Masteroppgave i kryptografi

Gudrun Sigfusdottir

Master of Science in Physics and Mathematics
Submission date: September 2015
Supervisor: Kristian Gjøsteen, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

NTNU

How a Fully Homomorphic Encryption Scheme is Created

Guðrún Sigfúsdóttir

Submission date: 17.09.15

Master of Science in Applied Mathematics
Department of Mathematical Sciences
Norwegian University of Science and Technology

Supervisor: Kristian Gjøsteen, IMF

ABSTRACTS

ABSTRACT

In this paper we look at the use of bootstrapping and squashing in order to make an encryption scheme fully homomorphic. The focus will be on what this is and how it can be used. The main focus will be on how this is applied in the paper [11] by van Dijk, Gentry, Halevi and Vaikuntanathan.

SAMMENDRAG

I denne artikkelen ser vi på hvordan et ganske enkelt offentlig nøkkel kryptosystem kan gjøres fullstendig homomorft ved bruk av støvelstropping (bootstrapping) og sammenpressing (squashing). Vi ser først på hva et fullstendig homomorft system er og hva det kan være nyttig for. Hovedfokus vil være på systemet til van Dijk, Gentry, Halevi and Vaikuntanathan presentert i [11].

ÚTDRÁTTUR

I þessari grein munum við skoða hvernig stígvélaþrenging (bootstrapping) og kremjun (squashing) geta nýst til að gera dulkóðunarkerfi algerlega sammótunarlegt. Við byrjum á að skoða hvað það þýðir að kerfi sé algerlega sammótunarlegt og í hvaða kringumstæðum slíkt kerfi er ganglegt. Aðaláhersla verðu lögð á kerfið sem van Dijk, Gentry, Halevi and Vaikuntanathan birtu í [11].

ACKNOWLEDGEMENTS

I would like to thank my supervisor Prof. Kristian Gjøsteen for his guidance, valuable feedback, and especially his patience with me during this process. I would also like to thank Mathilde Skylstad, Kristin Solbakken, and Stine Brekke Vennemo for proof reading my thesis. Finally, I would like to thank my mom for allways beeing there during my studies.

CONTENTS

1 INTRODUCTION 1
- 1.1 Homomorphic Encryption . . . 1
  - 1.1.1 Somewhat Homomorphic Encryption . . . 1
  - 1.1.2 Fully Homomorphic Encryption . . . 2

2 PRELIMINARIES 3
- 2.1 Polynomials . . . 3
- 2.2 Hash functions and Hash families . . . 3
- 2.3 Boolean circuits . . . 5
  - 2.3.1 The three-for-two trick . . . 5
- 2.4 The Approximate Greatest Common Divisor Problem . . . 5

3 A SOMEWHAT HOMOMORPHIC ENCRYPTION SCHEME 7
- 3.1 Construction of van Dijk et. al.'s scheme . . . 7
- 3.2 Correctness . . . 10
- 3.3 Reducing the Ciphertexts . . . 12
  - 3.3.1 Reduction during Evaluate . . . 12
  - 3.3.2 Compression . . . 13
- 3.4 Security . . . 13
  - 3.4.1 Reduction to the Approximate-GCD Problem . . . 13

4 SQUASHING 18
- 4.1 A Modified Encryption Scheme . . . 18

5 BOOTSTRAPPING 21
- 5.1 What is Bootstrapping? . . . 21
- 5.2 A Fully Homomorphic Encryption Scheme . . . 22
- 5.3 Security of the FHE Scheme . . . 25

6 CONCLUSION 26

ABBREVIATIONS AND NOTATIONS

ABBREVIATIONS

Throughout this thesis, we use the abbreviations listed below to ensure a better flow in the text.

FHE	Fully homomorphic encryption
SHE	Somewhat homomorphic encryption
PKS	Public key encryption scheme
SSSP	The sparse subset sum problem
gcd	Greates common divisor
lsb	Least significant bit
FFT	Fast Fourier transform

NOTATIONS

In the following paper, Greek letters will be used to denote parameters, while lower case letters from the Latin alphabet will be used to denote integers and real numbers. All logarithms are base-2, unless otherwise specified. Absolute value of a number is represented by $|\cdot|$. A dollar-sign, \$ over an arrow, $x \xleftarrow{\$} S$, is used to denote that a value x is *chosen at random* from the set S. For a real number x, $\lfloor x \rfloor$ denotes the floor function of x, $\lceil x \rceil$ denotes the ceiling function of x, and $\lfloor x \rceil$ denotes the integer closest to x, with a tie broken upwards.
For two sets A and B, $A \setminus B$ denotes the set containing all elements in A that are not in B. We use lower case boldface letters to denote a vector. For a set A, A^n denotes a vector with parameters from A of length n.
And, just for clearance, a binary point is for binary numbers what the decimal point is for decimal numbers.

1 INTRODUCTION

In this chapter we will introduce homomorphic encryption and the difference between somewhat homomorphic and fully homomorphic on an surface level. We will also have a short discussion about squashing and bootstrapping, how these came to be, and what these mean in the whole of things.

1.1 HOMOMORPHIC ENCRYPTION

Public key encryption is a tool for secure communication, where in principal, only someone who possesses the decryption key can read an encrypted message. The idea that operations can be made on encrypted data, without loosing information, has been around since it was first posed by Rivest et al. [1]. Being able to operate on encrypted data like this is called homomorphic encryption, and is of great significance since it allows a receiver (for example a server) to perform operations on encrypted data, without ever knowing the encryption key.

An encryption scheme that is homomorphic with regards to one particular operation, is called homomorphic with regards to that operation. For example, the well known RSA system is homomorphic with regards to multiplication. That is, the product of two messages can be computed without having to decrypt these messages. However, it is not homomorphic with regards to addition, due to the fact that adding two encrypted messages will not necessarily give the same result as adding the messages before the encryption.

1.1.1 Somewhat Homomorphic Encryption

Some systems are homomorphic with regards to both addition and multiplication for a few iterations. For each iteration errors in the bit-string, also referred to as noise, are added, and at a certain point, when there is too much noise, the system looses its homomorphic abilities. These schemes are said to be *somewhat homomorphic* (SHE). Lots of *public key encryption schemes* (PKS) are what we call SHE schemes, but the idea of a *fully homomorphic* encryption scheme (FHE) was an open problem until Craig Gentry carried it out in [9]. Gentry's scheme is highly inefficient and since 2009 there have been developed several new schemes based on Gentry's method of bootstrapping the decryption circuit. One of these schemes is the DGHV encryption scheme [11], which takes a simple integer-based encryption scheme and makes it fully homomorphic by bootstrapping and squashing the decryption circuit. We will get into what bootstrapping and squashing means in the next Section.

1.1.2 Fully Homomorphic Encryption

An encryption scheme that is homomorphic with regards to both addition and multiplication is referred to as a FHE scheme. The most common way to represent these schemes is as boolean functions that can make up any boolean circuit using only AND and XOR gates. Running encrypted bits through the circuit will yield the same result as running uncrypted bits through the circuit and then encrypting them.

As mentioned earlier, Craig Gentry made the first fully homomorphic encryption scheme in 2009. His method was to take a fairly simple SHE scheme and bootstrap it in order to make it fully homomorphic [9]. His idea was that a bootstrapping algorithm which ensures that the size of the operands stays the same, would make the system fully homomorphic. He created such an algorithm and by that created a FHE scheme. That is, a scheme that is homomorphic with regards to both addition and multiplication for a circuit of arbitrary size. Although his method works, it is far too costly and is therefore not really suitable for implementation. However, despite its limitations, Gentry's scheme has been of great significance. Since its introduction, there have been made several improvements on Gentry's scheme and it forms also the basis for all FHE schemes created since.

In the SHE scheme, ciphertexts will have some noise. *First generation ciphertexts*, which corresponds to ciphertexts encrypted by the algorithm Encrypt defined in Section 3.1, have little noise, but after a certain amount of operations, the noise will increase to unsustainable levels and become undecryptable nonsense.

In Gentry's PhD thesis, he describes bootstrapping and squashing with an example for super suspicious Alice who runs a jewelry store, but does not trust her employees [9]. She has a lot of faults in her security systems and has to find a solution to every problem, which then leads to more problems and more solutions and so on. At the moment, FHE is all about this kind of repetitive solutions trying to keep the noise levels at bay. As far as we know, there is no straight forward way of constructing an FHE scheme, but it can be done by repeating noise-removing algorithms between the multiplication and addition operations.

2

PRELIMINARIES

As we now have introduced the different signature schemes, and will in this chapter move on to the theory needed to understand the rest of this thesis, where a SHE scheme will be made into a FHE scheme.

