the entries of T_3 as

$$(T_3)_{i,j} = \mathsf{SymPol}_{2^{s-j}}(b_1, b_2, \ldots, b_{s_1}) \bmod 2,$$

for $i = 1, \ldots, t$ and $j = 1, \ldots, s$, where $b_1, b_2, \ldots, b_{s_1}$ are the bits of column i in T_2, i.e. $b_k = (T_2)_{k,i}$.

Notice that since only s_2 of the rows in T_2 are non-zero, we can represent the hamming weight of each column in T_2 as a row in T_3 of length s.

$$
\begin{tikzcd}[column sep=large, row sep=large]
T_3 = \begin{array}{cc} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} & \begin{array}{c} 2^{0} \\ 2^{-1} \\ 2^{-2} \\ 2^{-3} \\ 2^{-4} \end{array} \\ \begin{array}{ccc} 2^{2} & 2^{1} & 2^{0} \end{array} & \end{array} \arrow[r, "\text{Shift}"] \arrow[dr, "\text{Step 4}"'] &
\begin{array}{c} \left(\begin{array}{cc|ccccc} 0 & 1 & 0 & & & & \\ & 0 & 1 & 0 & & & \\ & & 1 & 0 & 0 & & \\ & & & 0 & 1 & 1 & \\ & & & & 0 & 1 & 0 \end{array}\right) \\ \begin{array}{ccccccc} 2^{2} & 2^{1} & 2^{0} & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} \end{array} \end{array} \arrow[d, "\text{Resize}"] \\
& T_4 = \begin{array}{c} \begin{pmatrix} 0 & & & & \\ 1 & 0 & & & \\ 1 & 0 & 0 & & \\ & 0 & 1 & 1 & \\ & & 0 & 1 & 0 \end{pmatrix} \\ \begin{array}{ccccc} 2^{0} & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} \end{array} \end{array}
\end{tikzcd}
$$

Figure 8.2: Diagram explaining step 4 in the decryption procedure. Relevant binary weights are shown for each row and column in T_3. The second matrix is a shifting of each row of T_3, and the separation line shows which elements is removed due to overflow in T_4. All empty cells are zero-entries.

Now since the rows of T_3 is the bit representation of the hamming weight, we have that $\sum_{i=1}^{s_1}(T_2)_{i,j} = \sum_{k=1}^{s}(T_3)_{j,k}2^{s-k}$. Inserted into Equation 8.0.1, this gives the following equation:

$$
\begin{aligned}
\left\lfloor \frac{c \cdot B}{p} \right\rceil \bmod 2 &= \left\lfloor \frac{c \cdot B}{p} \bmod 2 \right\rceil \\
&= \left\lfloor \sum_{i=1}^{t} 2^{-(i-1)} \sum_{k=1}^{s} (T_3)_{i,k} \cdot 2^{s-k} \right\rceil \bmod 2 \\
&= \left\lfloor \sum_{i=1}^{t} \sum_{k=1}^{s} (T_3)_{i,k} \cdot 2^{s-k-i+1} \right\rceil \bmod 2.
\end{aligned}
\tag{8.0.2}
$$

We can then view T_3 as a matrix where each bit in row i corresponds to the binary weight $2^{-(i-1)}$, and column each bit in column k corresponds to the binary weight 2^{s-k}. The matrix T_3 is shown in Figure 8.2.

Step 4

To obtain $\lfloor c \cdot B/p \rceil \bmod 2$ from T_3 we had to sum over the entries. Each entry in row i had to be multiplied by $2^{-(i-1)}$, and similarly each entry in column k had to be multiplied by 2^{s-k}.

We want to collect these bits in a matrix T_4 such that we simply can sum over the rows, without the need of multiplying each row by a power of two. Notice that if we shift the entries of a row in T_3 one step to the right, it will have the same effect as multiplying by 2^{-1}. Hence we can shift the entries of row i in T_3 by $i-1$ steps, increase the width and insert 0s in all empty cells, and we get the desired matrix T_4.

Now ignore all columns which correspond to a binary weight greater or equal to 2, since we are interested in $\lfloor c \cdot B/p \rceil \bmod 2$ and not just $\lfloor c \cdot B/p \rceil$. We are then left with the $t \times t$ matrix T_4 given by $(T_4)_{i,j} = (T_3)_{i,j-i+s}$ when the right hand side is defined, and zero otherwise.

If we sum the entries of T_4 and multiply by the binary weight corresponding to each column we get:

$$\begin{aligned} \sum_{j=1}^{t} 2^{-(j-1)} \sum_{i=1}^{t} (T_4)_{i,j} &= \sum_{i=1}^{t} \sum_{j=1}^{t} (T_3)_{i,j-i+s} \cdot 2^{-(j-1)} && (8.0.3) \\ &= \sum_{i=1}^{t} \sum_{k=s+1-i}^{s+t-i} (T_3)_{i,k} \cdot 2^{s-k-i+1}. && (8.0.4) \end{aligned}$$

But $(T_3)_{i,k}$ is only defined if $k \le s$, and since $s+t-i$ is always greater than s, we can reduce the upper limit of the second sum to s. Then we get

$$\sum_{i=1}^{t} \sum_{k=s-i+1}^{s} (T_3)_{i,k} \cdot 2^{s-k-i+1}. \tag{8.0.5}$$

This is exactly the same as Equation 8.0.2 without all terms where $k < 1-i+s \Rightarrow 0 < s-k-i+1$, that is all terms where $2^{s-k-i+1} \ge 2$, which is also the terms that are congruent to 0 modulo 2. Hence we get

$$\left\lfloor \sum_{j=1}^{t} 2^{-(j-1)} \sum_{i=1}^{t} (T_4)_{i,j} \right\rceil \bmod 2 = \left\lfloor \frac{c \cdot B}{p} \right\rceil \bmod 2.$$

In the following steps the goal is to sum the rows of the matrix T_4, and it will be done in a way such that $C_{\mathcal{D}}$ becomes as shallow as possible.

