Royal Holloway, University of London

A Survey of Fully Homomorphic Encryption

MSc Dissertation in Mathematics of Cryptography and Communications
Submitted by

Igors Stepanovs

2012

Abstract

Fully homomorphic encryption allows anyone to compute arbitrary functions over encrypted data without the knowledge of a decryption key or the data itself. It has numerous possible powerful applications, including secure cloud computing and encrypted interaction with search engines.

The question of existence of fully homomorphic encryption schemes was first raised in 1978 by Rivest, Adleman and Dertouzos [39]. However, it was only in 2009 when Gentry presented the first efficient and secure encryption scheme which possesses the desired properties [21].

In this work we discuss the theory of homomorphic encryption and review several encryption schemes with such properties. We provide a comprehensive analysis of Gentry's construction, and also give a brief summary of the earlier work related to homomorphic encryption. We describe two fully homomorphic encryption schemes. The first scheme is based on the approximate-GCD problem over the integers, and the second scheme is based on the lattice-based LWE problem. We discuss the "modulus switching" technique and finish with an overview of the attempts to implement fully homomorphic encryption in practice.

Contents

1 Introduction **5**

1.1 Contributions 6

1.2 Dissertation Outline 6

1.3 Introduction to Cryptography 7

2 Homomorphic Encryption **12**

2.1 Introduction to Homomorphic Encryption 12

2.2 Polly Cracker Cryptosystem 14

2.3 Early Homomorphic Encryption Schemes 16

2.4 Security of the Homomorphic Encryption Schemes 17

3 Fully Homomorphic Encryption **19**

3.1 Definition of Fully Homomorphic Encryption 19

3.2 Noisy Encryption Schemes 23

3.3 Bootstrappable Encryption Schemes 24

4 Fully Homomorphic Encryption over the Integers **29**

4.1 Encryption Scheme 29

4.2 Efficiency of the Scheme 31

4.3 Security of the Scheme 32

4.4 Managing the Noise 34

4.5 Squashing the Decryption Circuit 36

5 Lattice-based Encryption Scheme **38**

5.1 Introduction to Lattices 38

5.2 LWE-based Encryption Scheme 39

5.2.1 Re-Linearization Technique 41

5.2.2 Dimension-Modulus Reduction Technique 42

5.3 Modulus-Switching Technique 44

6 Implementing Homomorphic Encryption Schemes **47**
6.1 Gentry's Encryption Scheme 47
6.2 Encryption Scheme over the Integers 48
6.3 Further Optimizations 49

7 Conclusions **50**

Bibliography **51**

List of Notation

$\{0,1\}^n$	all binary strings of length n
$\log x$	$\log_2 x$, unless the base is specified explicitly
$\langle a_1, ..., a_n \rangle$	n-tuple of some scalar values
$[z]_p$	z mod p for $z \in \mathbb{R}$, $p \in \mathbb{Z}$ and with the result in $(-p/2, p/2]$
$\lfloor z \rceil$	real value z rounded up to the nearest integer
$x \xleftarrow{\$} S$	sample x from a set S uniformly at random
$x \leftarrow \mathsf{Alg}$	x is an output of algorithm Alg
$\mathsf{Alg}_x(...)$	algorithm Alg takes a value x as its first input argument
$\mathbb{Z}_q^n$	set of n-tuples with integers modulo q
$\mathbb{Z}_q[x_1, ..., x_n]$	set of n-variate polynomials with integer coefficients modulo q
$\mathbb{Z}_q^{m \times n}$	set of matrices with m rows and n columns with integer values modulo q
$\mathbf{v}$	vectors are denoted by lower-case bold letters
$\mathbf{A}$	matrices are denoted by upper-case bold letters
$\lvert\mathbf{v}\rvert_1$	l_1-norm of vector $\mathbf{v}$
$\langle \mathbf{v}, \mathbf{w} \rangle$	inner product of vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$
$\lVert\mathbf{v}\rVert$	Euclidean norm of a vector $\mathbf{v}$
$f(n) \in \mathcal{O}(g(n))$	$f(n) \leq k \cdot g(n)$ for some positive k
$f(n) \in \omega(g(n))$	$f(n) > k \cdot g(n)$ for all positive k
$f(n) \in \Theta(g(n))$	$k_1 \cdot g(n) \leq f(n) \leq k_2 \cdot g(n)$ for some positive k_1, k_2
$f(n) \in \tilde{\mathcal{O}}(g(n))$	$f(n) \in \mathcal{O}(g(n) \log^k g(n))$ for some $k \in \mathbb{N}$

Chapter 1

Introduction

The purpose of encryption is to protect the confidentiality of data from a possible adversary. Traditionally, encryption also limits the further processing of data. Once the data is encrypted, we can only store it or decrypt it. However, there are many applications which require the ability to operate on encrypted data (see [20, Section 1.8] and [36, Section 2.2]). For instance, we may use a remote untrusted server for storing our files. In order to preserve the confidentiality of our data, we encrypt the stored files. Now assume we want to find the files containing particular phrases. The simple and inefficient solution would be to download all files locally so that we can decrypt them one by one and search through their contents. Alternatively, we might want the server to provide this functionality for us, without revealing the contents of the encrypted data. This brings us to the question whether it is possible to operate on encrypted data without decrypting it at any point.

More formally, we are given encryptions of unknown plaintexts $c_1 \leftarrow \mathsf{Encrypt}(m_1), ...,$ $c_\ell \leftarrow \mathsf{Encrypt}(m_\ell)$ and we want to compute some function f on these plaintexts. A *homomorphic encryption* scheme allows *anyone* to compute an encryption of the result (i.e. $c_f \leftarrow \mathsf{Encrypt}(f(m_1, ..., m_\ell))$) while knowing only the function f and the ciphertexts $c_1, ..., c_\ell$. Any information about the plaintext messages must always remain hidden.

Throughout this work we discuss encryption schemes which possess different levels of homomorphism. When speaking about *homomorphic encryption*, we refer to the encryption schemes which are capable of computing *any* functions over encrypted data. We can further classify homomorphic encryption schemes into *fully homomorphic* and *somewhat homomorphic*, which can or cannot evaluate *all* efficiently computable functions respectively.

In 1978 Rivest, Adleman and Dertouzos [39] were the to first to ask whether there

exists a cryptosystem which allows to operate on encrypted data. Back then this was called *privacy homomorphisms*. For over 30 years a lot of different solutions were offered. However, all of them were either inherently inefficient, or limited to a very small set of homomorphically computable functions. Due to a wide range of possible applications, the existence of fully homomorphic encryption was one of the central open problems in cryptography, sometimes referred as the "Holy Grail" of cryptography.

Finally, in 2009 Gentry [21, 20] offered the first viable construction of a fully homomorphic encryption scheme. During the following years many improvements of the initial construction were subsequently offered. In this work we discuss the recent advancements in fully homomorphic encryption, starting from the initial blueprint offered by Gentry, and building up on it.

1.1 Contributions

The only currently available survey of the recent developments in fully homomorphic encryption is [45]. It briefly covers all of the results related to this area at the time of its publication in year 2011. However, it does not explain the covered topics in depth, only giving a sketch of a few main definitions and techniques.

In this work we attempt to identify the most important results in fully homomorphic encryption and provide their overview with sufficient details. At the same time we also try to give a brief summary of the topics which were not given a more thorough look within this work. We refer to a number of results which were published in 2012 and are currently state of the art.

1.2 Dissertation Outline

The remainder of this chapter provides an introduction to cryptography, defining public-key encryption and the related security notions. Chapter 2 offers an informal discussion about the basics of homomorphic encryption. It gives an example of a homomorphic encryption scheme and outlines the history of the encryption schemes with the desired properties. The chapter ends with a discussion about the security of homomorphic encryption.

Chapter 3 provides the formal definitions of homomorphic encryption and presents the blueprint of constructing fully homomorphic encryption schemes that were initially offered by Gentry [21, 20] and subsequently used by most of the recently proposed schemes. Chapter 4 shows the construction of the conceptually simplest known fully homomorphic encryption scheme over the integers from [47], using the previously de-

scribed blueprint.

Chapter 5 describes an encryption scheme based on the lattice-based LWE problem, as well as the "re-linearization" technique which is required to make this scheme efficient. Further on, the "dimension-modulus reduction" and "modulus-switching" techniques are introduced, which considerably simplify the construction of this and other fully homomorphic encryption schemes.

Chapter 6 discusses the recent attempts to implement fully homomorphic encryption schemes in practice. Finally, Chapter 7 provides the conclusions of this work.

1.3 Introduction to Cryptography

Assume we want to hide the contents of a sensitive message (called *plaintext*). Cryptosystems provide a way to achieve this by using a key to *encrypt* the plaintext message into another message (called *ciphertext*) which looks like gibberish. Further on, we can use the key to *decrypt* ciphertext back into plaintext.