2.1 POLYNOMIALS

Any symmetric polynomial can be expressed as a sum of elementary symmetric polynomial and we define these as follows:

Definition 2.1 (Elementary symmetric polynomials).
An elementary symmetric polynomial *in n variables, for a $k \geq 0$, is defined as*

$$e_k(x_1, \ldots, x_n) = \sum_{1 \leq i_1 < i_2 < \ldots < i_k \leq n} x_{i_1} \cdots x_{i_k}.$$

2.2 HASH FUNCTIONS AND HASH FAMILIES

Hash functions will be important in the security proof of the SHE scheme, since these can provide assurance of data integrity. Therefore we will now present these and the theory necessary to understand the security proof later on. Definitions 2.2 to 2.4 are necessary in order to fully grasp the definition of *universal hash functions.*

Definition 2.2 (Polynomial parameter [3]).
A parameter a is called a polynomial parameter *if there exists a constant $c_1 > 0$ such that*

$$\frac{1}{c_1} \leq a \leq c_1 n^{c_1}$$

holds for all positive integers n.
If there also exists a constant c_2, such that, for all n, a is computable in time at most $c_2 n^{c_2}$, we say that a is a ***P****-time polynomial parameter.*

Definition 2.3 (Function ensemble [3]).
Let t_n, l_n be integer valued **P***-time polynomial parameters. Let $f : \{0,1\}^{t_n} \to \{0,1\}^{l_n}$ be a function mapping from $\{0,1\}^{t_n}$ to $\{0,1\}^{l_n}$ with respect to n. Then we say that f is a* function ensemble.

Definition 2.4 (**P**-time function ensemble [3]).
We say $f : \{0,1\}^{t_n} \times \{0,1\}^{l_n} \to \{0,1\}^{l_m}$ is a T_n-time function ensemble *if f is a function ensemble such that for all $x \in \{0,1\}^{t_n}$ and all $y \in \{0,1\}^{l_n}$, $f(x,y)$ is computable in time T_n. We say f is a* **P**-time function ensemble *if there is a constant c such that, for all n, $T_n \leq cn^c$.*

In reality, what a hash function actually does is to add a certain number of bits to a bit-string, as an *authentication tag*. This process is defined as follows:

Definition 2.5 (Universal Hash Functions [3]).
Let $h : \{0,1\}^{l_n} \times \{0,1\}^n \to \{0,1\}^{m_n}$ be a **P***-time function ensemble. For a fixed $y \in \{0,1\}^{l_n}$, we view y as describing a function $h(\cdot)$ that maps n bits to m_n bits. Then h is a (pairwise independent)* universal hash function *if, for all $x \in \{0,1\}^n$, $x' \in \{0,1\}^n \setminus \{x\}$, and for all $a, a' \in \{0,1\} m_n$,*

$$\Pr\left[(h_Y(x) = a) \text{ and } (h_Y(x') = a')\right] = \frac{1}{2m_n},$$

where $Y \in \{0,1\}^{l_n}$.

Hash functions are added to data as sort of a digital fingerprint. This means that if the data is altered, the hash function will reveal it by having changed. This is key in what we are discussing. In fact, if you store some data on a non-secure server and the data has a hash function attached to it, you can easily verify its integrity at any time by simply monitoring the hash function. Here is where *hash families* come to play and its definition is as follows:

Definition 2.6 (A Hash Family [8]).
A hash family *is a four-tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$, defined as follows*

$\mathcal{X}$:	*a set of possible* messages *(these are usually bit-strings)*
$\mathcal{Y}$:	*a finite set of possible* authentication tags
$\mathcal{K}$:	*the* keyspace *(finite set of possible keys)*

For each $K \in \mathcal{K}$, there is a hash function *$h_K \in \mathcal{H}$ such that $h_K : \mathcal{X} \to \mathcal{Y}$.*

As can be seen from the definition above for a hash family with many keys there will be several hash functions. As we shall see in the following lemma, if a hash family $\mathcal{H}$ has 2-universal functions, then the resulting functions are dependent on the possible messages $\mathcal{X}$ and the possible authentication tags $\mathcal{Y}$.

Lemma 2.1 (Leftover Hash Lemma [3]). *Define $\mathcal{X}$ and $\mathcal{Y}$ as above and let $\mathcal{H}$ be a family of 2-universal hash functions from $\mathcal{X}$ to $\mathcal{Y}$. Choose $h \xleftarrow{\$} \mathcal{H}$ and $x \xleftarrow{\$} \mathcal{X}$. Then, $(h, h(x))$ is $\frac{1}{2}\sqrt{|\mathcal{Y}|/|\mathcal{X}|}$-uniform over $\mathcal{H} \times \mathcal{Y}$.*

2.3 BOOLEAN CIRCUITS

Some basic knowledge of boolean algebra is necessary to understand the bootstrapping step, performed in Chapter 5. A *boolean circuit* is made up of gates, and in our case only AND and XOR gates as shown in figure 1 will be used.

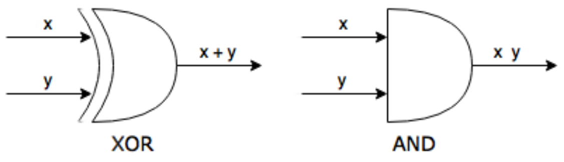

Figure 1: A XOR gate adds bits together and an AND gate multiplies them.

A *subcircuit* is, as the name implies, a smaller circuit contained within a greater circuit. A circuits *level* is the number of connected gates a bit must go through before it reaches the end of the circuit. [6]

2.3.1 The three-for-two trick

The three-for-two trick is a way of using constant-depht boolean circuits to compute the sum $\sum_{i-1}^{k} r_i$ of k rational numbers to transform three numbers of arbitrary bit-lenght into two numbers that are no more than 1 bit longer, where the sum of the two output numbers will still be the same as the sum of the three input numbers. This is explained in more detail in [2].

2.4 THE APPROXIMATE GREATEST COMMON DIVISOR PROBLEM

The approximate greatest common divisor problem is the problem of finding an approximate of some numbers gcd and is defined as follows:

Definition 2.7 (Approximate-gcd Problem [11]).
*To output p, when given polynomially many samples from a distribution $\mathcal{D}$, which is dependant upon variables l, n, for a randomly chosen m-bit odd integer p, is called the (l, m, n)*approximate-gcd problem.

In Chapter 3 the security of a SHE scheme will be proven by reducing it to the approximate-gcd problem and basing the systems security on the security proof for approximate-gcd.

The theory presented above should be a sufficient base to understand the creation of the SHE scheme presented in the next chapter.

3

A SOMEWHAT HOMOMORPHIC ENCRYPTION SCHEME

To create a FHE scheme, it is feasible to use a PKS that is as simple as possible. In this chapter we will see how van Dijk et al. created such a scheme [11].

3.1 CONSTRUCTION OF VAN DIJK ET. AL.'S SCHEME

First we define a few necessary parameters,

γ :	the bit-length of the integers in the public key
η :	the bit-length of the secret key (which is the hidden approximate-gcd of all the public-key integers)
ρ :	the bit-length of the noise (i.e. the distance between the public key and the nearest multiples of the secret key)
τ :	the number of integers in the public key

A fairly simple PKS is defined as a four-tuple of algorithms $\mathcal{E}$ =(KeyGen, Encrypt, Decrypt, Evaluate), where KeyGen, Encrypt, Decrypt, and Evaluate are as follows:

KeyGen:	The key is an odd integer p chosen from the interval $\left[2^{\eta-1}, 2^{\eta}\right)$.
Encrypt(p, m):	To encrypt a bit $m \in \{0, 1\}$, let the ciphertext be an integer with residue modulo p with the same parity (number of 1's) as the plaintext. That is, set $c = pq + 2r + m$, r, q are integers chosen at random from another interval, such that $\lvert 2r \rvert$ is smaller than $\lvert q/2 \rvert$.
Decrypt(p, c):	Outputs $(c \mod p) \mod 2$.
Evaluate$(pk, C, \mathbf{c})$:	Outputs a new ciphertext c.

This PKS will be the basis for our FHE scheme later on.

We choose an efficiently sampleable distribution for a specific, η-bit odd positive integer p over γ-bit integers as follows

$$\mathcal{D}_{\gamma,\rho}(p) = \left\{ q \xleftarrow{\$} \mathbb{Z} \cap \left[0, \frac{2^\gamma}{p}\right), r \xleftarrow{\$} \mathbb{Z} \cap (-2^\rho, 2^\rho) : x = pq + r \right\}.$$

We now define homomorphic decryption and homomorphic encryption. It is straight forward, the decryption is *correct*, if using the decryption algorithm from above on a correctly created ciphertext, results in the plaintext.