Notice that the goal of this matrix T_4 is to sum the rows. This is exactly the same as for the matrix T_2 we created in step 2. The difference is that T_2 has s_1 rows, while T_4 has only t, so the number of rows has been reduced drastically. This method is used to reduce the complexity of the procedure, which we will see later when we do the detailed analysis.

Step 5

In step 4 we created the $t \times t$ matrix T_4 where each column contains at most s nonzero bits. Since we are only interested in the sum of the rows in T_4, we can permute the entries in each column without affecting this sum. We will therefore do a permutation of the entries in each column, such that the all the zeroes come in the first rows. After doing this, we remove all zero rows, and what remains is the $s \times t$ matrix T_5. The permutation of the other elements in each column is irrelevant for this decryption procedure, but it will be important when we work with the Recrypt algorithm later.

Now we have reduced the number of rows to a minimum, and we will start doing the actual addition of the rows.

Step 6

In step 6 we start doing the actual summing. We apply a sequence of carry-save-adders to do this. A carry-save-adder takes three rows containing bits, and replaces them by two other rows with the same sum as the three original rows. Since we start out with an $s \times t$ matrix, and we reduce the number of rows by one for each time we apply a carry-save-adder, we have to apply a sequence of $s-2$ carry-save-adders to end up with a $2 \times t$ matrix.

Let v_1, v_2 and v_3 be the three first rows of T_5, and let w_1 and w_2 be the two rows we will replace them with. Then the entries in w_1 are calculated in the following way:

$$w_{1,i} = v_{1,i} + v_{2,i} + v_{3,i} \bmod 2 \qquad \text{for } i = 1, \ldots, t$$

$$w_{2,t} = 0$$

$$w_{2,i} = v_{1,i+1}v_{2,i+1} + v_{1,i+1}v_{3,i+1} + v_{2,i+1}v_{3,i+1} \bmod 2 \qquad \text{for } i = (t-1), \ldots, 1$$

As before, if we get something larger than 2 it is automatically reduced modulo 2, since each row only can store numbers in $[0, 2)$. This is acceptable since we are only interested in the result modulo 2.

The reason for using a carry-save-adder instead of a normal adder which adds two rows of bits, is that the carry-save-adder reduce the total depth.

Step 7

In step 7 we calculate the sum of the two rows in T_6. This can not be done by a carry-save-adder, but we will use a normal adder where we start from the back and keep a carry. Let v_1 and v_2 be the two rows of T_6, and let w be the row vector storing the bits of $v_1 + v_2$. We will let c_i be the carry obtained after summing $v_{1,i}$ and $v_{2,i}$. We then use the following procedure to calculate c_i and w_i.

AddRows(v_1, v_2):
- Set $c_{t+1} = 0$.
- For $i = t, \ldots, 1$:
  - $w_i = v_{1,i} + v_{2,i} + c_{i+1} \bmod 2$.
  - $c_i = v_{1,i}c_{i+1} + v_{2,i}c_{i+1} + v_{1,i}v_{2,i} \bmod 2$.
- Return $w = (w_i)$.

Notice that as in step 6, we ignore all overflow into the bit position corresponding to the binary weight 2^1 and above, hence we are left with a value in the range $[0, 2)$ after adding.

To do the last rounding and reduction modulo 2, we only need look at the first 3 bits in w. All other bits of w corresponds to a binary weight of 2^{-3} or less, so they will not have any effect on the rounding. We get

$$\lfloor c \cdot B/p \rceil \bmod 2 = w_1 + w_2 w_3 \bmod 2.$$

Finally we subtract this result from $c \bmod 2$ to obtain the message M.

The algorithm $C_{\mathcal{D}}$ presented and explained here calculates exactly the same result as $\mathsf{Decrypt}$, while only using bits and simple binary operations like multiplication and addition modulo 2. In other words, we can express $C_{\mathcal{D}}$ as a boolean circuit, which also means that we can evaluate it homomorphically with our SWHE scheme Π (if it is shallow enough).

Chapter 9

The Recrypt Algorithm

Since $C_{\mathcal{D}}$ can be represented as a circuit, we can evaluate it homomorphically. This means that each bit is replaced with a ciphertext that encrypts it, while multiplication and addition modulo 2 are replaced with Add and Mult respectively.

9.1 Presenting the Recrypt Algorithm

We will now present the Recrypt algorithm, which is almost identical to $C_{\mathcal{D}}$ if one disregards that Recrypt works with numbers modulo in $\mathbb{F}_p$ while $C_{\mathcal{D}}$ works with bits in $\mathbb{F}_2$.

Recrypt takes as input a ciphertext c and $\mathsf{PK} = (p, \alpha, s_1, s_2, \{\mathfrak{c}_i, B_i\}_{i=1}^{s_1})$, and returns a ciphertext c_{new} with less error than the input c.