According to the Kerckhoff's principle, the security of a cryptosystem should be based only on the secrecy of the used key, and not on the obfuscation of the encryption algorithm.

There are two kinds of encryption algorithms:

- *Symmetric encryption scheme* uses the same secret key for encryption and decryption. Therefore, any two people willing to communicate between themselves need to agree on their shared secret key first; it is hard to do that safely. And it is also inconvenient for everyone to store many keys. Nevertheless, symmetric encryption schemes are normally very fast and are used as often as possible.

- *Asymmetric encryption scheme* (also called *public-key cryptosystem*) uses two different keys for encryption and decryption. The message encrypted with someone's *public key*, can only be decrypted using his *private key* (also called *secret key*). The key management becomes easy – each user only needs to store his own private key, while all public keys are publicly available. However, asymmetric encryption is much slower than symmetric encryption. Normally it is only used to agree on a shared secret key, and the rest of the communication is then encrypted with some symmetric encryption scheme which uses the obtained shared key.

Homomorphic encryption schemes can be either symmetric or asymmetric, but in this work we will put more focus on the asymmetric case. Below we give a definition of a public key cryptosystem and introduce the related security notions. However,

everything we discuss can also be adapted to the symmetric case, by making minor formal adjustments.

Definition 1.3.1. *A public-key encryption scheme with plaintext space $\mathcal{P}$ and ciphertext space $\mathcal{C}$ is a 3-tuple of algorithms $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ satisfying the conditions below:*

- KeyGen *takes a unary representation of a security parameter λ as input and outputs a randomized pair of keys (pk, sk) where pk is a public key and sk is the corresponding secret key.*
- $\mathsf{Encrypt}$ *takes a public key pk and a plaintext $m \in \mathcal{P}$ as input and outputs a ciphertext $c \in \mathcal{C}$. This algorithm may be randomized.*
- $\mathsf{Decrypt}$ *takes a private key sk and a ciphertext $c \in \mathcal{C}$ as input and outputs a plaintext $m \in \mathcal{P}$. This algorithm is usually deterministic.*
- *(Decryption Correctness) For any key pair (pk, sk) which might be output by* KeyGen *and for any plaintext $m \in \mathcal{P}$ we require $m = \mathsf{Decrypt}(sk, \mathsf{Encrypt}(pk, m))$.*

It can be assumed without loss of generality that the plaintext space $\mathcal{P}$ can be determined from the public key pk. We can define the plaintext space as the set of plaintext messages m for which the algorithm $\mathsf{Encrypt}(pk, m)$ runs successfully (for example, does not return the "incorrect message" symbol $\perp$). Then we define the ciphertext space as $\mathcal{C} = \{c \mid c \leftarrow \mathsf{Encrypt}(pk, m) \text{ for all } m \in \mathcal{P}\}$. In this work we will normally encrypt each bit separately and hence use the plaintext space $\mathcal{P} = \{0, 1\}$.

In order to formalize the security level of a cryptosystem, we introduce a security parameter λ which specifies the bit-length of the keys. We normally want that the fastest successful attack against the cryptosystem runs in time exponential in λ. In turn, all the algorithms of the cryptosystem should run in time polynomial in λ. Generally, any algorithm is expected to be polynomial in its input length, so we normally pass a unary representation of λ to the KeyGen algorithm to indicate the requirement from above. We do not explicitly indicate it for the other algorithms of a cryptosystem.

We define an *adversary* as a probabilistic algorithm which runs in time polynomial in λ. A cryptosystem is secure if every possible adversary has at most a negligible probability (as a function of λ) of breaking it. Below we give the definition of a negligible function.

Definition 1.3.2 (Negligible Function)**.** *A function $f : \mathbb{N} \to \mathbb{N}$ is said to be negligible, and denoted $negl(k)$, if $\forall c > 0$ there exists $k_0 \in \mathbb{N}$ such that $\forall k \geq k_0 \implies f(k) < \frac{1}{k^c}$.*

As an example of a public-key encryption scheme we provide a definition of RSA cryptosystem below. We will briefly discuss its homomorphic properties in Chapter 2.

Definition 1.3.3 (RSA Cryptosystem [38])**.** *The security of the RSA public-key cryptosystem relies on the difficulty of factoring large integers.*

- KeyGen(*security parameter* λ)*: outputs public key* (e, N) *and private key* (d, N)*.*

  The algorithm first generates a λ*-bit modulus* $N = p \cdot q$ *where* p *and* q *are random primes of approximately the same bit-size. It then finds the encryption and decryption exponents* e *and* d *such that* $ed \equiv 1 \bmod \phi(N)$ *where* ϕ *is the Euler totient function.*

- Encrypt(*public key* (e, N)*, plaintext* m*): outputs ciphertext* $c = m^e \bmod N$*.*

- Decrypt(*private key* (d, N)*, ciphertext* c*): outputs plaintext* $m = c^d \bmod N$*.*

  The decryption algorithm uses an observation that $c^d \bmod N = m^{ed} \bmod N = m^{k\phi(N)+1} \bmod N = m \bmod N$ *holds for every* $k \in \mathbb{N}$*, because the Euler's theorem states that* $m^{\phi(N)} \equiv 1 \bmod N$ *for any co-prime integers* m *and* N*.*

The security of a cryptosystem can be viewed as a game in which an adversary is provided with one or two challenge ciphertexts and a public key (which is always available for anyone). It is then required to recover some information about the corresponding plaintexts or the private key. The cryptosystem is considered secure with respect to some "security game" if every polynomial-time adversary has at most a negligible probability of winning the game. There exist the following types of security games:

- Key Recovery – Given a ciphertext, the adversary needs to recover the corresponding private key.

- One-Way Encryption – Given a ciphertext, the adversary needs to recover the corresponding plaintext.

- Semantic Security – The adversary chooses a plaintext space $\mathcal{P}$ and is given the ciphertext which corresponds to some plaintext $m \in \mathcal{P}$. The adversary then chooses a deterministic function f and attempts to guess the value of $f(m)$ with a probability (more than negligibly) better than without knowing the provided ciphertext. Basically, the semantic security means that it is infeasible to use a ciphertext in order to derive any partial information about the corresponding plaintext.

- Indistinguishability (IND) – The adversary chooses two plaintexts of the same length m_0, m_1 and receives the encryption $c \leftarrow \mathsf{Encrypt}(m_b)$ where $b \in \{0,1\}$ is selected at random. He needs to guess the value of b with a probability (more than negligibly) greater than $\frac{1}{2}$.

  One can observe that a cryptosystem can only meet the indistinguishability requirement if its encryption function is randomized. Otherwise the adversary can simply use the public key to compare the encryptions of plaintexts m_0, m_1 with the challenge ciphertext c. Is is proven that indistinguishability is equivalent to semantic security [26].

In all of the above security games, the keys and the challenge plaintexts are always being chosen uniformly at random.

We can consider an adversary in the following attack models:

- Chosen Plaintext Attack (CPA) – The adversary knows the public key, hence it can encrypt plaintext messages of its choosing.
- Chosen Ciphertext Attack (CCA1) – The adversary knows the public key and has access to a decryption oracle. The oracle is able to decrypt arbitrary ciphertexts of the adversary's choosing, but only before the adversary receives the challenge ciphertext(s).
- Adaptive Chosen Ciphertext Attack (CCA2) – The adversary knows the public key and has access to a decryption oracle. The oracle is able to decrypt arbitrary ciphertexts of the adversary's choosing, both before and after the adversary receives the challenge ciphertext(s). However, once the adversary received the challenge ciphertext(s), it is no longer allowed to query the received ciphertext(s) to the oracle.

The newly proposed public key encryption algorithms are normally expected to meet the requirements of indistinguishability under an adaptive chosen ciphertext attack (IND-CCA2).

Definition 1.3.4 (IND-CCA2 Security)**.** *Consider the following "security game" which is played between an adversary and a challenger:*

(1) The challenger runs $\mathsf{KeyGen}(1^\lambda)$ *to generate* (pk, sk)*.*

(2) The adversary is given 1^λ*, the public key* pk *and access to a decryption oracle* $\mathsf{Decrypt}(sk, *)$*. The adversary can perform at most a polynomial number of operations in the security parameter* λ*.*

(3) The adversary chooses two messages $m_0, m_1 \in \mathcal{P}$ of equal length and gives them to the challenger.

(4) The challenger chooses $b \in \{0,1\}$ uniformly at random and returns the challenge ciphertext $c^* \leftarrow \mathsf{Encrypt}(pk, m_b)$.

(5) Same as (2), but the adversary can no longer query the decryption oracle on c^*.

(6) The adversary outputs a guess $b' \in \{0,1\}$ and wins the "security game" if $b = b'$.

The advantage of the adversary is defined as $Pr[b = b'] - \frac{1}{2}$. We say that the public-key encryption scheme is IND-CCA2 secure if every polynomial-time adversary wins this "security game" with at most negligible advantage, $negl(\lambda)$.