Definition 3.1 (Correct Homomorphic Decryption [11]).
*Define $\mathcal{E}$ as the scheme $\mathcal{E}$ =(*KeyGen, Encrypt, Decrypt, Evaluate*). The secret and public keys,* sk *and* pk *respectively, are randomly generated by* KeyGen, KeyGen $\xrightarrow{r}$ (sk,pk)*.*
We say that $\mathcal{E}$ is correct *for a circuit C that takes t bits as input, if for any plaintext bits $m_1, \dots, m_t$ and any ciphertexts $\mathbf{c} = \langle c_1, \dots, c_t \rangle$ with $c_i \leftarrow$*Encrypt*(pk, m_i), then*

$$\mathsf{Decrypt}(\mathrm{sk},\ \mathsf{Evaluate}(\mathrm{pk}, C, \mathbf{c})) = C(m_1, \dots, m_t).$$

At this point it is feasible to introduce a *security parameter*, λ. The vastness of this parameter is not important at this point.

Definition 3.2 (Homomorphic Encryption [11]).
*The scheme $\mathcal{E}$ =(*KeyGen, Encrypt, Decrypt, Evaluate*) is homomorphic for a class $\mathcal{C}_\mathcal{E}$ if $\mathcal{E}$ is correct for $\mathcal{C}_\mathcal{E}$ and $\mathsf{Decrypt}_\mathcal{E}$ can be expressed as a circuit $D_\mathcal{E}$ of size* $\mathrm{poly}(\lambda)$*.*

Definition 3.3 (Compact Homomorphic Encryption [9]).
*The scheme $\mathcal{E}$ =(*KeyGen, Encrypt, Decrypt, Evaluate*) is said to be* compact*, if for every value of λ, there exists a polynomial h, such that $\mathcal{E}$'s decryption algorithm can be expressed as a circuit, $\mathcal{D}_\mathcal{E}$, of size at most $h(\lambda)$.*

Definition 3.4 (Compactly Evaluates [9]).
The scheme $\mathcal{E}$ is said to compactly evaluate *circuits in $\mathcal{C}_\mathcal{E}$ if $\mathcal{E}$ is compact and $\mathcal{E}$ is correct for circuits in $\mathcal{C}_\mathcal{E}$.*

Some constraints on the parameters ρ, η, γ, τ and ρ' are set by the security parameter λ as follows:

$\rho = \omega(\log \lambda)$,	to protect against brute-force attacks on the noise;
$\eta \geq \rho \cdot \Theta(\lambda \log^2 \lambda)$,	in order to support homomorphism for deep enough circuits to evaluate the "squashed decryption circuit";
$\gamma = \omega(\eta^2 \log \lambda)$,	to thwart various lattice-based attacks on the underlying approximate-gcd problem;
$\tau \geq \gamma + \omega(\log \lambda)$,	in order to use leftover hash lemma in the reduction to approximate gcd;
$\rho' = \rho + \omega(\log \lambda)$,	a secondary noise parameter.

In this section we go through the construction of Dijk et. al's encryption scheme, [11], and discuss it's properties, which properties are most important and why.

The algorithms of the scheme $\mathcal{E}$ =(KeyGen, Encrypt, Decrypt, Evaluate) may be constructed as follows:

The schemes private and public keys are created in Algorithm 1, KeyGen(λ), where λ is the systems security parameter.

Algorithm 1 KeyGen(λ)

```
1: procedure CHOOSE A SECRET KEY (sk = $p$, an $\eta$-bit integer)
2:     $p \overset{\$}{\leftarrow} (2\mathbb{Z}+1) \cap \left[2^{\eta-1}, 2^{\eta}\right)$
3: procedure CHOOSE A PUBLIC KEY (pk $=< x_0, x_1, \ldots, x_\tau >$)
4:     while true do
5:         for $i = 0..\tau$ do
6:             $x_i \overset{\$}{\leftarrow} \mathcal{D}_{\gamma,\rho}(p)$
7:             if $x_i > x_0$ then
8:                 $x_i \leftrightarrow x_0$
9:         if $x_0$ is odd and $r_p(x_0)$ is even then
10:            break
```

Algorithm 2 takes the public keys and a message bit, m, as input and outputs the encrypted bit, c.

Algorithm 2 Encrypt(**pk**,$m \in \{0,1\}$)

```
1: Choose a random subset $S$ from $\{1, 2, \ldots, \tau\}$
2: $r \overset{\$}{\leftarrow} (-2^{\rho'}, 2^{\rho'})$
3: Output c $\leftarrow (m + 2r + \sum_{i \in S} x_i) \mod x_0$
```

Algorithm 3, Evaluate(**pk**,$C, c_1, \ldots, s_t$), takes the public keys, a circuit of AND and XOR gates C, and t encrypted bits c_i, $i \in \{1, \ldots, t\}$. It runs the bits through the circuit and outputs a string c.

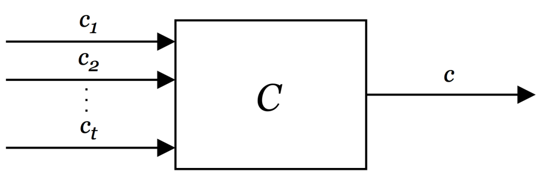

Figure 2: C is a circuit consisting of a total number of t AND and XOR gates.

Algorithm 3 $\mathsf{Evaluate}(\mathbf{pk}, C, c_1, \ldots, c_t)$

```
1: Send the ciphertexts $c_1, \ldots, c_t$ through the circuit $C$    % $C$ is a circuit as in Figure 2.
2: Output $c$    % the output of $C$.
```

Finally, in order to decrypt the messages, there is an algorithm $\mathsf{Decrypt}(\mathbf{sk},c)$ as defined in Algorithm 4.

Algorithm 4 $\mathsf{Decrypt}(\mathbf{sk},c)$

```
1: Output $m' \leftarrow (c \mod p) \mod 2$
```

3.2 CORRECTNESS

It is very important to check that everything is correct and to see how much noise is added in each iteration of the schemes algorithms. In order to check this, it is necessary to define a permitted circuit for the scheme.

Definition 3.5 (Generalised Circuit).
A boolean circuit C made up of only AND *and* XOR *gates has a corresponding* generalised circuit *C' which consists of the corresponding* Mult *and* Add *gates, in the same order, but operates over the integers.*

Definition 3.6 (Permitted Circuit ([10])).
A circuit where for any $\alpha \geq 1$, any set of integers that all have absolute value less than $2^{\alpha(\rho'+2)}$ is a permitted circuit *if the generalised circuits output has absolute value less than or equal to $2^{\alpha(\eta-4)}$.*

We call a ciphertext created by Encrypt a *first generation ciphertext* and for every iteration through Evaluate, the ciphertexts generation value increases by 1.

We see from Definition 3.6 that a first generation ciphertext's noise is at most $2^{\rho'+2}$, and thus the noise of a second generation ciphertext is at most $2^{\eta-4} < \frac{p}{8}$.

Lemma 3.1 (Lemma 3.3 in [11]).
The scheme $\mathcal{E}$ is correct for $\mathcal{C}_{\mathcal{E}}$, where $\mathcal{C}_{\mathcal{E}}$ is the set of permitted circuits.

Proof. We fist show that for $c \overset{\$}{\leftarrow} \mathsf{Encrypt}(\mathbf{pk}, m)$, where m is a bit and $(pk$ is created using $\mathsf{KeyGen}(\lambda)$, it always holds that $c = ap + 2b + m$ for some integers a, b where $|2b + m| \leq \tau 2^{\rho+3}$, presented graphically in Figure 3.

By definition, $c \leftarrow m + 2r + \sum_{i \in S} x_i (\mod x_0)$, where r is a random integer and $|r| \leq 2^\rho$ and S is a random subset of $\{1, 2, \ldots, \tau\}$. Thus

$$c = m + 2r + \sum_{i \in S} x_i + k \cdot x_0, \text{ where } |k| \leq \tau.$$

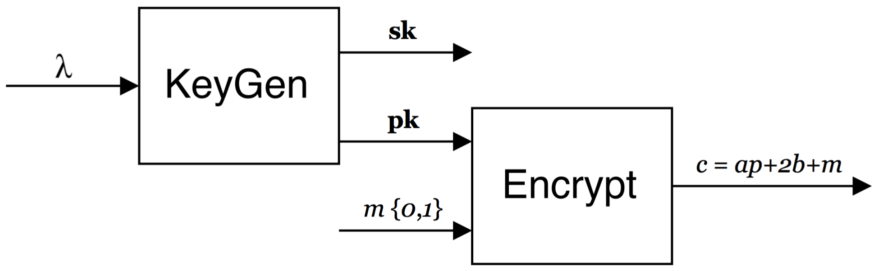

Figure 3: Encryption process for a bit m.