$\mathsf{Recrypt}(c, \mathsf{PK})$:

1. Write down the first t bits of the s_1 floating point numbers $(c \cdot B_i \bmod 2p)/p$ as an $s_1 \times t$ matrix T_1. Then encrypt each of the bits in T_1 under PK to obtain the matrix U_1, e.i $(U_1)_{i,j} \xleftarrow{R} \mathsf{Encrypt}((T_1)_{i,j}, \mathsf{PK})$.
2. Multiply the i-th row of U_1 by $\mathfrak{c}_i$ to obtain the $s_1 \times t$ matrix U_2 where $(U_2)_{i,j} = (U_1)_{i,j} \cdot \mathfrak{c}_i$.
3. Create the matrix U_3, where $(U_3)_{i,j} = \mathsf{SymPol}_{2^{s-j}}(c_1, c_2, \ldots, c_{s_1}) \bmod p$, where $c_1, c_2, \ldots, c_{s_1}$ are the ciphertexts of column i in U_2, i.e. $c_k = (U_2)_{k,i}$.
4. Form the $t \times t$ matrix U_4 with $(U_4)_{i,j} = (U_3)_{i,j-i+s}$ whenever the right hand side is defined and zero otherwise.
5. Merge the rows of the matrix U_4, so as to obtain an $s \times t$ matrix U_5 such that the sum of the rows of U_5 equals the sum of the rows of U_4.
6. Apply carry-save-adders to reduce the 3 first rows to 2 rows. Repeat this procedure until we have a $2 \times t$ matrix U_6.
7. Perform the final addition of the two rows in U_6 to obtain the vector U_7. Then compute $(U_7)_1 + (U_7)_1 \cdot (U_7)_2 \bmod p$. Finally subtract the result to $c \bmod p$ and output the result as c_{new}.

As we can see, this is exactly the same algorithm as $C_{\mathcal{D}}$, except for that we are working with encryptions of the bits instead of the bits themselves.

Step 1

In step 1 we create the matrix T_1 as we did in $C_{\mathcal{D}}$. After this we encrypt each of the entries under Π to obtain the matrix U_1. This matrix U_1 is the matrix corresponding to T_1 from $C_{\mathcal{D}}$.

Step 2

The vector $(\mathfrak{c}_i)$ is the encryption of the vector sk_i. In $C_{\mathcal{D}}$ we multiplied the entries of row i in the matrix T_1 with the sk_i modulo 2 to obtain T_2. Similarly we multiply each element of row i in U_1 by $\mathfrak{c}_i$ modulo p to obtain the matrix U_2. The multiplication in step 2 is the first homomorphic operation we do in Recrypt, and the result U_2 will then be the encryption of the matrix T_2 in $C_{\mathcal{D}}$. Now we clearly see a relation between $C_{\mathcal{D}}$ and Recrypt, namely that the encryption of T_i in $C_{\mathcal{D}}$ equals U_i in Recrypt. This will be true for all $i = 1, \ldots, 7$.

Step 3

To construct the matrix U_3, we use the symmetric polynomials we used in $C_{\mathcal{D}}$. These symmetric polynomials are also defined for inputs in $\mathbb{F}_p$, so we get

$$(U_3)_{i,j} = \mathsf{SymPol}_{2^{s-j}}(c_1, c_2, \ldots, c_{s_1}) \bmod p,$$

where $c_1, c_2, \ldots, c_{s_1}$ are the ciphertexts of column i in U_2, i.e. $c_k = (U_2)_{k,i}$. By correctness of homomorphic encryption we have that U_3 is an encryption of T_3.

Step 4

In step 4 we just move the elements of U_3 to the matrix U_4 like we did in when we constructed $C_{\mathcal{D}}$.

Step 5

In step 5 we permute the column entries of the matrix U_4. Like in the matrix T_4, at least $s-t$ of the entries in each column in U_4 equals zero. We put those elements in the first rows. It appears to be optimal to place the remaining ciphertexts such that the amount of noise increases as we descend the matrix. In the analysis we do later, we will see why this is optimal. The first $t-s$ rows will always be zero-rows, and we remove them to obtain the $s \times t$ matrix U_5.

Step 6

In step 6 we apply carry-save-adders to obtain the $2 \times t$ matrix U_6. This follows the same rules as for bits.

Step 7

In step 7 we add the last two rows of U_6 to obtain the vector U_7. U_7 is then an encryption of T_7. To obtain an encryption of $\lfloor c \cdot B/p \rceil$ we calculate

$$(U_7)_1 + (U_7)_1 \cdot (U_7)_2 \bmod p.$$

Finally we subtract this result from $\mathsf{Encrypt}(c, \mathsf{PK})$ to obtain c_{new} an encryption of $c - \lfloor c \cdot B/p \rceil = M$. This shows that the Recrypt algorithm does exactly what we want it to do, it takes as input a ciphertext c with high noise value (a *dirty* ciphertext) and returns a clean ciphertext c_{new} s.t.

$$\mathsf{Decrypt}(c_{new}, \mathsf{SK}) = \mathsf{Decrypt}(c, \mathsf{SK}).$$

9.2 Error Analysis of Recrypt

To find out how large our parameters should be to allow fully homomorphic encryption, we need to do a more detailed analysis of Recrypt. In particular we will see how much the error terms grow in each of the steps of Recrypt. To make things simple we will express all other relevant parameters as functions of N. Hence at the end we will find the required size of N needed to do fully homomorphic encryption.

In the calculations we will use $\mu = \sqrt{N}$, $\delta_\infty = N$ and $s_1 = N \cdot \sqrt{N}$. It will therefore be useful to define the growth parameter $\rho = \sqrt{N}$.