If we no longer allow the use of the decryption oracle in the stage (5) of the IND-CCA2 security game, then we get the definition of IND-CCA1. Further on, if we also remove the decryption oracle from the stage (2), then we get the definition of IND-CPA.

IND-CPA security is considered to be the absolute minimum requirement for asymmetric encryption schemes because the CPA attack model is always available to the adversary due to its knowledge of the public key. When we speak about the semantic security without mentioning particular attack models, we normally mean IND-CPA security.

The last security game we will discuss deals with a non-malleability property which is most often also considered outside of particular attack models. All of the previously discussed security games deal with the confidentiality of data (i.e. the requirement that the adversary cannot understand the encrypted message). In contrast, the non-malleability property deals with the integrity of data (i.e. the requirement that the adversary can not unnoticeably alter the message):

- Non-malleability – Given the ciphertext c which corresponds to some plaintext m, the adversary needs to find a different ciphertext $c' = \mathsf{Encrypt}(m')$ such that the relation $m' = g(m)$ between m and m' is known to the adversary.

If a cryptosystem is malleable, then it is not IND-CCA2 secure. Assume an adversary receives some challenge ciphertext c during the IND-CCA2 game, and he needs to determine whether it is the encryption of plaintext m_0 or m_1 (both known by the adversary). Since the cryptosystem is malleable, the adversary can compute $c' \leftarrow \mathsf{Encrypt}(g(m_0))$ for some known relation g. Now it can use the decryption oracle to query the decryption $m' \leftarrow \mathsf{Decrypt}(sk, c')$ and, finally, compare m' with $g(m_0)$ and $g(m_1)$, hence winning the security game.

Chapter 2

Homomorphic Encryption

This chapter provides an outline of homomorphic encryption. We give an informal definition of homomorphic encryption, followed by an example of an inefficient fully homomorphic encryption scheme. We then offer a summary of the early homomorphic encryption schemes and an overview of the security of homomorphic encryption.

2.1 Introduction to Homomorphic Encryption

Remember we are trying to solve the problem of efficiently computing arbitrary functions over encrypted data. Assume we have some plaintext bits $m_1, m_2, ..., m_\ell \in \{0,1\}$ and we want a third party to compute some function $f : \{0,1\}^\ell \to \{0,1\}$ for us. However, we can only reveal the encrypted version of our data, namely, $c_1 \leftarrow \mathsf{Encrypt}(m_1), c_2 \leftarrow \mathsf{Encrypt}(m_2), ..., c_\ell \leftarrow \mathsf{Encrypt}(m_\ell)$. Therefore, we want them to be able to compute the encrypted value of our function $c_f \leftarrow \mathsf{Encrypt}(f(m_1, m_2, ..., m_\ell))$ while knowing only f and $c_1, c_2, ..., c_\ell$. The secret key and the plaintext values must remain unknown. We want to receive the encrypted result c_f, then we will decrypt it ourselves.

Note that we can represent any computable function as a program which can be run on a universal Turing machine. And any program can be essentially represented as data. Therefore, we would also be able to hide (i.e. encrypt) the functions we are evaluating. Specifically, we would use an unecrypted universal Turing machine which takes some encrypted data as input. The encrypted data would then represent both the program (i.e. function) we want to compute and the data which is being accepted by this encrypted program.

This might give an impression that we are capable of obfuscating our programs which was proven impossible in [5]. However, in case of the obfuscation, the input and

the output of the program must be unencrypted. In contrast, homomorphic encryption provides no way to use unencrypted input/output with an encrypted function.

Before we discuss the computation of arbitrary functions, let us consider a simpler case – computing additions and multiplications over encrypted data.

Definition 2.1.1 ((Informal) Additively and Multiplicatively Homomorphic Encryption)**.** *Consider a cryptosystem $\mathcal{E}$ which is defined over plaintext space $\mathcal{P}$ and ciphertext space $\mathcal{C}$. We say that*

- *$\mathcal{E}$ is additively homomorphic if for any given encryption key and any plaintexts $m_1, m_2 \in \mathcal{P}$ the following condition is satisfied:* $\mathsf{Encrypt}(m_1 + m_2) \equiv_{\mathcal{E}} \mathsf{Encrypt}(m_1) +_{\mathcal{C}} \mathsf{Encrypt}(m_2)$
- *$\mathcal{E}$ is multiplicatively homomorphic if for any given encryption key and any plaintexts $m_1, m_2 \in \mathcal{P}$ the following condition is satisfied:* $\mathsf{Encrypt}(m_1 \cdot m_2) \equiv_{\mathcal{E}} \mathsf{Encrypt}(m_1) \cdot_{\mathcal{C}} \mathsf{Encrypt}(m_2)$

where $+_{\mathcal{C}}, \cdot_{\mathcal{C}} \in \mathcal{C}$ (i.e. can be computed directly, without any intermediate decryption), and $=_{\mathcal{E}}$ is the equivalence in regards to $\mathcal{E}$ (i.e. the resulting ciphertexts must represent the same plaintext).

For example, the RSA cryptosystem defined in Section 1.3 is multiplicatively homomorphic because $\mathsf{Encrypt}(e, m_1 \cdot m_2) = (m_1 \cdot m_2)^e \bmod N$ and $\mathsf{Encrypt}(e, m_1) \cdot \mathsf{Encrypt}(e, m_2) = (m_1^e \bmod N) \cdot (m_2^e \bmod N)$ are equivalent. However, it is easy to check that RSA is not additively homomorphic.

Notice that $(\mathcal{P}, +)$ or $(\mathcal{P}, \cdot)$ must be a group in order for a cryptosystem to be additively or multiplicatively homomorphic respectively. Otherwise the $\mathsf{Encrypt}$ algorithm will not be defined on some of the possible expressions (i.e. the result of some arithmetic operation will not belong to the plaintext space), and hence the equivalence required in the definition will not always hold. Similarly, $(\mathcal{C}, +_{\mathcal{C}})$ or $(\mathcal{C}, \cdot_{\mathcal{C}})$ must also be a group.

So the cryptosystem should essentially possess a homomorphism between the plaintext group $(\mathcal{P}, \odot)$ and the ciphertext group $(\mathcal{C}, \odot_{\mathcal{C}})$ (where $\odot$ denotes either addition or multiplication). By performing an operation on the ciphertext elements, we also implicitly change the corresponding plaintext elements. Then the encryption and the decryption algorithms map the elements between the plaintext group and the ciphertext group. A possible way to think about it, is that there should be a commutativity between the encryption function and the group operation.

Now consider the case when a cryptosystem is both additively and multiplicatively homomorphic, i.e. both the plaintext space and the ciphertext space form a ring and there is a homomorphism between then.

Definition 2.1.2 ((Informal) Algebraically Homomorphic Encryption)**.** *Consider a cryptosystem $\mathcal{E}$ which is defined over plaintext space $\mathcal{P}$ and ciphertext space $\mathcal{C}$. We say that $\mathcal{E}$ is algebraically homomorphic if $\mathcal{E}$ is both additively and multiplicatively homomorphic.*

Assume we have a cryptosystem which is algebraically homomorphic. This would mean we are able to add and multiply arbitrary ciphertexts. Now consider the least significant bit while performing these operations. If we ignore the other bits, we basically get the implementation of logic XOR and logic AND gates in the ciphertext space. The two logic gates together form a complete logic system, so we can implement an arbitrary logic circuit and hence compute an arbitrary function. Therefore, in order to achieve fully homomorphic encryption, we need to find cryptosystems which are algebraically homomorphic.

All currently known fully homomorphic encryption schemes use Boolean circuits to represent the functions we want to compute homomorphically. In this work we will also normally use Boolean circuits for this purpose.

2.2 Polly Cracker Cryptosystem

In this section we discuss an algebraically homomorphic encryption scheme Polly Cracker [18]. The symmetric version of Polly Cracker uses a secret key $\mathbf{sk} \in \mathbb{Z}_q^n$. Given a plaintext message $m \in \mathbb{Z}_q$, we choose a random polynomial $p \in \mathbb{Z}_q[x_1, ..., x_n]$ such that $p(\mathbf{sk}) = 0 \bmod q$. Then the corresponding ciphertext is a polynomial $c = p + m \pmod q$. In order to decrypt the ciphertext, we evaluate it at the secret key $\mathbf{sk}$ modulo q. The polynomial p vanishes and we get back the plaintext message, i.e. $c(\mathbf{sk}) = p(\mathbf{sk}) + m = m \pmod q$.

Consider the ciphertexts $c_1 = p_1 + m_1 \pmod q$ and $c_2 = p_2 + m_2 \pmod q$ We can use them to compute the following arithmetic operations:

- $c_1 + c_2 = (p_1 + p_2) + (m_1 + m_2) \mod q$
- $c_1 \cdot c_2 = (p_1 \cdot m_2 + p_2 \cdot m_1) + (m_1 \cdot m_2) \mod q$

This encryption scheme is obviously algebraically homomorphic because adding or multiplying the ciphertexts would also implicitly perform the same operation on the underlying plaintexts modulo q.