Due to the way the x_i's are constructed, there exist integers q_i's and r_i's, such that $x_i = pq_i + 2r_i$ and $|r_i| \leq 2^\rho$. It now follows that

$$\begin{aligned} c &= m + 2r + \sum_{i\in S}(pq_i + 2r_i) + k(q_0 p + 2r_0) \\ &= p\left(kq_0 + \sum_{i\in S} q_i\right) + m + 2r + 2kr_0 + 2\sum_{i\in S} r_i. \end{aligned}$$

Then $a = (kq_0 + \sum_{i\in S} q_i)$ and $b = 2r + 2kr_0 + 2\sum_{i\in S} r_i$ and

$$\begin{aligned} |m + 2r + 2kr_0 + \sum_{i\in S} r_i| &\leq (4\tau + 3)2^\rho \\ &\leq \tau 2^{\rho+3}. \end{aligned}$$

We now show that when repeating this process for t different bits m_i, $(i \in \{1, 2, \ldots, t\})$, and then evaluating the corresponding c_i's over a circuit $C \in \mathcal{C}_\mathcal{E}$, it holds that the resulting $c \leftarrow$Evaluate$(\mathbf{pk}, C, c_1, \ldots, c_t)$ is equal to $ap + 2b + m$, for some integers a and b, such that $|2b + m| < \frac{p}{8}$. This process is graphically demonstrated in Figure 4.

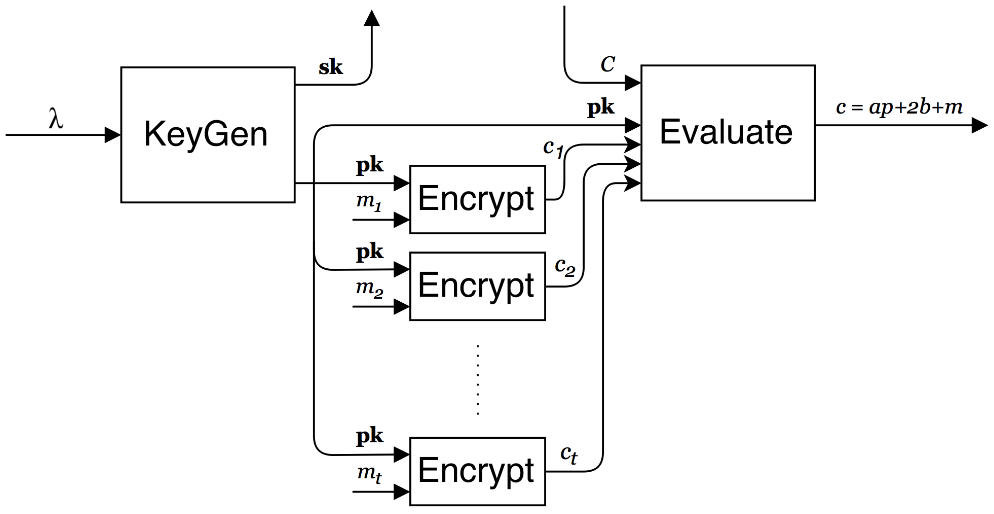

Figure 4: The encryption of a t-bit message m made up of the bits $m_1, m_2, \ldots, m_t$, using Evaluate$(\mathbf{pk}, C, c_1, \ldots, c_t)$, where the c_i's are encryptions of the corresponding m_i's, for $i \in \{1, 2, \ldots, t\}$, and $C \in \mathcal{C}_\mathcal{E}$.

If C' is the generalised circuit operating over the integers that corresponds to C then $C'(c_1, c_2, \dots, c_t) = C'(2b'_1 + m_1, 2b'_2 + m_2, \dots, 2b'_t + m_t) + p\mathbb{Z}$. This implies that $C'(2b'_1 + m_1, 2b'_2 + m_2, \dots, 2b'_t + m_t) \mod p$ has the same parity as $m = C(m_1, m_2, \dots, m_t)$. We have already proved that $|2b_i + m_i| \leq \tau 2\rho + 3$ and from this it follows that $|C'(2b'_1 + m_1, 2b'_2 + m_2, \dots, 2b'_t + m_t)| \leq \frac{2^\eta}{16} \leq \frac{p}{8}$ by the restrictions on the parameters and the definition of p. From this, it follows directly that $c = 2b + m \mod p$ and thus $m = (c \mod 2) \mod p$. $\mathcal{E}$ is hence correct for $\mathcal{C}_\mathcal{E}$. □

Lemma 3.2 (Lemma 3.5 in [11]).
Let C be a boolean circuit with t inputs. Let C' be its corresponding generalised circuit and $f(x_1, \dots, x_t)$ be the multivariate polynomial with degree d computed by C'. Then $C \in \mathcal{C}_\mathcal{E}$, where $\mathcal{C}_\mathcal{E}$ is the set of permitted circuits, if the length of $\mathbf{f}$ *is at most $\frac{2^{\eta-4}}{2^{d(\rho'+2)}}$.*

Definition 3.7 (Permitted polynomials [11]).
It follows directly from Lemma 3.2 *that*

$$d \leq \frac{\eta - 4 - \log|\mathbf{f}|}{\rho' + 2}. \tag{1}$$

A polynomial for which this holds is referred to as a permitted polynomial*. The set of permitted polynomials is called $\mathcal{P}_\mathcal{E}$ and the circuits that compute these are called $\mathcal{C}(\mathcal{P}_\mathcal{E})$.*

3.3 REDUCING THE CIPHERTEXTS

3.3.1 Reduction during Evaluate

Remark 3.1. Adding two integers will result in an integer with length of at most one larger than the original integers. However, multiplying two integers will result in an integer with length up to double the length of the original integers. Thus, it is obvious that the Mult gates are the bottlenecks of the permitted circuits.

Due to the multiplication that occurs in Evaluate, the ciphertexts produced by this will mostly be too large for a simple reduction (like the reduction modulo x_0 in Encrypt). A way of modular reduction in Evaluate is to create more public key elements as follows:

```
for $i = 0..\gamma$ do
    $r'_i \xleftarrow{\$} \mathbb{Z} \cap (-2^\rho, 2^\rho)$
    $q'_i \xleftarrow{\$} \mathbb{Z} \cap \left[\frac{2^{\gamma+i-1}}{p}, \frac{2^{\gamma+i}}{p}\right)$
    $x'_i \leftarrow 2(q'_i \cdot p + r'_i)$
```

and when a ciphertext c surpasses 2^γ in length in Evaluate, it is reduced as follows:

$$c = \left(\left(c \mod x'_\gamma\right) \mod x'_{\gamma-1}\right) \dots \mod x'_0,$$

which will result in the new c having a length less than or equal to γ.

3.3.2 Compression

van Djik et. al. suggest the parameter set

$$\rho = \lambda,\ \rho' = 2\lambda,\ \eta = \tilde{O}(\lambda^2),\ \gamma = \tilde{O}(\lambda^5), \text{ and } \tau = \gamma + \lambda$$

for the SHE scheme. However, this set of parameters results in ciphertexts of length $\tilde{\theta}(\lambda^5)$. There is a way to compress the ciphertexts to more manageable lengths, but these can not be bootstrapped. Therefore, the compression must take place on the last generation of ciphertexts before deciphering. That is, the ciphertexts cannot go through Evaluate after they have been compressed. This compression reduces the ciphertext to (asymptotically) the size of an RSA modulus.

3.4 SECURITY

3.4.1 Reduction to the Approximate-GCD Problem

For the security proof of the above SHE scheme, we reduce it to the Approximate-GCD Problem, in Theorem 3.5. In order to prove Theorem 3.5, we first need to introduce a couple of lemmas.

Lemma 3.3.
For $M, T \in \mathbb{Z}$,

$x_1, x_2, \ldots, x_T \xleftarrow{\$} \mathbb{Z}_M$,
$\mathbf{s} \xleftarrow{\$} \{0,1\}^T$, *and*
$x_{T+1} \leftarrow \sum_{i=1}^{T} s_i \cdot x_i \mod M$.

Then, $(x_1, x_2, \ldots, x_T, x_{T+1})$ *is* $\frac{1}{2}\sqrt{\frac{M}{2^T}}$*-uniform over* $\mathbb{Z}_M^{T+1}$.