Assume we have two ciphertexts c_1 and c_2 corresponding to two randomizations $C_1(x) = M_1 + N_1(x)$ and $C_2(x) = M_2 + N_2(x)$; where $M_i \in \{0, 1\}$ are the messages and $N_i(x) \in \mathcal{B}_{\infty,N}(r_i - 1)$ are the randomnesses, i.e. $C_1(x) \in \mathcal{B}_{\infty,N}(r_1)$ and $C_2(x) \in \mathcal{B}_{\infty,N}(r_2)$. Let $\mathsf{rad}(c)$ denote the radius of the ball containing the corresponding polynomial $C(x)$, i.e. we have $C(x) \in \mathcal{B}_{\infty,N}(\mathsf{rad}(c))$. We recall from Section 5 that

$$\mathsf{rad}(c_1 \cdot c_2) = \delta_\infty \cdot \mathsf{rad}(c_1) \cdot \mathsf{rad}(c_2)$$

and

$$\mathsf{rad}(c_1 + c_2) = \mathsf{rad}(c_1) + \mathsf{rad}(c_2).$$

If A is a matrix of ciphertexts, we let $\mathsf{rad}(A)$ denote the matrix obtained by applying rad to each entry in A. We will also use the notation $g \sim f$, which means that $\lim_{\rho \to \infty} f/g = 1$.

Originally we start out with clean ciphertexts of radius $\mu + 1$, and as we add and multiply ciphertexts this radius increases.

Step 1

The result after step 1 is that we obtain an $s_1 \times t$ matrix U_1 containing clean ciphertexts with $\mathsf{rad}(U_1) = \mu + 1 \sim \rho$.

Step 2

In step 2 we multiply each entry of U_1 by another clean ciphertext from the matrix $(\mathfrak{c}_i)$. This results in the matrix U_2 with $(\mathsf{rad}(U_2))_{i,j} = \delta_\infty \cdot r_2^2 \sim \rho^4$ for all i, j.

Step 3

In U_2 each of the entries have the same noise value, but after step 3, we obtain U_3, a matrix where the noise value is different for each column. Recall that $(U_3)_{i,j} = \mathsf{SymPol}_{2^{s-j}}(c_1, c_2, \ldots, c_{s_1})$, where $c_1, c_2, \ldots, c_{s_1}$ are the ciphertexts of column i in U_2, i.e. $c_k = (U_2)_{k,i}$. So we obtain a ciphertext in column j, by summing $\binom{s_1}{2^{s-j}}$ terms, where each term is the product of 2^{s-j} ciphertexts from U_2. So each term is of order

$$(\rho^4)^{2^{s-j}} \cdot \delta_\infty^{2^{s-j}-1} = \rho^{6 \cdot 2^{s-j}-2}.$$

There are in total $\binom{s_1}{2^{s-j}}$ terms, where

$$\binom{s_1}{2^{s-j}} \sim \frac{s_1^{2^{s-j}}}{2^{s-j}!} \sim \frac{\rho^{3 \cdot 2^{s-j}}}{2^{s-j}!}.$$

We then get

$$\mathsf{rad}((U_3)_{i,j}) \sim \binom{s_1}{2^{j-1}} \rho^{6\cdot 2^{s-j}-2} \sim \frac{\rho^{3\cdot 2^{s-j}-2}}{2^{j-1}!}.$$

This value only depends on j, the column number, and not i.

Step 4

In step 4 we do no multiplication nor addition to the existing ciphertexts. The only thing we do is to move some of the entries from U_3 over to U_4. From this point the result depends on the choice of s and t, we will therefore just show $\mathsf{rad}(U_4)$ for common choices of (s,t).

Case $(s,t) = (3,5)$:

$$\mathsf{rad}(U_4) \sim \begin{pmatrix} \rho^7 & 0 & 0 & 0 & 0 \\ \rho^{16}/2 & \rho^7 & 0 & 0 & 0 \\ \rho^{34}/4! & \rho^{16}/2 & \rho^7 & 0 & 0 \\ 0 & \rho^{34}/4! & \rho^{16}/2 & \rho^7 & 0 \\ 0 & 0 & \rho^{34}/4! & \rho^{16}/2 & \rho^7 \end{pmatrix}$$

Case $(s,t) = (4,5)$:

$$\mathsf{rad}(U_4) \sim \begin{pmatrix} \rho^7 & 0 & 0 & 0 & 0 \\ \rho^{16}/2 & \rho^7 & 0 & 0 & 0 \\ \rho^{34}/4! & \rho^{16}/2 & \rho^7 & 0 & 0 \\ \rho^{70}/8! & \rho^{34}/4! & \rho^{16}/2 & \rho^7 & 0 \\ 0 & \rho^{70}/8! & \rho^{34}/4! & \rho^{16}/2 & \rho^7 \end{pmatrix}$$

Case $(s,t) = (4,6)$:

$$\mathsf{rad}(U_4) \sim \begin{pmatrix} \rho^7 & 0 & 0 & 0 & 0 & 0 \\ \rho^{16}/2 & \rho^7 & 0 & 0 & 0 & 0 \\ \rho^{34}/4! & \rho^{16}/2 & \rho^7 & 0 & 0 & 0 \\ \rho^{70}/8! & \rho^{34}/4! & \rho^{16}/2 & \rho^7 & 0 & 0 \\ 0 & \rho^{70}/8! & \rho^{34}/4! & \rho^{16}/2 & \rho^7 & 0 \\ 0 & 0 & \rho^{70}/8! & \rho^{34}/4! & \rho^{16}/2 & \rho^7 \end{pmatrix}$$

As we can see get the same triangular-like pattern which we got in $C_{\mathcal{D}}$. Notice that the noise value is smallest on the diagonal, and that it increases as we go further away from the diagonal line. Also notice how none of the columns have more than s non-zero entries.

Step 5

Now we will permute the entries in each column, and after that we will resize U_4 to end up with the $s \times t$ matrix U_5. It turns out that the best way of doing this is to order the column entries such that the noise increases as you descend a column. This will also put all the zeroes at the top, and we can easily delete these rows. There will be at least $t-s$ zero rows after permuting, and by removing these rows we end up with the $s \times t$ matrix U_5. For our three different cases we get the following matrices.