Recall that an *ideal* I of a ring R is a subset of R which is closed under addition, and also closed under multiplication with elements in R. Below we define the ideal membership problem which provides some elements of an ideal I and asks whether a specific other element in R also belongs to this ideal.

Definition 2.2.1 (Ideal Membership Problem)**.** *Given a polynomial $p_0 \in \mathbb{Z}[x_1, ..., x_n]$ from a ring R and a set of polynomials $p_1, p_2, ..., p_k \in \mathbb{Z}[x_1, ..., x_n]$ from an ideal $I \in R$, is it possible to express $p_0 = a_1p_1 + ... + a_kp_k$ where $a_1, ..., a_k \in \mathbb{Z}[x_1, ..., x_n]$.*

In order for Polly Cracker to be semantically secure, the adversary must not be able to distinguish whether two ciphertexts c_a and c_b encrypt the same plaintext m. Notice that this is equivalent to determining whether the ciphertext $c = c_a - c_b$ encrypts 0, i.e. whether the ciphertext c evaluates to zero at the secret key $\mathbf{sk}$. It can be seen that the semantic security is based on the *ideal membership problem* which is considered to be hard. In this case the ring R contains all polynomials, and the ideal I contains the polynomials which evaluate to zero at $\mathbf{sk}$.

It is also possible to transform Polly Cracker into a public-key cryptosystem. In order to give anyone a chance to encrypt their plaintext message, they should be able to generate a somewhat random element from the ideal. It is trivial to do that with the knowledge of the private key, however we can not reveal this key. Instead, we will generate a public key which will consist of many polynomials from the ideal, i.e. polynomials which evaluate to zero at the secret key. Then the encryption process would involve taking a random *subset sum* of these polynomials and adding them to the message.

This method of converting a symmetric encryption scheme into an asymmetric encryption scheme is formalized in [40]. We will also implicitly use it in the next chapters. Here we might need an additional requirement that an adversary can not determine what was the subset we chose during the encryption process. Recall that the subset sum is an NP-complete problem. We give its definition below, followed by the definition of the public-key version of Polly Cracker.

Definition 2.2.2 (Subset Sum Problem)**.** *Given a set of integers $x_1, ..., x_n$ and an integer s, is there a subset $S \in \{1, ..., n\}$ such that $\sum_{i \in S} x_i = s$.*

Definition 2.2.3 (Polly Cracker Cryptosystem [18])**.** *The public-key version of Polly Cracker relies on the hardness of the ideal membership and the subset sum problems.*

- KeyGen(*security parameter 1^λ*)*: outputs private key $(q \in \mathbb{Z}, \mathbf{sk} \in \mathbb{Z}_q^n)$ and public key $(q \in \mathbb{Z}, \{p_i\} \in \mathbb{Z}_q[x_1, ..., x_n])$ such that $p_i(\mathbf{sk}) = 0 \bmod q$ for all $i \in \{1, ..., k\}$. The values n, q, k depend on the security parameter λ.*
- Encrypt(*public key $(q, \{p_i\})$, plaintext $m \in \mathbb{Z}_q$*)*: outputs ciphertext $c = m + \sum_{i \in S} p_i \pmod q$, where $S \subseteq \{1, ..., k\}$ is a random subset.*
- Decrypt(*private key $(q, \mathbf{sk})$, ciphertext c*)*: outputs plaintext $m = c(\mathbf{sk}) \bmod q$.*

Unfortunately, Polly Cracker is typically easy to break using linear algebra [2]. Another problem is that every homomorphic multiplication of two ciphertexts can square the amount of monomials in the resulting ciphertext. Hence the size of the ciphertext can grow exponentially with the number of performed homomorphic multiplications.

2.3 Early Homomorphic Encryption Schemes

Some of the previous surveys of homomorphic encryption are available in [19, 28, 45] as well as [20, Chapter 3]. In this section we give a brief overview of the early homomorphic schemes which were not yet capable of computing *arbitrary* functions (i.e. somewhat homomorphic encryption schemes).

Here we are only interested in cryptosystems which are semantically secure. For instance, the textbook version of the RSA cryptosystem (defined in Section 1.3) is multiplicatively homomorphic, however it is not IND-CPA secure because its encryption algorithm is deterministic. RSA can be randomized using padding schemes, but then it loses its homomorphic properties.

The first semantically secure additively and multiplicatively homomorphic encryption schemes were Goldwasser-Micali [27] and ElGamal [17] cryptosystems respectively. Numerous other encryption schemes were subsequently proposed, featuring either additive or multiplicative homomorphism, many of them building up on Goldwasser-Micali cryptosystem. Most of these cryptosystems also support addition and/or multiplication by a known constant. For details on encryption schemes with a single homomorphism see surveys [19, 28].

Remark 2.3.1. *Notice that any additively homomorphic encryption scheme satisfies the following property:* $\mathsf{Encrypt}(m_1 + m_2) = \mathsf{Encrypt}(m_1) +_{\mathcal{C}} \mathsf{Encrypt}(m_2)$ *for any plaintext messages* $m_1, m_2 \in \mathcal{P}$*. Therefore, if* $+_{\mathcal{C}}$ *is a linear function in the ciphertext space, then it also follows that* $\mathsf{Encrypt}(k \cdot m_1) = \mathsf{Encrypt}(m_1) +_{\mathcal{C}} \ldots +_{\mathcal{C}} \mathsf{Encrypt}(m_1) = k \cdot \mathsf{Encrypt}(m_1)$ *for any constant* $k \in \mathbb{N}$*, i.e it supports multiplication by a known constant*[1]*.*

The secure somewhat homomorphic encryption schemes which offer more than a single homomorphism are as follows:

- Polly Cracker encryption scheme [18], described in Section 2.2.
- Boneh-Goh-Nissim cryptosystem [7] is an additively homomorphic encryption scheme which also allows to perform a single multiplication. More precisely, it allows to compute quadratic (i.e. 2-DNF) formulas over encrypted data.

[1] This observation actually follows from the definition of a linear function.

- Armknecht-Sadeghi cryptosystem [4] is an additively homomorphic encryption scheme which also allows to perform an arbitrary number of multiplications (fixed in advance). However, the ciphertext size grows exponentially with the number of multiplications.
- Melchor-Gaborit-Herranz cryptosystem [33] is able to homomorphically evaluate multi-variate polynomials, but the size of the ciphertext grows exponentially in the degree of the polynomial (i.e. in the number of consecutive multiplications).
- Sander-Young-Yung cryptosystem [41] allows to homomorphically evaluate arbitrary circuits, but the ciphertext size grows exponentially with the depth of the circuit. Hence we can only efficiently evaluate NC^1 circuits.
- Ishai-Paskin cryptosystem [29] allows to homomorphically evaluate branching programs, which is essentially equivalent to evaluating NC^1 circuits (according to [6]).

In 2009 Gentry was the first to show a *fully homomorphic encryption* scheme, i.e. a cryptosystem which allows to efficiently compute arbitrary functions over encrypted data [20, 21].

2.4 Security of the Homomorphic Encryption Schemes

First of all, notice that any homomorphic encryption scheme is always malleable by design. Given any encryption of a known constant r and an arbitrary ciphertext c, we can either add or multiply them together to get another valid ciphertext. The malleability in turn implies that a homomorphic encryption scheme can not be IND-CCA2 secure (for a formal proof refer to [48]).

Some of the known cryptosystems with a single homomorphism are IND-CCA1 secure, and there is also one more complex construction of somewhat homomorphic encryption scheme which is proven to be IND-CCA1 secure [31]. However, the existence of IND-CCA1 secure *fully* homomorphic encryption scheme is currently an open problem [20, Page 18].

Additionally, it is proven that any deterministic additively homomorphic encryption scheme is not IND-CCA1 secure if its addition operator in ciphertext space is a linear function [1, Section 4.2] (also [48, 28]). Assume $[c_1, ..., c_\ell]$ is a binary representation of a ciphertext message. In order to mount the attack, it is required to find the plaintexts corresponding to the following ciphertexts:

$$\mathsf{Encrypt}(m_1) = [1, 0, ..., 0]$$

$$\mathsf{Encrypt}(m_2) = [0, 1, ..., 0]$$
$$...$$
$$\mathsf{Encrypt}(m_\ell) = [0, 0, ..., 1]$$

Then any ciphertext can be decrypted by computing $m = \sum_{i=1}^{\ell} m_i c_i$.

Other known attacks against the homomorphic encryption schemes are as follows:

- If a homomorphic encryption scheme allows to encrypt known constants and supports an operation $\leq$ inside the ciphertext space, then we can use binary search to find the plaintext values corresponding to any ciphertexts [39, Page 4].
- Any algebraically homomorphic encryption scheme which offers an equality testing oracle (e.g. if the scheme is deterministic) over finite commutative ring (e.g. $\mathbb{Z}_n$) can be broken in sub-exponential time [8, Section 4.1].
- Given an ability to decide whether a ciphertext is an encryption of zero, it is possible to break any algebraically homomorphic encryption scheme in polynomial time using quantum computation [46, Section 4.1].