Proof. We define a hash family, $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$, where $\mathcal{X}$ is the set of bit-strings of length T, and $\mathcal{Y}$ is $\{\mathbb{Z}_{\mathbb{M}}\}$ such that the hashes are $(h_1, h_2, \ldots, h_T) \in \mathbb{Z}_M^T$.
We choose an $\mathbf{s} \in \mathcal{X}$ and see that $h(\mathbf{s}) = \sum_{i=1}^{T} s_{ii} \in \mathbb{Z}_M$, which implies that the hash family is 2-universal, and the leftover hash lemma (Lemma 2.1) then gives us that $(h, h(x))$ is $\frac{1}{2} \cdot \sqrt{M/2^T}$-uniform over $\mathbb{Z}_M^{T+1}$. □

Lemma 3.4.
For the parameters $(\rho, \rho', \eta, \gamma, \tau)$ *from above,* sk $= p$, *and* pk$= \langle x_0, x_1, \ldots, x_\tau \rangle$ *chosen at random in* Keygen, *then any integer* $x^* \in [0, 2^\gamma]$, *at most* 2^ρ *away from a multiplum of* p, *consider the following distribution:*

$$\mathcal{C}_{\text{pk}}(x^*) = \left\{ S \subseteq_\$ \{1, \ldots, \tau\}, r \xleftarrow{\$} (-2^{\rho'}, 2^{\rho'}) : \textit{output } c' \leftarrow x^* + 2r + \sum_{i \in S} x_i \mod x_0 \right\}.$$

Then, every distribution $\mathcal{C}_{\text{pk}}$ *is statistically close to the distribution* Encrypt(pk, $m = [x^*]_2$) *(up to a negligible statistical distance).*

Proof. Writing $c' = q'p + 2r' + m$, we would like to prove that both q' and r' are distributed as in the scheme.

We start by proving that q' is distributed by the scheme. The quotient $q_p(c)$ of the ciphertext is uniform in $\left(-\frac{q_0}{2}, \frac{q_0}{2}\right]$ by the leftover hash lemma (2.1). From the fact that c' is created the same way as in Encrypt, and Lemma 3.3 it follows that q' is also uniform in $\left(-\frac{q_0}{2}, \frac{q_0}{2}\right]$.

Now we look at r', and from the way the scheme is created, we have additional noise for the ciphertext in both the scheme and the reduction. Due to the magnitude of the distribution being significantly larger than the noise of the public-key elements, the added noise statistically drowns the possible differences caused by the public-key elements and the integer x^*, in the noise distribution. ☐

Theorem 3.5 (Theorem 4.2 in [11]).
Define $(\rho, \rho', \eta, \gamma, \tau)$ as above. Then any attack $\mathcal{A}$ with advantage ϵ on the scheme can be converted into an algorithm $\mathcal{B}$ that solves the (ρ, η, γ)-approximate-gcd problem with success probability at least $\epsilon/2$. The running time of $\mathcal{B}$ is polynomial in the running time of $\mathcal{A}$, in λ, and in $1/\epsilon$.

Proof. Let $z = q_p(z) \cdot p + r_p(z)$ where, per usual, $q_p(z)$ is the quotient, and $r_p(z)$ is the remainder of z with regards to p. $\mathcal{A}$ is an attacker against the scheme. A graphic representation of $\mathcal{A}$'s attack strategy is shown in Figure 5

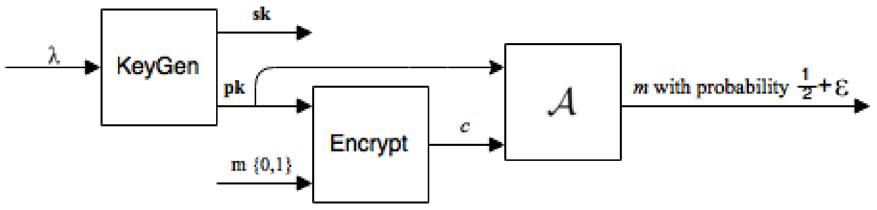

Figure 5: $\mathcal{A}$'s attack finds the message m, using **pk** and the encrypted message c, with probability $1/2 + \epsilon$

First off, we use $\mathcal{A}$ to construct $\mathcal{B}$. $\mathcal{B}$ is a solver for approximate-gcd with parameters ρ, η, and γ.

For a randomly chosen η-bit odd integer p, the solver $\mathcal{B}$ has access to as many samples from $\mathcal{D}_{\gamma,\rho}(p)$ as it needs, and the goal is to find p.

In Lemma 3.4 it was shown that for all but a negligible fraction of the public keys generated by the scheme, the ciphertext c_j is distributed almost identically to a valid encryption of the bit $[r_p(z)]_2 \oplus \mathsf{parity}(z)$. It follows that if $\mathcal{A}$ has a noticeable advantage in guessing the encrypted bit under pk, then Learn-LSB(z,pk) will return $q_p(z) \mod 2$ with overwhelming probability.

Since $\mathcal{A}$ is now an oracle for the lsb of $q_p(z)$ we can recover p in the following way:

Algorithm 5 $\mathcal{B}$

```
1: procedure CREATING A PUBLIC KEY(pk = $\langle x_0, x_1, \ldots, x_\tau \rangle$)
2:     while true do
3:         for $i = 0..\tau$ do
4:             $x_i \overset{\$}{\leftarrow} \mathcal{D}_{\gamma,\rho}(p)$
5:             if $x_i > x_0$ then
6:                 $x_i \leftrightarrow x_0$
7:         if $x_0$ is odd then
8:             break
9:     Output pk = $\langle x_0, x_1, \ldots, x_\tau \rangle$
10: procedure LEARN-LSB($z$,pk)
11:     Input: $z \in [0, 2^\gamma)$ with $|r_p(p)| < 2^\rho$, a public key pk = $\langle x_0, x_1, \ldots, x_\tau \rangle$
12:     for $j = 1..\frac{\text{poly}(\lambda)}{\epsilon}$ do          % $\epsilon$ is the overall advantage of $\mathcal{A}$.
13:         $r_j \overset{\$}{\leftarrow} (-2^{\rho'}, 2^{\rho'})$
14:         $x_j \overset{\$}{\leftarrow} \{0, 1\}$
15:         $S_j \subseteq \{1, \ldots, \tau\}$
16:         $c_j \leftarrow (z + m_j + 2r_j + 2 \sum_{k \in S_j} x_k) \mod x_0$
17:         $a_j \leftarrow \mathcal{A}(\text{pk}, c_j)$
18:         $b_j \leftarrow a_j \oplus \mathsf{parity}(z) \oplus m_j$          % $b_j$ should be the parity of $q_p(z)$.
19:     Output: The majority vote among the $b_j's$          % This is the least-significant-bit of $q_p(z)$.
```

Given two integers, $z_1 = q_p(z_1) \cdot p + r_p(z_1)$ and $z_1 = q_p(z_1) \cdot p + r_p(z_1)$, where $r_p(z_i) \ll p$, $i = 1, 2$.

Algorithm 6 Finding p

```
Repeat the following process:
if $z_1 < z_2$ then
    $a = z_2$
    $z_2 = z_1$
    $z_1 = a$
for $i = 1, 2$ do
    $b_i = q_p(z_i) \mod 2$ found using $\mathcal{A}$
if $q_p(z_1)$ is odd AND $q_p(z_2)$ is odd then
    $z_1 = z_1 - z_2$
    $b_1 = 0$
for $i = 1, 2$ do
    if $b_i = 0$ then $z_i = (z_i - \mathsf{parity}(z_i))/2$ % Note that $z_i - \mathsf{parity}(z_i)$ is even, so the new $z_i$ is an integer.
```

By the Algorithm 6 we observe that when $p \gg r_p(z_i)$, changing the parity-bit will not effect the quotient. It follows that in this instance, $q_p(z_i' - \mathsf{parity}(z_i)) = q_p(z_i)$.

Plugging this into the last line of Algorithm 6, we get:

$$\begin{aligned} z_i' &= \frac{z_i - \mathsf{parity}(z_i)}{2} \\ \Rightarrow q_p(z_i')p + r_p(z_i') &= \frac{q_p(z_i - \mathsf{parity}(z_i))p + r_p(z_i - \mathsf{parity}(z_i))}{2} \\ \Rightarrow q_p(z_i') &= \frac{q_p(z_i - \mathsf{parity}(z_i))}{2} \\ \text{and } r_p(z_i') &= \frac{r_p(z_i - \mathsf{parity}(z_i))}{2} \end{aligned} \tag{2}$$

And since the parity of $q_p(z_i)$ is even (by the nature of Algorithm 6), $q_p(z_i') = q_p(z_i)/2$. From the Equation 2, it follows directly that $|r_p(z_i')| \leq (|r_p(z_i)| + 1)/2 \leq |r_p(z_i)|$. After another iteration of the algorithm, we set $z_1' = z_1 - z_2$ and get that $z_1'' = (z_1' - \mathsf{parity}(z_1'))/2$, and thus

$$\begin{aligned} r_p(z_1'') &= \frac{r_p(z_1') - \mathsf{parity}(z_1'))}{2} \\ &= \frac{r_p(z_1) - r_p(z_2) - \mathsf{parity}(z_1)}{2}, \end{aligned}$$

which further implies that $|r_p(z_1'')| \leq \max\{|r_p(z_1)|, |r_p(z_2)|\}$ that is, the $r_p(z_i)$'s will never grow beyond the first $r_p(z_i)$, which confirms that $p \gg r_p(z_i)$.