Case $(s,t) = (3,5)$:

$$\mathsf{rad}(U_5) = \begin{pmatrix} \rho^7 & \rho^7 & \rho^7 & 0 & 0 \\ \rho^{16}/2 & \rho^{16}/2 & \rho^{16}/2 & \rho^7 & 0 \\ \rho^{34}/4! & \rho^{34}/4! & \rho^{34}/4! & \rho^{16}/2 & \rho^7 \end{pmatrix}$$

Case $(s,t) = (4,5)$:

$$\mathsf{rad}(U_5) = \begin{pmatrix} \rho^7 & \rho^7 & 0 & 0 & 0 \\ \rho^{16}/2 & \rho^{16}/2 & \rho^7 & 0 & 0 \\ \rho^{34}/4! & \rho^{34}/4! & \rho^{16}/2 & \rho^7 & 0 \\ \rho^{70}/8! & \rho^{70}/8! & \rho^{34}/4! & \rho^{16}/2 & \rho^7 \end{pmatrix}$$

Case $(s,t) = (4,6)$:

$$\mathsf{rad}(U_5) = \begin{pmatrix} \rho^7 & \rho^7 & \rho^7 & 0 & 0 & 0 \\ \rho^{16}/2 & \rho^{16}/2 & \rho^{16}/2 & \rho^7 & 0 & 0 \\ \rho^{34}/4! & \rho^{34}/4! & \rho^{34}/4! & \rho^{16}/2 & \rho^7 & 0 \\ \rho^{70}/8! & \rho^{70}/8! & \rho^{70}/8! & \rho^{34}/4! & \rho^{16}/2 & \rho^7 \end{pmatrix}$$

Step 6

In step 6 we perform a sequence of carry-save-adders to reduce the number of rows to 2. Our carry-save-adders replace the three first rows with two new rows, and repeat this until only 2 rows remain. We will in the calculations get sums of more than one term, and we ignore lower degree terms to keep the results simple.

Case $(s,t) = (3,5)$:

$$\mathsf{rad}(U_6) = \begin{pmatrix} \rho^{34}/4! & \rho^{34}/4! & \rho^{34}/4! & \rho^{16}/2 & \rho^7 \\ \rho^{52}/48 & \rho^{52}/48 & \rho^{25}/2 & 0 & 0 \end{pmatrix}$$

Case $(s,t) = (3,5)$:

$$\mathsf{rad}(U_6) = \begin{pmatrix} \rho^{70}/8! & \rho^{70}/8! & \rho^{34}/4! & \rho^{16}/2 & \rho^7 \\ \rho^{106}/4! \cdot 8! & \rho^{52}/48 & \rho^{25}/2 & 0 & 0 \end{pmatrix}$$

Case $(s,t) = (3,5)$:

$$\mathsf{rad}(U_6) = \begin{pmatrix} \rho^{70}/8! & \rho^{70}/8! & \rho^{70}/8! & \rho^{34}/4! & \rho^{16}/2 & \rho^7 \\ \rho^{124}/2 \cdot 4! \cdot 8! & \rho^{106}/2 \cdot 4! & \rho^{52}/48 & \rho^{25}/2 & 0 & 0 \end{pmatrix}$$

Step 7

We now add the two last rows using a normal adder to obtain the matrix U_7, and we get the following results.

Case $(s,t) = (3,5)$:

$$\mathsf{rad}(U_7) = \begin{pmatrix} \rho^{115}/(2\cdot 4!) & \rho^{61}/(2\cdot 4!) & \rho^{34}/4! & \rho^{16}/2 & \rho^{7} \end{pmatrix}$$

Case $(s,t) = (4,5)$:

$$\mathsf{rad}(U_7) = \begin{pmatrix} \rho^{133}/(2\cdot 4!\cdot 8!) & \rho^{70}/8! & \rho^{34}/4! & \rho^{16}/2 & \rho^{7} \end{pmatrix}$$

Case $(s,t) = (4,6)$:

$$\mathsf{rad}(U_7) = \begin{pmatrix} \rho^{241}/2(4!\cdot 8!)^2 & \rho^{133}/(2\cdot 4!\cdot 8!) & \rho^{70}/(8!) & \rho^{34}/4! & \rho^{16}/2 & \rho^{7} \end{pmatrix}$$

(s,t)	$\mathsf{rad}(c_{new})$
$(3,5)$	$\frac{\rho^{115}}{(2\cdot 4!)^2}$
$(4,5)$	$\frac{\rho^{133}}{2\cdot 4!\cdot 8!}$
$(4,6)$	$\frac{\rho^{241}}{2(4!\cdot 8!)^2}$

Table 9.1: Table showing the resulting noise of the ciphertext returned from Recrypt for given parameters s and t.

To do the rounding we calculate $(U_7)_0 + (U_7)_1 \cdot (U_7)_2$ and we ignore all terms except for the one with highest degree. The last step is to subtract this from c, but since $c \sim \rho$, we can ignore this last step because the error of c is much smaller than the error of $\lfloor c \cdot B/p \rceil$. We obtain the new ciphertext c_{new} with $\mathsf{rad}(c_{new})$ given in Table 9.1.

So by using Recrypt we can now recrypt a ciphertext to obtain a new ciphertext c_{new} with error bounded by $\mathsf{rad}(c_{new})$. We will use Recrypt to reduce the error of ciphertexts during evaluation of large circuits. More precisely we will apply Recrypt on each ciphertext before we use it as input for either Add or Mult.