As mentioned in Section 2.1, we use Boolean circuits to represent the functions we want to compute homomorphically. We end this section with two more related comments from [22, Section 2.2].

First of all, Boolean circuits are sometimes less efficient than random access machines. For instance, binary search algorithm will work only in linear time on circuits. However, it is worth to note that a homomorphic computation of any function f should always access each of the available ciphertexts, and hence cannot work faster than in linear time. If the computation ignored any ciphertexts, it would therefore leak some information about the irrelevance of particular ciphertexts with respect to the computable function.

Similarly, if the amount of data returned by a function is not hardwired beforehand (e.g. searching in the files), it should be impossible to determine the amount of output data during the homomorphic computation. Otherwise it means that the system knows some information about the relations between the function and the encrypted data beforehand, which contradicts the semantic security. Therefore, it is necessary to explicitly indicate the amount of data we want to receive as a result of computing any function homomorphically. The output of a secure system will then be padded or truncated to meet the required size.

Chapter 3

Fully Homomorphic Encryption

In this chapter we give a formal definition of a fully homomorphic encryption scheme and describe Gentry's blueprint [21, 20] of constructing such schemes. The most comprehensive work on the general theory of homomorphic encryption was done in the original work by Gentry. However, sufficient theoretical background can also be found in a number of later publications, e.g. [45, 22, 47, 9, 11]. We base this chapter on all of these sources, using the definitions from several of them.

3.1 Definition of Fully Homomorphic Encryption

In order to define homomorphic encryption, we extend the definition of public-key encryption given in Section 1.3. We add a public evaluation key *evk* and an algorithm Evaluate. This algorithm uses the evaluation key to compute a function over encrypted data, and returns the result of this function encrypted under the same public key as the initial ciphertexts.

Definition 3.1.1 (Homomorphic Encryption Scheme)**.** *A homomorphic encryption scheme is a* 4*-tuple of probabilistic polynomial-time algorithms* HE = (KeyGen, Encrypt, Decrypt, Evaluate) *satisfying the conditions below:*

- KeyGen *takes a unary representation of a security parameter* 1^λ *as input and outputs a randomized triple of keys* (pk, evk, sk) *where* pk *is a public key,* evk *is an evaluation key and* sk *is a private key.*
- Encrypt *takes a public key* pk *and a plaintext* $m \in \{0,1\}$ *as input and outputs a ciphertext* c*.*
- Decrypt *takes a private key* sk *and a ciphertext* c *as input and outputs the corresponding plaintext* $m \in \{0,1\}$*.*

- Evaluate *takes an evaluation key evk, a function* $f : \{0,1\}^\ell \to \{0,1\}$ *and an* ℓ*-tuple of ciphertexts* $\langle c_1, ..., c_\ell \rangle$ *as input and outputs a ciphertext* c_f*.*

We define the plaintext space as $\{0,1\}$ because all of the known fully homomorphic encryption schemes encrypt bit-by-bit. However, the definition can be trivially generalized to work with an arbitrary plaintext space.

Most authors do not distinguish between the public key and the evaluation key. However, we follow suit of the recent works from [45, 9, 11] and define the public key and the evaluation key as separate keys. This is done in order to clearly indicate the different purpose of these keys (we only use the public key for the Encrypt algorithm and the evaluation for the Evaluate algorithm).

As mentioned in Section 2.1, the functions passed to the Evaluate algorithm are represented by arithmetic circuits over $GF(2)$, i.e. Boolean circuits with addition and multiplication modulo 2. Similar to the other algorithms, the Evaluate algorithm must run in time polynomial in λ and also in time polynomial in the size of the Boolean circuit which corresponds to the processed function f.

We will normally use algorithms Add and Mult in order to refer to the logic XOR and AND gates used in the circuits. Both algorithms take an evaluation key and two ciphertext bits (which encrypt the arguments of addition or multiplication) as input and output one ciphertext bit (which encrypts the result), i.e. $c_{add} \leftarrow \mathsf{Add}_{evk}(c_1, c_2)$ and $c_{mul} \leftarrow \mathsf{Mul}_{evk}(c_1, c_2)$.

The semantic security of a homomorphic encryption scheme is defined the same way as usual. The main difference from a public-key encryption scheme is the Evaluate algorithm, however this algorithm uses no secrets.

Notice that the presented definition of a homomorphic encryption scheme does not restrict the behavior of the Evaluate algorithm. This is done on purpose so we can further define different types of homomorphic encryption schemes with respect to the circuit classes which should be correctly processed. If an encryption scheme possess such properties, we call it somewhat homomorphic with respect to the according set of permitted functions $\mathcal{F}$. We then say that HE can *handle* functions in $\mathcal{F}$.

A natural way to classify the circuits is by looking at their depth.

Definition 3.1.2 (L-homomorphism [9, Definition 2.3])**.** *A scheme* HE *is* L*-homomorphic, for* $L = L(n)$*, if for any depth* L *arithmetic circuit* f *(over* $GF(2)$*) and any set of inputs* $m_1, ..., m_\ell$*, it holds that*

$$Pr[\mathsf{HE.Decrypt}_{sk}(\mathsf{HE.Evaluate}_{evk}(f, c_1, ..., c_\ell)) \neq f(m_1, ..., m_\ell)] = negl(\lambda),$$

where $(pk, evk, sk) \leftarrow \mathsf{HE.KeyGen}(1^\lambda)$ *and* $c_i \leftarrow \mathsf{HE.Encrypt}_{pk}(m_i)$*.*

More generally, we can state a similar definition for an arbitrary class of functions.

Definition 3.1.3 ($\mathcal{F}$-homomorphism, adapted from [45, Definition 3.2])**.** *Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ be a class of functions (together with their respective representations). A scheme* HE *is $\mathcal{F}$-homomorphic (or, homomorphic for the class $\mathcal{F}$) if for any function $f_\lambda \in \mathcal{F}_\lambda$ with $f_\lambda : \{0,1\}^{\ell(\lambda)} \to \{0,1\}$, and respective inputs $m_1, ..., m_\ell \in \{0,1\}$, it holds that*

$$Pr[\mathsf{HE.Decrypt}_{sk}(\mathsf{HE.Evaluate}_{evk}(f, c_1, ..., c_\ell)) \neq f(m_1, ..., m_\ell)] = negl(\lambda),$$

where $(pk, evk, sk) \leftarrow \mathsf{HE.KeyGen}(1^\lambda)$ and $c_i \leftarrow \mathsf{HE.Encrypt}_{pk}(m_i)$.

Both of the definitions above guarantee the correctness of the Evaluate algorithm for a limited set of circuits. In contrast, we do not require the traditional correctness of decryption. It is possible that a ciphertext c_i can only be decrypted after applying the Evaluate algorithm to it. Note that it might still be possible to run the Evaluate algorithm in order to compute an identity function, if such function belongs to the set of the allowed circuits; and afterwards launch the decryption algorithm.

Additionally, we do not require that the output of the Evaluate algorithm can be directly used in another processing of Evaluate algorithm. However, it is possible in most of the currently known fully homomorphic encryption schemes. We call this property "multi-hop homomorphism" or "k-hop homomorphism" where k is the possible amount of additional runs (e.g. "1-hop homomorphism" allows to run the Evaluate algorithm on its own output *once*).

The definitions above do not exclude some trivial implementations of Evaluate algorithm. Suppose the algorithm Evaluate needs to compute a circuit C on ciphertexts $c_1, ..., c_\ell$. One of the possible tricks is to simply output the description of $(C, c_1, ..., c_\ell)$ without evaluating the circuit. Instead, delegate the processing to the Decrypt algorithm, which could evaluate C on the plaintext bits after decrypting the corresponding ciphertexts. This kind of solution obviously does not satisfy us. In order to avoid it, we define a *compactness* property which requires that the size of the decryption circuit is independent of the function we want to evaluate.

Definition 3.1.4 (Compact Homomorphic Encryption [20, Definition 2.1.2])**.** *We say that a homomorphic encryption scheme* HE *is compact if there is a polynomial g such that, for every value of the security parameter λ,* HE*'s decryption algorithm can be expressed as a circuit $D_\mathcal{E}$ of size at most $g(\lambda)$.*

There also exist other ways to define compactness. For instance, in [11, Definition 3.4] the number of the output bits of the HE.Evaluate algorithm is restricted polynomially in the security parameter λ, independent of the processed circuit C. However,

our definition also implies this restriction on the size of the ciphertext. If the size of the decryption circuit is limited, it obviously limits (puts an upper bound on) the size of the ciphertexts it can process. The compactness property implicitly requires our encryption scheme to be effective. If our ciphertexts would grow exponentially in the number of multiplications we perform, then at some point the decryption algorithm would fail to recover the plaintext.