This proves that Algorithm 6 finds the binary gcd of two integers and it takes the algorithm $O(\gamma)$ operations to find integers z_1' and z_2', with z_2' as the odd part of $\gcd(q_p(z_1), q_p(z_2))$.

We now move on to recovering p. $\mathcal{B}$ is used to draws two elements $z_1^*, z_2^* \overset{\$}{\leftarrow} \mathcal{D}_{\gamma,\rho}(p)$. Then, p is found using Algorithm 6 on z_1^*, z_2^*. The probability of the $\gcd(z_1^*, z_2^*) = 1$ is $1/\zeta(2)$, where $\zeta(2)$ is the *Riemann zeta function* with $s = 2$ [5]. Since $\zeta(2) = \pi^2/6$ and thus the probability that the odd part of $\gcd(q_p(z_1^*), q_p(z_2^*))$ equals 1 is $\pi^2/6$. From this, we have that the final z is $\acute{z} = 1 \cdot p + r$ with probability $|r| \leq 2^\rho$. $\mathcal{B}$ will draw again and again until $\acute{z} = 1 \cdot p + r$. Now, set $z_1 = z_1^*$ and $z_2 = \acute{z}$ in Algorithm 6, and $\mathcal{B}$ recovers $p =_1^* /q_p(z_1^*)\rceil$. This proves that given an oracle for computing $q_p(z) \mod 2$, the algorithm $\mathcal{B}$, converted from $\mathcal{A}$, can find p.

It is now time to prove that $\mathcal{B}$'s success probability is at least $\epsilon/2$. Lemma 3.4 states that for every secret key p and the respective public keys pk, created by *Keygen* (Algorithm 1). We let $\mathcal{P}$ be the set of odd integers in $[2^{\eta-1}, 2^\eta)$ for which $\mathcal{A}$ has more than $\epsilon/2$ advantage. We denote this advantage by $\mathsf{advantage}(\mathcal{A}) = \epsilon/2$, and define

$$\mathcal{P} = \Big\{ p \in [2^{\eta-1}, 2^\eta) : \ \mathsf{advantage}(\mathcal{A}) \text{conditioned on } p \text{ is at least } \epsilon/2 \Big\}$$

By double counting, we see that there must be at least $\epsilon/2$ elements in $\mathcal{P}$, and it follows that $\mathsf{advantage}(\mathcal{A}) \geq \epsilon/4$ for each public key pk.

If we now run $\mathcal{B}$ once, where it has access to $\mathcal{D}_{\gamma,\rho}(p)$ for some $p \in \mathcal{P}$. The probability of the public key being produced by $\mathcal{B}$ being negligibly close to the right distribution is 0.5 by the definition of our scheme. Thus, $\mathcal{B}$ produces the correct pk with probability $\epsilon' \geq \epsilon/4 - \mathsf{negl}$ (negl denotes a negligible amount). From this and Lemma 3.4 we can further deduce that $a_i = \mathcal{A}(\mathrm{pk}, c_j)$ in Algorithm 5 is correct with probability $\epsilon/4 - \mathsf{negl}$. Due to the size of the for-loop in the LEARN -LSB, it will return the right answer with overwhelming probability, and $\mathcal{B}$ will recover the approcimate-gcd p.

It finally follows that when $p \in \mathcal{P}$, the probabilitiy of $\mathcal{B}$ recovering it in just one run of the algorithm is greater than or equal to $0.5 \cdot (\epsilon/4 - \mathsf{negl})$. Then, running the algorithm $(8/\epsilon) \cdot \omega(\log \lambda)$ times will return the correct p with overwhelming probability. This implies that the success probability of $\mathcal{B}$ is at least the density of $\mathcal{P}$, which is $\epsilon/2$.

□

4

SQUASHING

In this section we will explain the act of squashing the decryption circuit in detail. We will start by introducing some sum problems and then go into detail on how these can be applied to the squashing algorithm.

In all existing schemes, the squashing technique induces an additional assumption: that the sparse subset sum problem (SSSP) is hard.

The subset sum problem is finding a non-empty subset of a given integer set, whose sum is zero. The SSSP hardness assumption is that if for a sufficiently large set of integers and a target sum, not necessarily 0, it is assumed hard to find a sparse subset sum that hits the target.

Definition 4.1 (The Sparse subset sum problem, [7])**.**
We let A be of positive integers $a_1, a_2, \dots, a_n$ and construct a linear combination $s = \sum_{i=1}^{n} m_i a_i$ where $m_i \in \mathbb{Z}$ for all i and $r = \sum_{i=1}^{n} m_i^2$ is small. The sparse subset sum problem (SSSP) *is to recover the m_i's.*

4.1 A MODIFIED ENCRYPTION SCHEME

For simpler notation, three more parameters are now introduced as functions of λ. These are $\kappa = \gamma\eta/\rho'$, $\theta = \lambda$, and $\Theta = \omega(\kappa \cdot \log \lambda)$. To make the scheme bootstrappable, it needs to be tweaked. Therefore the scheme from Section 3.1 will now be referred to as $\mathcal{E}^*$ and all variables created using that original scheme will get the same treatment of having a star, (*), added to them.

The new scheme $\mathcal{E}$ consists of Algorithms 7-9.

Since all $0 < u_i < 2^{\kappa+1}$ and $y_i = u_i/2^\kappa$, then $0 < y_i < 2$ with κ bits of precision after the binary point and $[\sum_{i \in S} y_i]_2 = \frac{1}{p} - \Delta_p$ for some $|\Delta_p| < \frac{1}{2^\kappa}$.

Algorithm 7 KeyGen(λ)

```
1: Produce sk*,pk* as in Algorithm 1
2: $x_p \leftarrow \lfloor 2^\kappa / p \rceil$
3: $\mathbf{s} = \langle s_1, \dots, s_\Theta \rangle$    % $\mathbf{s}$ is a $\Theta$-bit vector, chosen at random and with Hamming weight $\theta$.
4: $S = \{i : s_i = 1\}$
5: while true do
6:     for $i = 1..\Theta$ do
7:         $u_i \xleftarrow{\$} \mathbb{Z} \cap [0, 2^{\kappa+1})$
8:     if $\sum_{i \in S} u_i = x_p \mod 2^{\kappa+1}$ then
9:         break
10: for $i = 1..\Theta$ do
11:     $y_i = u_i / 2^\kappa$
12: $\mathbf{y} = \{y_1, \dots, y_\Theta\}$
13: Output (sk,pk)= $(\mathbf{s}, (\text{pk}^*, \mathbf{y}))$
```

Algorithm 8 Encrypt and Evaluate

```
1: Choose a random subset $S$ from $\{1, 2, \dots, \tau\}$
2: $r \xleftarrow{\$} (-2^{\rho'}, 2^{\rho'})$
3: $c^* \leftarrow [m + 2r + \sum_{i \in S} x_i]_{x_0}$
4: for $i = 1..\Theta$ do
5:     $z_i \leftarrow [c^* \cdot y_i]_2$    % with only $n = \lceil \log \theta \rceil + 3$ bits of precision after binary point
6: $\mathbf{z} = \langle z_1, \dots, z_\Theta \rangle$
7: Output $(c^*, \mathbf{z})$
```

Algorithm 9 Decrypt(**sk**,c)

```
1: Output $m' \leftarrow [c^* - \lfloor \sum_i s_i z_i \rceil]_2$
```

Lemma 4.1 (Lemma 6.1 in [11]).
The scheme $\mathcal{E}$ is correct for $\mathcal{C}(\mathcal{P}_\mathcal{E})$

Proof. Create a private and a public key as in Algorithm 7. Let $\{y_i\}_{i=1}^{\Theta}$ denote the rational numbers in the public key and let $\{s_i\}_{i=1}^{\Theta}$ denote the secret key.