A typical case is the one where we have just recrypted two ciphertexts to obtain two relatively clean ciphertexts c_1 and c_2, and we want to multiply the results. Since $\mathsf{rad}(c_1) = \mathsf{rad}(c_1) = \mathsf{rad}(c_{new})$, this gives us a new ciphertext c with $\mathsf{rad}(c) = \delta_\infty \cdot \mathsf{rad}(c_{new})^2$. It must be possible to decrypt c, so we need to choose $\rho = \sqrt{N}$ s.t.

$$\mathsf{rad}(c) = \delta_\infty \cdot \mathsf{rad}(c_{new})^2 \leq \mathsf{r}_{\mathsf{Dec}}/2,$$

where the extra factor of 2 comes from the fact that we reduced $\mathsf{r}_{\mathsf{Dec}}$ by a factor of 2. Table 9.2 gives the results for s_2 between 5 and 14.

s_2	(s,t)	$\mathsf{rad}(c)$	d
5,6,7	(3,5)	$\frac{\rho^{232}}{(2\cdot 4!)^4}$	7
8	(4,5)	$\frac{\rho^{268}}{(2\cdot 4!\cdot 8!)^2}$	7
9,10,11,12,13,14	(4,6)	$\frac{\rho^{484}}{2^2(4!\cdot 8!)^4}$	8

Table 9.2: Table showing the radius of the ciphertext c, and the corresponding depth.

A similar analysis for $(s,t) = (5,7)$, which corresponds to s_2 between 17 and 31, gives a radius $\mathsf{rad}(c)$ of

$$\mathsf{rad}(c) = \frac{\rho^{880}}{(8!)^2 \cdot (4! \cdot 16!)^4}.$$

For $F(x) = x^{2^n} + 1$, $\mu = \sqrt{N}$ and $\eta = 2^{\sqrt{N}}$ we get $\mathsf{r_{Dec}} = 2^{\sqrt{N}}/(2 \cdot \sqrt{N}) = 2^\rho/4\rho$, which shows that for $\rho \geq 11680$ it is possible to obtain a fully homomorphic encryption scheme. This corresponds to $N \geq 136422400$ and $n \approx 27$.

This shows that a fully homomorphic scheme is possible, but very impractical. With $n = 27$, we get $log_2 p \approx 1554950000000$ which corresponds to about 200GB. A ciphertext can be as large as p, which means that each encryption of a bit may take 200GB of memory, so the results are not very promising.

Now we have explained the Recrypt algorithm, and we will use it to recrypt ciphertexts before they are used as inputs for Add and Mult. We then get $\mathsf{Add}_{\mathrm{FHE}}$ and $\mathsf{Mult}_{\mathrm{FHE}}$, algorithms which take as input two ciphertexts c_1 and c_2, and recrypts them before applying Add and Mult respectively.

Our final fully homomorphic encryption scheme then consists of the following five algorithms:

$$(\mathsf{KeyGen}_{\mathrm{FHE}}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Add}_{\mathrm{FHE}}, \mathsf{Mult}_{\mathrm{FHE}}).$$

Chapter 10

Conclusion

In this last section we will try to summarize what we have discovered about our scheme. We will in particular see how the different parameters affect the result, and how they should be set to obtain bootstrappability and sufficient security.

10.1 Theoretical Results

As we saw in Section 5, $F(x) = x^N + 1$, where $N = 2^n$, is a good choice for our irreducible polynomial in KeyGen. For a given N this polynomial is the one which makes $\mathsf{r_{Dec}}$ as large as possible. For this $F(x)$ we get $\delta_\infty = N$. We used this polynomial when we analysed Recrypt earlier, and we will use it also in this final discussion.

We calculated $\mathsf{r_{Dec}}$, the greatest radius a ciphertext can have before it becomes impossible to decrypt:

$$\mathsf{r_{Dec}} = \frac{\sqrt{N} \cdot \eta}{2 \cdot \delta_\infty},$$

which equals

$$\frac{\eta}{2 \cdot \sqrt{N}}$$

for our choice of $F(x)$.

We see that this value depends on N and η. But typically η is a function of N, so for growing N, $\mathsf{r_{Dec}}$ becomes greater if η grows faster than $\sqrt{N}$. We want $\mathsf{r_{Dec}}$ to be as large as possible, so here we prefer an η which grows fast.

In the security section we set the security to be $2^{N/\epsilon}$, where

$$2^\epsilon = \frac{\mathsf{r_{Dec}}}{\mathsf{r_{Enc}}} = \frac{\sqrt{N} \cdot \eta}{2 \cdot \delta_\infty \cdot \mu}.$$

With $\delta_\infty = N$, we then get

$$2^\epsilon = \frac{\eta}{2 \cdot \sqrt{N} \cdot \mu} \Rightarrow \epsilon = \log_2\left(\frac{\eta}{2 \cdot \sqrt{N} \cdot \mu}\right).$$

This gives

$$2^{N/\epsilon} = 2^{\frac{N}{\log_2\left(\frac{\eta}{2 \cdot \sqrt{N} \cdot \mu}\right)}}.$$

This shows that a large N and μ typically increase the security, while a large η makes our scheme more vulnerable. More precisely, if η is of order 2^N, it will be the dominant factor in the logarithm, hence

$$2^{N/\epsilon} \approx 2^{\frac{N}{N}} = 2,$$

which is not secure at all. Hence we should choose an η which grows slower than 2^N. We therefore see that $\eta = 2^{\sqrt{N}}$ is a good choice. This also grows faster than N, so we get a fairly large $\mathsf{r}_{\mathsf{Dec}}$ as well. We will use $\eta = 2^{\sqrt{N}}$ in the remainder of this section.