Finally, we are able to define the desired functionality of computing *arbitrary* functions over encrypted data.

Definition 3.1.5 (Fully Homomorphic Encryption [11, Definition 3.3])**.** *A scheme* HE *is fully homomorphic if it is both compact and homomorphic for the class of all arithmetic circuits over* $GF(2)$*.*

We will also use a relaxation of fully homomorphic encryption as provided below.

Definition 3.1.6 (Leveled Fully Homomorphic Encryption [11, Definition 3.6])**.** *A leveled fully homomorphic encryption scheme is a homomorphic scheme where the* $\mathsf{HE.KeyGen}$ *gets an additional input* 1^L *(now* $(pk, evk, sk) \leftarrow \mathsf{HE.KeyGen}(1^\lambda, 1^L)$*) and the resulting scheme is homomorphic for all depth-*L *binary arithmetic circuits. The length of the ciphertext should remain independent of* L*.*

We might also want that the output of the $\mathsf{Evaluate}$ algorithm reveals no information about the processed circuit C. That would mean that the ciphertexts returned by $\mathsf{Decrypt}$ and $\mathsf{Evaluate}$ algorithms have the same statistical distribution. This property is called *circuit privacy.*

Definition 3.1.7 ((Statistical) Circuit Private Homomorphic Encryption [20, Definition 2.1.6])**.** *We say that a homomorphic encryption scheme* HE *is circuit-private for circuits in* $\mathcal{F}_{\mathsf{HE}}$ *if, for any key-triple* (sk, evk, pk) *output by* $\mathsf{KeyGen}(1^\lambda)$*, any circuit* $C \in \mathcal{F}_{\mathsf{HE}}$*, and any fixed ciphertexts* $\Psi = \langle c_1, ..., c_\ell \rangle$ *that are in the image of* $\mathsf{Encrypt}_{\mathsf{HE}}$ *for plaintexts* $m_1, ..., m_\ell$*, the following distributions (over the random choices in* $\mathsf{Encrypt}_{\mathsf{HE}}$*,* $\mathsf{Evaluate}_{\mathsf{HE}}$*) are (statistically) indistinguishable:*

$$\mathsf{Encrypt}_{\mathsf{HE}}(pk, C(m_1, ..., m_\ell)) \approx \mathsf{Evaluate}_{\mathsf{HE}}(pk, C, \Psi)$$

The obvious correctness condition of $\mathsf{Evaluate}$ *algorithm must still hold.*

One of the ways to ensure circuit-privacy is adding a random encryption of 0 to every output of $\mathsf{Encrypt}$ and $\mathsf{Evaluate}$. However, it depends on the construction details of a particular encryption scheme. This approach will only work if a small number of randomizations (i.e. adding zero several times) can result in a statistically random

encryption [20, Page 30]. Basically, if a ciphertext has a complex structure, we might get the random result only within a limited set of ciphertexts, or with a distribution which differs from the desired distribution.

3.2 Noisy Encryption Schemes

In this section we consider cryptosystems whose ciphertexts contain some amount of *random noise*. Normally, the noise is a distance to the nearest multiple of some unknown value or vector, and the plaintext message forms a small portion of this noise. The encryption algorithm of such cryptosystem produces ciphertexts which may contain up to B amount of noise (we call such ciphertexts *fresh*). And the maximum permitted amount of noise is N, such that $B \ll N$. When performing any homomorphic operations on the ciphertexts, we get the result which has more noise than the original ciphertexts. Finally, if a ciphertext message contains more than N amount of noise, it can no longer be decrypted correctly.

Our concept of noise is basically analogous to the nearest neighbour decoding concept. One can imagine that the decryption algorithm chooses the closest plaintext in the proximity of the ciphertext. If the noise becomes too big, the ciphertext moves too far from the original plaintext and the decryption algorithm fails.

As an example of a noisy encryption scheme we consider the Noisy Polly Cracker cryptosystem. Noisy Polly Cracker is an extension of Polly Cracker cryptosystem (from Section 2.2) which adds noise to the ciphertexts in order to improve its security against the attacks using linear algebra.

We add a small and even noise $2 \cdot e_i + m_i$ to each of the polynomials we use. In order to recover the message from the noise, we take the remainder modulo 2 at the end of the decryption algorithm. Therefore, our plaintext space is reduced to $\{0, 1\}$.

Definition 3.2.1 (Noisy Polly Cracker Cryptosystem [2])**.**

- KeyGen(*security parameter* 1^λ)*:*

  outputs private key $(q \in \mathbb{Z}, \mathbf{sk} \in \mathbb{Z}_q^n)$ *and public key* $(q \in \mathbb{Z}, \{p_i\} \in \mathbb{Z}_q[x_1, ..., x_n])$ *such that* $p_i(\mathbf{sk}) = 2 \cdot e_i \bmod q$ *for all* $i \in \{1, ..., k\}$ *where* $e_i \in \mathbb{Z}$, $e_i \ll q$. *The values* n, q, k *depend on the security parameter* λ.

- Encrypt(*public key* $(q, \{p_i\})$, *plaintext* $m \in \{0, 1\}$)*:*

  outputs ciphertext $c = m + \sum_{i \in S} p_i \bmod q$, *where* $S \subseteq \{1, ..., \ell\}$ *is a random subset.*

- Decrypt(*private key* $(q, \mathbf{sk})$, *ciphertext* c)*:*

outputs plaintext $m = (c(\mathbf{sk}) \bmod q) \bmod 2 = ((\sum_i qp_i + 2e_i + m) \bmod q) \bmod 2 = (\sum_i 2e_i + m) \bmod 2 = m \bmod 2$.

Notice that the Noisy Polly Cracker cryptosystem still supports homomorphic addition and homomorphic multiplication. Consider the ciphertexts c_1, c_2 which encrypt some plaintexts m_1, m_2, i.e. $c_1 = p_1 + (2e_1 + m_1)$ and $c_2 = p_2 + (2e_2 + m_2)$. Then:

- $c_1 + c_2 = (p_1 + p_2) + (2e_1 + 2e_2 + m_1 + m_2)$.
- $c_1 \cdot c_2 = (p_1 p_2 + m_1 p_2 + 2e_1 p_2 + m_2 p_1 + 2e_2 p_1) + (2e_1 + m_1) \cdot (2e_2 + m_2)$.

However, these operations will also increase the noise of the resulting ciphertext. Adding two ciphertexts adds the noises, and multiplying two ciphertexts will multiply the noises.

Therefore, the Noisy Polly Cracker cryptosystem can homomorphically evaluate Boolean circuits of limited complexity. Now we try to give a rough estimate of the depth of such circuits. Notice that the evaluation of every consecutive gate can square the noise in the worst case. If the initial noise is at most B, then after the first gate it can increase up to B^2, after the second consecutive gate up to B^4, and so on. If the permitted noise threshold is N, then we can keep performing arithmetic operations on the ciphertext until the power of the initial noise B reaches the value of $\log_B N$. This in turn allows us to safely evaluate circuits whose depth is roughly $\log_2 \log_n N = \log_2 \left(\frac{\log_2 N}{\log_2 n} \right) = \log_2 \log_2 N - \log_2 \log_2 n$.

All of the currently known fully homomorphic encryption schemes are based on this idea of having ciphertexts with noise which accumulates over time. Their constructions use different mathematical primitives and underlying hardness assumptions, but their design is very similar.

Notice that any noisy cryptosystem is somewhat homomorphic for the circuits of particular depth. The somewhat homomorphic encryption schemes can already be practical for evaluating the functions which can be expressed as low-degree polynomials (for examples see [34]). More importantly, in the next section we show how to transform a somewhat homomorphic encryption scheme into a fully homomorphic encryption scheme.

3.3 Bootstrappable Encryption Schemes

In the previous section we introduced a concept of noisy encryption schemes. Using these encryption schemes we can compute circuits of limited depth, because after certain

amount of homomorphic evaluations the noise in the ciphertexts grows too big. The next logical step is to find some way to reduce the ciphertext noise once in a while, so that we can keep on evaluating circuits indefinitely. We will use the solution proposed by Gentry in [21, 20].

In order to manage the noise level, we introduce a new algorithm $\mathsf{Recrypt}$. Every time the noise in a ciphertext becomes too big, this algorithm *refreshes* the ciphertext by constructing a fresh ciphertext which encrypts the same plaintext value (possibly under different key) and contains only a small amount of random noise. Then the new ciphertext can be used in subsequent homomorphic operations. Notice that if the threshold of the allowed noise is N, then we want $\mathsf{Recrypt}$ to return a ciphertext with at most $\sqrt{N}$ noise so that we are able to perform at least one multiplication.