Determine a permitted polynomial $P(x_1, x_2, \dots, x_t) = \mathcal{P}_\mathcal{E}$, an arithmetic circuit C which calculates P, and t ciphertexts $\{c_i\}_{i=1}^{t}$ that encrypts C's input. Then let

$$c^* = \mathsf{Evaluate}(pk, C, c_1, \dots, c_t)$$

We must establish that

$$\left\lfloor \frac{c^*}{p} \right\rceil = \left\lfloor \sum_i s_i z_i \right\rceil \mod 2$$

where for every i, $z_i = c^* y_i \mod 2$ with only $\lceil \log \theta \rceil + 3$ bit precision after the binary point. Using that the y_is were created in a way such that $\sum_i s_i y_i \mod 2 = \frac{1}{p} - \Delta_p$ where $|\Delta_p| \leq 2^{-\kappa}$. Thus $c^* y_i \mod 2 = z_i - \Delta_i$ with $|\Delta_i| \leq \frac{1}{16\theta}$ and we now see that

$$\begin{aligned}\left(\frac{c^*}{p} - \sum s_i z_i\right) \mod 2 &= \left(\frac{c^*}{p} - \sum s_i (c^* y_i \mod 2) + \sum s_i \Delta_i\right) \mod 2 \\ &= \left(\frac{c^*}{p} - \sum c^* (s_i y_i \mod 2) + \sum s_i \Delta_i\right) \mod 2 \\ &= \left(\frac{c^*}{p} - c^* \left(\frac{1}{p} - \Delta_p\right) + \sum s_i \Delta_i\right) \mod 2 \\ &= \left(c^* \Delta_p + \sum s_i \Delta_i\right) \mod 2.\end{aligned}$$

We observe that $|\sum s_i \Delta_i| \leq \theta \cdot \frac{1}{16\theta} = \frac{1}{16}$. Since c^* is created by evaluating the polynomial P over the ciphertext inputs c_i. Per definition, P's input magnitude is less than or equal to $2^{\alpha(\rho'+2)}$ for every $\alpha \geq 1$, and given that an input has magnitude, the outputs magnitude is less than or equal to $2^{\alpha(\eta-4)}$. ,When P's input is of first generation (and thus its magnitude is at most 2^γ), the ciphertext c^* has magnitude less than or equal to $2^{\gamma(\eta-4)/(\rho'+2)} < 2^{\kappa-4}$. Thus $|c^* \cdot \Delta_p| < \frac{1}{16}$ and $|c^* \cdot \Delta_p + \sum s_i \Delta_i| < \frac{1}{8}$. Since c^* is per definition a valid output from a permitted polynomial, $\frac{c^*}{p}$ is within a distance $\frac{1}{8}$ from an integer and together, these prove the Lemma.

□

5

BOOTSTRAPPING

In this chapter we learn what bootstrapping is and how it can be used to make the encryption scheme fully homomorphic. Where does it come from and what are the uses.

5.1 WHAT IS BOOTSTRAPPING?

In [10], the idea is to take a SHE scheme and make it fully homomorphic by creating a scheme that is *bootstrappable*. Bootstrapping is the act of removing noise in the ciphertext. Bootstrapping is in essentials taking a homomorphic encryption scheme that can compactly evaluate its own decryption circuit and making it capable of encryption for circuits of arbitrary depth. For a scheme to be bootstrappable, it needs to be capable of evaluating slightly augmented versions of its decryption circuit, in addition to of course being able to evaluate the circuit itself. This capability is necessary when noise comes into play and allows for nontrivial operations on plaintexts and continuous progress throughout the circuit.

Definition 5.1 (Augemented Decryption Circuits [11]).
Let the scheme $\mathcal{E}$ =(KeyGen, Encrypt, Decrypt, Evaluate) *be such that for a fixed value of* λ *(the security parameter), the size of the secret key must be fixed and the cipertexts must be the same length. Then the set of* augmented decryption circuits, $D_{\mathcal{E}}$ *contains two circuits,*

$$\begin{aligned} C_{+}(sk, c_1, c_2) &\to m_1 + m_2 \mod 2 \text{ and} \\ C_{\cdot}(sk, c_1, c_2) &\to m_1 \cdot m_2 \mod 2\ , \end{aligned}$$

where c_1 *and* c_2 *are two distinct ciphertexts and* m_1 *and* m_2 *are their corresponding decryptions.*

Definition 5.2 (Bootstrappable Encryption [11]).
A homomorphic encryption scheme $\mathcal{E}$ is bootstrappable *if*

$$\mathcal{D}_{\mathcal{E}}(\lambda) \subseteq \mathcal{C}_{\mathcal{E}}(\lambda)$$

holds for every λ and $\mathcal{C}_{\mathcal{E}}(\lambda)$ is the corresponding set of circuits for which the set of decryption circuits $\mathcal{D}_{\mathcal{E}}$ will return the correct plaintexts.

Theorem 5.1 (Theorem 2.7 in [11]). *For a bootstrappable encryption scheme $\mathcal{E}$ and a parameter d, there is an efficient transformation which outputs a description of another encryption scheme $\mathcal{E}^{(d)}$ such that*

$\mathcal{E}^{(d)}$ is compact and the Decrypt *circuit in $\mathcal{E}^{(d)}$ and $\mathcal{D}$ are identical, and*

$\mathcal{E}^{(d)}$ is homomorphic for all circuits of depth up to d.

Theorem 5.2 (Theorem 4 in [10]). *$\mathcal{E}^{(d)}$ is* semantically secure *if any attack with advantage ϵ on $\mathcal{E}^{(d)}$ can be converted into an attack on $\mathcal{E}$ with advantage at least $\frac{\epsilon}{ld}$, where l is the length of the secret key in $\mathcal{E}$.*

Definition 5.3 (Fully Homomorphic Encryption [10]).
*The scheme $\mathcal{E}$ =(*KeyGen, Encrypt, Decrypt, Evaluate*) is fully homomorphic if it is homomorphic for all circuits.*

We now apply the the theory presented above to the scheme, in order to make it an FHE scheme.

5.2 A FULLY HOMOMORPHIC ENCRYPTION SCHEME

Lemma 5.3 (Lemma 6.3 in [11]).
Let $\boldsymbol{\varrho} = \varrho_1, \varrho_2, \dots, \varrho_t$ be a binary vector, let $W = W(\boldsymbol{\varrho})$ denote the number of non-zero coordinates in $\boldsymbol{\varrho}$, and let $W_n W_{n-1} \dots W_0$, where all W_i's are bits and $W = \sum_{i=0}^{n} 2^i W_i$, denote the binary representation of W.
Then for every $i \leq n$, the bit $W_i(\varrho)$ can be expressed as a binary polynomial of degree exactly 2^i in the variables $\varrho_1, \varrho_2, \dots, \varrho_t$. Moreover, there is an arithmetic circuit of size $2^i \cdot t$ that simultaneously computes all the polynomials for $W_0, \dots, W_i$.

Proof. Set $e_k(\cdot)$ to be the k'th elementary symmetric polynomial. We use that the i'th bit in the binary representation of the Hamming weight (number of elements that are non-zero) of $\boldsymbol{\varrho} = e_{2^i}(\boldsymbol{\varrho}) \mod 2$ [4]. Thus,

$$W_i(\varrho) = e_{2^i}(\varrho) \mod 2 = \left(\sum_{|S|=2^i} \prod_{j \in S} \varrho_j \right) \mod 2.$$

and it follows directly that the degree of $e_{2^i}(\boldsymbol{\varrho})$ is 2^i.

The coefficients of the polynomial $P_{\boldsymbol{\varrho}}(z) = \prod_{i=1}^{t}(z - \varrho_i)$ are the elementary symmetric polynomials in the ϱ_i's, where $e_k(\boldsymbol{\varrho})$ is the coefficient of z^{t-k}. The lower-order terms in $P_{\boldsymbol{\varrho}}(z)$ are negligible and we can ignore the z^j's for $j < t - 2^i$, for the first $i + 1$ W_l's $(l = 0, \cdots, i)$.

One way of finding these polynomials is expressed in Algorithm 10.

Algorithm 10

```
1: **Input**: bits $\varrho_1, \ldots, \varrho_t$
2: $P_{0,0} = 1$
3: **for** $j = 1..2^i$ **do**
4:     $P_{j,0} = 0$
5: **for** $k = 1..t$ **do**
6:     **for** $j = 2^i, 2^{i-1}, ..1$ **do**
7:         $P_{j,k} = \varrho_k \times P_{j-1,k-1} + P_{j,k-1}$
8: **Output**: $P_{1,t}, P_{2,t}, P_{4,t}, \ldots, P_{2^i,t}$
```

Multiplying the polynomials using FFT [12], we can find the entire polynomial $P_{\boldsymbol{\varrho}}(z)$ with complexity $t \cdot \text{polylog}(t)$.

□

Theorem 5.4 (Theorem 6.2 in [11]).
Let $\mathcal{D}_{\mathcal{E}}$ denote the set of decryption circuits for the scheme $\mathcal{E}$. Then, $\mathcal{D}_{\mathcal{E}} \subset C(\mathcal{P}_{\mathcal{E}})$.

To make the scheme fully homomorphic, bootstrapping needs to be applied to the squashed scheme from Chapter 4. For this to be true, Theorem 5.4 must hold. The proof of this theorem is largely based on the three-for-two trick from Section 2.3.

Proof. To show that $\mathcal{D}_{\mathcal{E}} \subset C(\mathcal{P}_{\mathcal{E}})$ is the same as showing that $\mathcal{E}$s decryption circuit, $m' \to (c^* - \lfloor \sum_i s_i z_i \rceil) \mod 2$, is a permitted polynomial and then show that there exists a circuit of polynomial size that can solve this polynomial.