When we did the error analysis of the Recrypt algorithm we used $\mu = \sqrt{N} \sim \rho$. The choice of μ is important, since it affects the result of this analysis directly. A large choice of μ will therefore result in a much larger error of the ciphertext c_{new} we get after recrypting. This indicates that μ should be as small as possible. But a small μ reduces the security, so we can not set μ to be too small. We earlier established that $\mu = 2$ is the lowest choice of μ we can make. Smart and Vercauteren gives two different choices of μ, namely $\mu = 2$ and $\mu = \sqrt{N}$. We will use both as well.

When we calculated $\mathsf{r}_{\mathsf{Dec}}$ earlier we used the fact that

$$p \simeq \|G(x)\|_2^N \cdot \|F(x)\|_2^{N-1}.$$

And since $\|G(x)\|_\infty \simeq \eta$, we get $\|G(x)\|_2 \simeq \eta \cdot \sqrt{N}$, which leads to

$$p \simeq \sqrt{N}^N \cdot \eta^N \cdot \|F(x)\|_2^{N-1}.$$

Now since $F(x) = x^{2^n} + 1 = x^N + 1$ we get $\|F(x)\|_2 = \sqrt{2}$. Together with our choice of $\eta = 2^{\sqrt{N}}$, this gives us p expressed only by N:

$$p \simeq \sqrt{N}^N \cdot 2^{\sqrt{N} \cdot N} \cdot \sqrt{2}^{N-1}.$$

Now we substitute $N = 2^n$ and take the binary logarithm on both sides and end up with

$$\begin{aligned}
\log_2 p &\simeq \log_2(\sqrt{2^n}^{2^n} \cdot 2^{\sqrt{2^n} \cdot 2^n} \cdot \sqrt{2}^{2^n - 1}) \\
&= \log_2 2^{n \cdot 2^{n-1}} + \log_2 2^{2^{3n/2}} + \log_2 2^{(2^n - 1)/2} \\
&= (n \cdot 2^{n-1}) + 2^{3n/2} + (2^n - 1)/2
\end{aligned}$$

Since the second term is the dominant, we only need to consider that term. We then get

$$\log_2 p \simeq 2^{3n/2}.$$

Recall that we want to avoid a low density for the SSSP. Hence we set $s_1 = \log p$. Now we select s_2 such that

$$\sqrt{\binom{s_1}{s_2}} > 2^{N/\epsilon},$$

which ensures that the difficulty of SSSP is at least as hard as BDDP.

We present two tables with theoretical results. The first one, Table 10.1 shows the results for $\mu = \sqrt{N}$. This table is very comprehensive. The second one, Table 10.2, uses $\mu = 2$, and is less complete.

n	$\log_2 p$	$2^{N/\epsilon}$	$s_1 \sim \log p$	s_2	(s,t)	$\frac{r_{\mathsf{Dec}}}{2}$	$\mathsf{rad}(c)$	d	$\hat{d}$
8	4096	2^{36}	2839	8	(4,5)	1024	$\approx 10^{310}$	0.0	6.4
9	11585	2^{40}	8030	8	(4,5)	71587	$\approx 10^{350}$	0.3	6.4
10	32768	2^{48}	22713	8	(4,5)	$3.3 \cdot 10^{7}$	$\approx 10^{391}$	0.7	6.4
11	92681	2^{61}	64242	9	(4,6)	$2.3 \cdot 10^{11}$	$\approx 10^{777}$	1.2	7.3
12	262144	2^{80}	181704	11	(4,6)	$7.2 \cdot 10^{16}$	$\approx 10^{850}$	1.6	7.3
13	741455	2^{107}	513938	13	(4,6)	$4.8 \cdot 10^{24}$	$\approx 10^{922}$	2.1	7.3
14	$2.1 \cdot 10^6$	2^{144}	$1.4 \cdot 10^6$	17	(5,7)	$6.6 \cdot 10^{35}$	$\approx 10^{1786}$	2.5	8.1
15	$5.9 \cdot 10^6$	2^{298}	$4.1 \cdot 10^6$	22	(5,7)	$4.3 \cdot 10^{51}$	$\approx 10^{1919}$	2.9	8.1
16	$1.7 \cdot 10^7$	2^{274}	$1.1 \cdot 10^7$	28	(5,7)	$\approx 10^{74}$	$\approx 10^{2051}$	3.4	8.1
17	$4.7 \cdot 10^7$	2^{380}	$3.2 \cdot 10^7$	30	(5,7)	$\approx 10^{105}$	$\approx 10^{2184}$	3.8	8.2
26	$5.5 \cdot 10^{11}$	2^{8219}	$3.8 \cdot 10^{11}$	30	(5,7)	$\approx 10^{2461}$	$\approx 10^{3376}$	7.7	8.2
27	$1.6 \cdot 10^{12}$	2^{11613}	$1.1 \cdot 10^{12}$	30	(5,7)	$\approx 10^{3482}$	$\approx 10^{3508}$	8.2	8.2

Table 10.1: Table showing how the parameters change as n increase. We have used $F(x) = x^{2^n} + 1$, $\eta = 2^{\sqrt{N}}$ and $\mu = \sqrt{N}$.

Let us first consider the values in Table 10.1. The first thing to notice is that $\log_2 p$ is very large. Ciphertexts of our scheme may be as large as p, and for $n = 27$ we get ciphertexts which may take as much as $1.6 \cdot 10^{12}$ bits $\approx$ 200GB. This is a large amount of memory spent on the encryption of one single bit!