The only way we know how to reduce the noise of the ciphertext is decrypting it. The decryption algorithm removes any noise associated to the ciphertext. We could then encrypt the resulting plaintext again to get a fresh ciphertext with a low noise. However, we obviously can not reveal our secret decryption key to anyone. Therefore, we will do the described procedure in the reverse order. First, we encrypt our noisy ciphertext, thus getting a doubly encrypted plaintext using two keys simultaneously. And then we decrypt the inner encryption layer, hence removing any noise which was associated with it.

In order to access the inner encryption layer and decrypt it, we will use the homomorphic properties of the encryption scheme. We will compute the circuit of the $\mathsf{Decrypt}$ algorithm homomorphically, using a previously encrypted secret key. Notice that computing this circuit will also add a new noise to the outer encryption layer. However, we still made a progress as long as the old ciphertext contained more noise than we get after evaluating the decryption circuit on the fresh ciphertext.

Below we give a description of the $\mathsf{Recrypt}$ algorithm which takes a noisy ciphertext c_0 which is encrypted under public key pk_0 and outputs a ciphertext c_1 which contains less noise and is encrypted under another public key pk_1. The algorithm also takes public and evaluation keys pk_1, evk_1, the decryption circuit $\mathsf{Decrypt}_{\mathsf{HE}}$ and an encryption of the secret key sk_0 under public key pk_1, denoted as $\overline{sk_0}$ and computed $\overline{sk_0} = \mathsf{Encrypt}(pk_1, sk_0)$. Normally the encryption of the secret keys (e.g. $\overline{sk_0}$) will be considered to be a part of the evaluation key, however we indicate $\overline{sk_0}$ separately in this case.

$\mathsf{Recrypt}(c_0, \overline{sk_0}, pk_1, evk_1, \mathsf{Decrypt}_{\mathsf{HE}})$:

- Compute $\overline{c_0} \leftarrow \mathsf{Encrypt}(pk_1, c_0)$.
- Return $c_1 \leftarrow \mathsf{Evaluate}(evk_1, \mathsf{Decrypt}_{\mathsf{HE}}, \langle \overline{sk_0}, \overline{c_0} \rangle)$.

Notice that it is safe to encrypt sk_0 under pk_1, because the semantic security implies sk_0 can not be distinguished from any other encryption under the key pk_1. However, this also immediately means that this construction can not be CCA1-secure, since an adversary would then query the encrypted secret keys to the decryption oracle.

Notice that in order to process the Recrypt algorithm, the encryption scheme must be able to process its own decryption algorithm homomorphically. In order to process a more complicated decryption circuit we need to expand the set of circuits the scheme can handle homomorphically. And if we increase the class of such circuits, the decryption circuit becomes more complex at the same time. The requirement towards the complexity of the decryption circuit is self-referential. Hence we simply want to choose the encryption schemes whose decryption circuits are as simple as possible (e.g. with a circuit depth at most $\log\log N - \log\log n$)[1].

While performing a homomorphic computation of a circuit, apart from reducing the noise level, we also want to make progress at processing the circuit. We would like to homomorphically compute at least one arithmetic operation (either Add or Mult) per every run of Recrypt. If this is possible, we call such scheme *bootstrappable*.

Definition 3.3.1 (Bootstrappable Encryption Scheme [11, Definition 3.7])**.** *Let* HE *be* $\mathcal{F}$*-homomorphic, and let* f_{add} *and* f_{mult} *be the augmented decryption functions of the scheme defined as*

$$f_{add}^{c_1,c_2}(s) = \mathsf{Add}_{evk}(\mathsf{HE.Dec}_{sk}(c_1),\ \mathsf{HE.Dec}_{sk}(c_2))$$

and

$$f_{mult}^{c_1,c_2}(s) = \mathsf{Mult}_{evk}(\mathsf{HE.Dec}_{sk}(c_1),\ \mathsf{HE.Dec}_{sk}(c_2)).$$

Then HE *is bootstrappable if*

$$\{f_{add}^{c_1,c_2}, f_{mult}^{c_1,c_2}\}_{c_1,c_2} \subseteq \mathcal{F}.$$

Namely, the scheme can homomorphically evaluate f_{add} *and* f_{mult}*.*

If an encryption scheme is bootstrappable, we are able to evaluate circuits of arbitrary depth, but the size of the public key increases accordingly. Specifically, we can topologically arrange the logic circuit C into L levels of gates. We will then generate L sets of keys and use them to subsequently apply the Recrypt algorithm after processing each level of gates.

All of the gates at the 0-th level are encrypted under the public key pk_0. After all of these gates are processed, we use the Recrypt algorithm to refresh all of the intermediate

[1] In order to meet the circuit depth requirements, the initial blueprint of Gentry also described a *decryption circuit squashing* method. We describe it separately in section 4.5.

ciphertexts received from output wires of 0th level gates, hence re-encrypting them under different public key pk_1[2]. We continue to do the same with every next level of gates.

The described approach basically transforms an arbitrary bootstrappable scheme into a leveled fully homomorphic scheme. This observation is reflected in the theorem below.

Theorem 3.3.2 (Adapted from [20, Theorems 4.2.1, 4.2.3] and [44, Theorem 2.2.]). *Given a bootstrappable scheme* HE *and a parameter* $d = poly(\lambda)$*, it is possible to construct a leveled fully homomorphic encryption scheme* $\mathsf{HE}^{(d)}$ *which is compact and correct (in respect to* $\mathsf{Evaluate}$*) for all circuits of depth* d*.*

Furthermore, if the scheme HE *is semantically secure, then the scheme* $\mathsf{HE}^{(d)}$ *is also semantically secure.*

We now define the *weak circular security* (alternatively, *key-dependent message* (KDM) *security*) property which describes the cryptosystems which are secure when an adversary has access to key-dependent messages, including the messages which encrypt the decryption key itself.

Definition 3.3.3 (Weak Circular Security [11, Definition 3.8]). *A public key encryption scheme* $(\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *is weakly circular secure if it is IND-CPA secure even for an adversary with auxiliary information containing encryption of all secret key bits:* $\{\mathsf{Encrypt}_{pk}(sk[i])\}_i$.

Finally, as a straightforward consequence, if a bootstrappable scheme is circular secure, then we can construct a fully homomorphic encryption scheme. This is done in the same way as constructing leveled fully homomorphic encryption scheme, but always using $\mathsf{Recrypt}$ to refresh the ciphertexts under the same key instead of having to pick a different key every time.

Theorem 3.3.4 (Adapted from [20, Theorems 4.3.2]). *Given a KDM-secure bootstrappable scheme* HE*, it is possible to construct a semantically secure fully homomorphic scheme* FH.

We should explicitly note that the length of the ciphertexts does not change in the bootstrapping process.

Lemma 3.3.5 ([11, Lemma 3.3]). *If a scheme* FH *is obtained from applying either Theorem 3.3.2 or Theorem 3.3.4 to a bootstrappable scheme* HE*, then both* $\mathsf{FH.Encrypt}$

[2]We assume that every level of gates contains at least one multiplication gate and the noise grows too big, otherwise we might be able to process several levels with the same key.

and FH.Evaluate *produce ciphertexts of the same length as* HE.Evaluate *(regardless of the length of the ciphertext produced by* HE.Evaluate*).*

Chapter 4

Fully Homomorphic Encryption over the Integers

In this chapter we discuss the conceptually simplest construction of fully homomorphic encryption scheme from [47] (informal description also available in [22]). This scheme works with integers and its security is based on the approximate greatest common divisor ("approximate GCD") problem. Given a list of near-multiples of some hidden integer, this problem asks to recover the hidden integer.

4.1 Encryption Scheme

We start this section by defining the public-key encryption scheme by van Dijk, Gentry, Halevi and Vaikuntanathan [47]. We then discuss the choice of parameters and the security of this scheme.

Remark 4.1.1 (Notation)**.** *In this and the next chapters we denote the reduction of* $z \in \mathbb{R}$ *modulo* $p \in \mathbb{Z}$ *by* $[z]_p$ *(along with the usual notation) with the result in the interval* $(-p/2, p/2]$*. We write* $\lfloor z \rceil$ *to denote a real value* z *rounded up to the nearest integer. We use* $f(n) \in \tilde{\mathcal{O}}(g(n))$ *to denote* $f(n) \in \mathcal{O}(g(n) \log^k g(n))$ *for some* $k \in \mathbb{N}$*. The* l_1 *norm of a vector* $\mathbf{v}$ *is denoted by* $|\mathbf{v}|_1$ *and equals* $\sum_i |v_i|$*.*

Given an odd integer p as a private key, we encrypt a message m by computing $c = pq + 2r + m$ where q is a large random integer and $r \ll p$ is a small random noise component (also integer). In order to decrypt, we compute $m = (c \bmod p) \bmod 2$.

The public key will contain random near-multiples of the private key (not definitely containing an even amount of noise). We will use a subset sum of these numbers for the encryption, multiplying them by two and hence getting random encryptions of zero.

It is easy to check that this encryption scheme is algebraically homomorphic, although currently it is only somewhat homomorphic due to the noise which will accumulate over time.