Since $c^* \in \mathbb{Z}$, all $s_i \in \{0, 1\}$, and all $z_i \in \mathbb{Q} \cap [0, 2)$ in binary with $n = \lceil \log \theta \rceil + 3$ bits of precision after the binary point, we know that $\sum s_i z_i$ is within $\frac{1}{4}$ of an integer and that only θ of the bits $s_1, \cdots, s_\Theta$ are non-zero.

First, for $i \in \{1, 2, \ldots, \Theta\}$, set $a_i \to s_i z_i \; \forall i \in \mathbb{Q}$, then $a_i \in \mathbb{Q} \cap [0, 2)$. From this it is clear that $a_i = z_i$ when $s_i = 1$ and $a_i = 0$ otherwise.

Now, using the a_i's we create $n+1$ other rational numbers $\{w_j\}_{j=0}^{n}$ that all have precision of less than n bits, such that $\sum_j w_j = \sum_i a_i \mod 2$. We use the three-for-two trick at most $\lceil \log_{3/2} k \rceil + 2$ times, resulting in two numbers s_1 and s_2 such that $s_1 + s_2 = \sum_{i-1}^{k} r_i$. This implies that it takes a circuit of depth $d' \leq 2^{\lceil \log_{3/2} k \rceil + 2} < 8k^{1/\log(3/2)} < 8k^{1.71}$ to reduce k numbers into only two numbers .

In general, circuits need to be of a level that is logarithmic to the bit-length of their input numbers in order to compute the final sum of those numbers. However, since it is given that $s_1 + s_2$ are within $\frac{1}{4}$ of an integer and we are only interested in $\lfloor s_1 + s_2 \rceil \mod 2$, all we need to compute is a multivariate polynomial of degree 4. From this we deduce that in order to compute $\left\lfloor \sum_{i=1}^{k} r_i \right\rceil \mod 2$, the circuit's corresponding polynomial, whose coefficient vectors l_1-norm is at most 27^d, is of degree $d \leq 32k^{1/\log(3/2)}$. When $k = \Theta$, this polynomial degree is still a very large, and therefore we use the following technique (from [9]):

Denote the bit representation of each number a_i by $a_{i,0}a_{i,-1}a_{i,-n}$, where $a_i = \sum_{j=0}^{n} 2^{-j} a_{i,-j}$.

Compute integers W_{-j}, $j = 0, 1, \ldots, n$, where W_{-j} is the number of non-zero elements of the "column" of bits $(a_{1,-j}, a_{2,-j}, \ldots, a_{\Theta,-j})$, as represented below

$$\begin{array}{ccccc}
a_{1,0} & a_{1,-1} & a_{1,-2} & & a_{1,-n} \\
a_{2,0} & a_{2,-1} & a_{2,-2} & & a_{2,-n} \\
a_{3,0} & a_{3,-1} & a_{3,-2} & & a_{3,-n} \\
 & & & \cdots & \\
a_{\Theta,0} & a_{\Theta,-1} & a_{\Theta,-2} & & a_{\Theta,-n} \\
\hline
W_0 & W_{-1} & W_{-2} & & W_{-n}
\end{array}$$

By our parameter settings, we know that at most θ of the a_i's are non-zero, and thus the W_{-j}'s are smaller than or equal to θ, which implies that the W_{-j}'s can be represented by $\lceil \log(\theta + 1) \rceil < n$ bits, and Lemma 5.3 says that each of the bits in w_{-j} can be expressed as a polynomial of degree at most θ. Moreover, all of these polynomials can be computed simultaneously by an arithmetic circuit of size $O(\theta \cdot \theta)$.

$$a_i = \sum_{j=0}^{n} 2^{-j} a_{i,-j} \Rightarrow \sum_i a_i = \sum_j 2^{-j} W_{-j},$$

We now define $w_j = (2^{-j} \cdot W_{-j}) \mod 2$, for all $j \in \{0, n\}$ so the w_j's are rational numbers with $\lceil \log(\theta + 1) \rceil < n$, and using the three-for-two trick we obtain the sum of the a_i's $\mod 2$.

This means that the degree of the polynomials in the first step is two, the degree of the polynomials in the second step is at most θ, and the degree of the polynomials in the third step is at most

$$32(n+1)^{1/\log(3/2)} < 32\lceil \log\theta + 4\rceil^{1.71} < 32\log^2\theta.$$

The total degree of the decryption circuit is therefore bounded by $2 \cdot \theta \cdot 32\log^2\theta = 640\log^2\theta$. In this case $\theta = \lambda$, and thus the degree is at most $64\lambda\log^2\lambda$.

It follows that the augmented decryption circuits $\mathcal{D}_{\mathcal{E}}$ can be expressed as polynomials of degree at most $128\lambda\log^2\lambda$ in the Θ variables s_i. Since the logarithm of the l_1-norm of this polynomial is relatively small compared to η, and $\Theta = \frac{\eta\gamma}{\rho} \cdot \omega(\log\lambda) < \lambda^7$. Plugging these values into 1, we can deduce that it is sufficient to set $\eta = \rho' \cdot \Theta(\lambda\log^2\lambda)$, which in return indicates that we can get $\mathcal{D}_{\mathcal{E}} \subset C(\mathcal{P}_{\mathcal{E}})$, making the scheme bootsrappable, by setting $\eta = \rho \cdot \Theta(\lambda\log^2\lambda))$.

$\square$

Note that our first circuit implementation of the procedure from above is not "shallow". Still, since it computes only low degree polynomials (up to degree 2^i), then by Lemma 3.2 it is a permitted circuit.

5.3 SECURITY OF THE FHE SCHEME

As mentioned in Chapter 4, the new scheme relies on the additional assumption that the SSSP (Definition 4.1 is hard. Based on the hardness assumptions, the scheme can be made secure by choosing θ hard enough too withstand brute-force attacks and Θ larger than $\omega(\log\lambda)$, which follows quite literally from the parameter definitions.

6

CONCLUSION

In conclusion, what has been done above is that the meaning of a scheme being fully homomorphic and how that can be achieved was discussed. The FHE scheme created by van Dijk et. Al. was presented in full detail and the security of this scheme was taken into consideration.

For further work, it would be interesting to look closer at this scheme's time consumption. An even bigger undertaking would be to try and envision a new way of creating a FHE scheme, which would not need to be repeatedly bootstrapped.

BIBLIOGRAPHY

[1] R. L. Rivest, A. Shamir, and L. Adleman. “A Method for Obtaining Digital Signatures and Public-key Cryptosystems”. In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126. issn: 0001-0782. doi: `10.1145/359340.359342` (cit. on p. 1).

[2] Richard M. Karp. *A Survey of Parallel Algorithms for Shared-Memory Machines*. Tech. rep. Berkeley, CA, USA, 1988 (cit. on p. 5).

[3] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. “A Pseudorandom Generator from any One-way Function”. In: *SIAM J. COMPUT.* 28.4 (1999), pp. 1364–1396 (cit. on pp. 3, 4).

[4] Joan Boyar, René Peralta, and Denis Pochuev. “On the multiplicative complexity of Boolean functions over the basis (,,1)”. In: *Theoretical Computer Science* 235.1 (2000), pp. 43–57. issn: 0304-3975. doi: `http://dx.doi.org/10.1016/S0304-3975(99)00182-6` (cit. on p. 22).

[5] Harold M Edwards. *Riemann’s zeta function*. Vol. 58. Courier Corporation, 2001 (cit. on p. 16).

[6] Jón Hafsteinn Jónsson, Níels Karlsson, and Stefán G. Jónsson. *STÆ 513*. Tölvunot ehf, 2003 (cit. on p. 5).

[7] PhongQ. Nguyn and Jacques Stern. “Adapting Density Attacks to Low-Weight Knapsacks”. English. In: *Advances in Cryptology - ASIACRYPT 2005*. Ed. by Bimal Roy. Vol. 3788. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 41–58. isbn: 978-3-540-30684-9. doi: `10.1007/11593447_3` (cit. on p. 18).

[8] Douglas R. Stinson. *Cryptography, theory and pactice*. Third edition. 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742: Chapman & Hall/CRC, an imprint of Taylor and Francis Group, 2006 (cit. on p. 4).

[9] Craig Gentry. “A fully homomorphic encryption scheme”. `crypto.stanford.edu/craig`. PhD thesis. Stanford University, 2009 (cit. on pp. 1, 2, 8, 24).

[10] Craig Gentry. “Fully homomorphic encryption using ideal lattices”. In: *Stoc ’09*. ACM, 2009, pp. 169–178 (cit. on pp. 10, 21, 22).

[11] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. “Fully homomorphic encryption over the integers”. In: *Advances in cryptology–EUROCRYPT 2010*. Springer, 2010, pp. 24–43 (cit. on pp. II, 2, 5, 7–10, 12, 14, 19, 21–23).

[12] Yan-Bin Jia. “Polynomial Multiplication and Fast Fourier Transform”. In: (2014) (cit. on p. 23).