Usually one require a security level of 2^{80} or more. This means that for $n < 13$ our scheme is not sufficiently secure. However, the security level grows fast as we increase n, and for $n = 27$, the hardness of BDDP is about 2^{11613}, which is more than we ever need. Earlier we decided that s_2 should be chosen such that SSSP becomes as hard as BDDP, but we can for $n = 27$ use a smaller s_2, since the security does not seem to be a problem. In the analysis of recrypt we chose $(s, t) = (5, 7)$, which corresponds to $17 \leq s_2 \leq 31$. With $n = 27$ we get $s_1 \sim 1.1 \cdot 10^{12}$, so even if $s_2 = 30$, we still get a hardness of SSSP of about 2^{546}, which is sufficient. Actually if $n > 16$ it is sufficient with $s_2 = 30$.

Table 10.1 also includes the theoretical values of $\mathsf{r_{Dec}}/2$ and $\mathsf{rad}(c)$. We know that bootstrappability is possible if $\mathsf{rad}(c) < \mathsf{r_{Dec}}/2$, which is true when $n = 27$. We also calculated this when we did the error analysis of Recrypt, with the same conclusion.

We have also included the calculated values of d and $\hat{d}$. The value d is the depth related to the value $\mathsf{r_{Dec}}/2$, i.e. it is the maximum multiplicative depth of circuits we can decrypt correctly. Similarly, $\hat{d}$ is the depth related to $\mathsf{rad}(c)$, i.e. the multiplicative depth we need to manage to obtain bootstrappability. In other words, to get our fully homomorphic scheme, we require that $d > \hat{d}$. This happens when $n = 27$, as we have already seen. d and $\hat{d}$ are calculated by the following two formulas:

$$d \log 2 = \log\log\left(\frac{\mathsf{r_{Dec}}}{2}\right) - \log\log(N \cdot \sqrt{N})$$

$$\hat{d} \log 2 = \log\log\left(\mathsf{rad}(c)\right) - \log\log(N \cdot \sqrt{N}).$$

As we discussed earlier, the values of d and $\hat{d}$ are not very accurate, and they assume that we are working with perfectly balanced circuits. However, they give a better intuition than the values $\mathsf{r_{Dec}}/2$ and $\mathsf{rad}(c)$.

n	$\log_2 p$	$2^{N/\epsilon}$	$s_1 \sim \log p$	s_2	(s, t)	$\frac{\mathsf{r_{Dec}}}{2}$	$\mathsf{rad}(c)$	d	$\hat{d}$
8	4096	2^{26}	2839	6	(3,5)	1024	$\approx 10^{272}$	0.2	6.7
9	11585	2^{32}	8030	6	(3,5)	71587	$\approx 10^{307}$	0.7	6.7
10	32768	2^{41}	22713	7	(3,5)	$3.3 \cdot 10^7$	$\approx 10^{342}$	1.2	6.7
11	92681	2^{54}	64242	8	(4,5)	$2.3 \cdot 10^{11}$	$\approx 10^{431}$	1.7	6.9
12	262144	2^{73}	181704	10	(4,6)	$7.2 \cdot 10^{16}$	$\approx 10^{849}$	2.1	7.8
13	741455	2^{100}	513938	12	(4,6)	$4.8 \cdot 10^{24}$	$\approx 10^{922}$	2.6	7.8

Table 10.2: Table showing how the parameters change as n increase. We have used $F(x) = x^{2^n} + 1$, $\eta = 2^{\sqrt{N}}$ and $\mu = 2$.

In Table 10.2 we use $\mu = 2$. The motivation for this was to increase d as much as possible. As we can see, d is greater with $\mu = 2$ than with $\mu = \sqrt{N}$, as we expected. However, the value of $\hat{d}$ has also increased compared to $\mu = \sqrt{N}$, so the advantage is not crucial. Also notice that our scheme is weaker if $\mu = 2$.

10.2 Implementation Results

In the end we present some of the results of Smart and Vercauteren's actual implementations. This will show how much time the scheme takes to run, which is at least as important as the memory usage. Table 10.3 shows the running time of the algorithms Encrypt, Decrypt, Add and Mult on a desk-top machine. The results are copied directly from [6].

				d	
n	Encrypt	Decrypt	Mult	$\mu = 2$	$\mu = \sqrt{N}$
8	4.2	0.2	0.2	1.0	0.0
9	38.8	0.3	0.2	1.5	1.0
10	386.4	0.6	0.4	2.0	1.0
11	3717.2	3.0	1.6	2.5	1.5

Table 10.3: Running times for the different algorithms.

We do not present results for KeyGen because of its long running time. For $n = 12$, Smart and Vercauteren were unable to generate keys because it took so much time, so it seems impossible to generate keys for $n = 27$. The running time of the Encrypt algorithm seems to multiply by a factor of about 10 for each time we increase n. If this trend continue one encryption may take about a billion years for $n = 27$, in other words, to obtain a fully homomorphic scheme, we need much more computer power than what we have today.

If we look at the depth we are actually able to handle, we see that in practice the scheme performs better than what we expected. This is reasonable since we calculated d for the worst case scenario earlier.

Bibliography

[1] Robert B. Ash. A course in algebraic number theory, 2003. http://www.math.uiuc.edu/ r-ash/ANT.html.

[2] P.B. Bhattacharya, S.K. Jain, and S.R. Nagpaul. *Basic Abstract Algebra*. Cambridge University Press, 1994.

[3] Gu Chunsheng. Cryptanalysis of the smart-vercauteren and gentry-halevis fully homomorphic encryption. Cryptology ePrint Archive, Report 2011/328, 2011. http://eprint.iacr.org/.

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

[5] J. S. Milne. Algebraic number theory, 2011. http://www.jmilne.org/math/.

[6] N.P. 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/.