In this scheme we introduce several parameters, all of which depend on the security parameter λ. Below we give description and the advised size for these parameters:

- ρ – bit-length of the noise (used for the generation of the public key). Set $\rho = \omega(\log \lambda)$ to avoid brute force attacks on the noise.
- η – bit-length of the secret key. Set $\eta \geq \rho \cdot \Theta(\lambda \log^2 \lambda)$ to allow the evaluation of the circuits deep enough to achieve fully homomorphic encryption (see Section 4.4).
- γ – bit-length of the integers in the public key. Set $\gamma = \omega(\eta^2 \log \lambda)$ to avoid lattice attacks on the approximate GCD problem.
- τ – number of integers in the public key. Set $\tau \geq \gamma + \omega(\log \lambda)$ to allow the security reduction to the approximate GCD problem.
- ρ' – bit-length of the secondary noise (used by the encryption algorithm). Set $\rho' = \rho + \omega(\log \lambda)$ to allow the security reduction to the approximate GCD problem (see Section 4.3).

Definition 4.1.2 (vDGHV Encryption Scheme, adapted from [47, Section 3.1])**.** *The vDGHV public-key encryption scheme relies on the hardness of the approximate GCD problem.*

- KeyGen(*security parameter* 1^λ) *outputs:*
  - *An empty evaluation key evk (it is not required by the* Evaluate *algorithm).*
  - *An odd*[1] *η-bit private key integer* $p \xleftarrow{\$} (2\mathbb{Z} + 1) \cap [2^{\eta-1}, 2^\eta)$ *(i.e. chosen uniformly at random).*
  - *A public key* $\langle x_0, x_1, ..., x_\tau \rangle$ *consisting of* $\tau + 1$ *random* γ*-bit integers. We sample the values of* x_i *uniformly at random from the distribution* $\mathcal{D}_{\gamma,\rho}(p) = \{$*choose* $q \xleftarrow{\$} \mathbb{Z} \cap [0, 2^\gamma/p), r \xleftarrow{\$} \mathbb{Z} \cap (-2^\rho, 2^\rho)$ *: output* $x = pq + r\}$*. We relabel the contents of the public key so that* x_0 *is the largest integer, and we restart the* KeyGen *algorithm unless* x_0 *is odd and* $x_0 \bmod p$ *is even*[2]*.*

[1] The private key p has to be odd in order for the security reduction to work (see Section 4.3).

[2] We use x_0 as a modulus in the encryption algorithm, hence it needs to be the largest integer and also needs to encrypt zero.

- Encrypt(*public key* $\langle x_0, x_1, ..., x_\tau \rangle$, *plaintext* $m \in \{0, 1\}$):

  outputs $c = \left[m + 2r + 2 \sum_{i \in S} x_i \right]_{x_0}$ *where the integer noise* $r \in (-2^{\rho'}, 2^{\rho'})$ *and the subset* $S \subseteq \{1, 2, ..., \tau\}$ *are chosen at random.*

- Evaluate(*evaluation key evk, fanin-*ℓ *circuit* C, ℓ*-tuple of ciphertext bits* $\langle c_1, ..., c_\ell \rangle$):

  outputs the result of evaluating C *at the input values* $\langle c_1, c_2, ..., c_\ell \rangle$*, where integer addition and multiplication is used in order to evaluate the gates* XOR *and* AND *respectively.*

- Decrypt(*private key* p, *ciphertext* $c \in \mathbb{Z}$):

  outputs the plaintext $m = (c \bmod p) \bmod 2$.

It is suggested in [47, Page 6] to use the following convenient parameter set which meets the requirements towards the size of the parameters: $\rho = \lambda, \rho' = 2\lambda, \eta = \tilde{\mathcal{O}}(\lambda^2), \gamma = \tilde{\mathcal{O}}(\lambda^5)$ and $\tau = \gamma + \lambda$.

4.2 Efficiency of the Scheme

One of the problems of the currently described scheme is the growth of the ciphertext as a result of evaluating the circuits. Remember that we use x_0 for the modular reduction in the Encrypt algorithm. However, a result of a single multiplication of two γ-bit ciphertexts can contain up to γ^2 bits. If we reduce the γ^2-bit long result modulo the γ-bit long number x_0 (which can contain around ρ bits of noise), we then introduce a new noise of size up to $(\gamma^2/\gamma) \cdot \rho$ bits into the result, which is unacceptable. Surprisingly, one of the possible solutions is to generate x_0 as an exact multiple of p. This allows to perform modular reduction after every operation performed during the Evaluate algorithm. There are no attacks known which could considerably benefit from x_0 being the exact multiple (see Section 4.3).

Notice that the size of the ciphertext is roughly $\gamma = \tilde{\mathcal{O}}(\lambda^5)$ bits. Similarly, the public key contains τ integers, each of them γ bits long, so the size of the public key is approximately $\tau \cdot \gamma = \tilde{\mathcal{O}}(\lambda^{10})$ bits. According to [15, Page 4], the value of γ should be at least 2^{23} to prevent lattice attacks. This means the ciphertexts are at least 2^{23} bits long (for each bit of plaintext), and the public key is at least 2^{46} bits long which is highly impractical.

The efficiency of the scheme was improved in subsequent works. In [15] it is suggested to encrypt using a quadratic form in the public key elements which helps to reduce the value of τ down to roughly $2\sqrt{\tau}$. Together with other improvements the

suggested value of τ is $\tilde{\mathcal{O}}(\lambda^2)$ and hence the size of the public key becomes $\tilde{\mathcal{O}}(\lambda^7)$. The follow-up work [16] further reduces the size of the private key down to $\tilde{\mathcal{O}}(\lambda^5)$. In order to generate the public key elements, it uses a pseudo-random number generator with a public random seed. The generated information is used for $\gamma - \eta$ most significant bits of the public key elements, while the remaining η bits are being set using the stored set of corrections which was generated by KeyGen algorithm with respect to the secret key p. This work also showed how to apply the modulus-switching technique to the integer scheme (for the overview of modulus-switching see Section 5.3).

In Section 6.2 we will discuss the attempts to implement this encryption scheme using the optimizations above.

4.3 Security of the Scheme

As mentioned above, the security of the vDGHV encryption scheme relies on the approximate GCD problem. We give the definition of this problem below.

Definition 4.3.1 (Approximate GCD [47, Definition 4.1])**.** *The (ρ, η, γ)-approximate-gcd problem is: given polynomially many samples from $\mathcal{D}_{\gamma,\rho}(p)$ for a randomly chosen η-bit odd integer p, output p.*

The security reduction of this encryption scheme fixes a randomly-chosen public key and shows that if an adversary can decrypt a random ciphertext under this public key, then it can also recover the private key p and hence solve the approximate GCD problem.

Theorem 4.3.2 ([47, Theorem 4.2])**.** *Fix the parameters $(\rho, \rho', \eta, \gamma, \tau)$ as in the Somewhat Homomorphic Scheme from Definition 4.1.2 (all polynomial in the security parameter λ).*

Any attack $\mathcal{A}$ with advantage ε on the encryption scheme can be converted into an algorithm $\mathcal{B}$ for solving (ρ, η, γ)-approximate-gcd with success probability at least $\varepsilon/2$. The running time of $\mathcal{B}$ is polynomial in the running time of $\mathcal{A}$, and in λ and $1/\varepsilon$.

Below we outline the proof of Theorem 4.3.2 given in [47]:

- Assume we are given an adversary $\mathcal{A}$ which for a pair of public key and ciphertext outputs the correct plaintext with probability $\frac{1}{2} + \varepsilon$. Then we are able to build a subroutine which finds the least significant bit of number q (we denote it by $\mathsf{LSB}(q)$) for any ciphertext $z = pq + 2r + m$ with overwhelming probability.

  Specifically, given a ciphertext c which encrypts a known plaintext $\mu \in \{0, 1\}$, we can compute $\mathsf{LSB}(q) = \mathsf{Decrypt}(z + c) \oplus \mathsf{parity}(z) \oplus \mu$, where $\oplus$ is addition modulo 2.

This observation depends on the fact that the private key p is an odd integer. If $m = 0$ then

$$\mathsf{LSB}(q) = \mu \oplus \mathsf{parity}(z) \oplus \mu = \mathsf{parity}(z) = \mathsf{parity}(pq + 2r),$$

otherwise

$$\mathsf{LSB}(q) = (\neg\mu) \oplus \mathsf{parity}(z) \oplus \mu = \neg\mathsf{parity}(z) = \mathsf{parity}(pq + 2r),$$

where $\mathsf{parity}(pq + 2r) = \mathsf{parity}(q)$ because p is odd and $2r$ is even.

We use the attack $\mathcal{A}$ as an implementation of $\mathsf{Decrypt}$ algorithm. We compute $\mathsf{LSB}(q)$ many times and choose the majority amongst its outputs (i.e. amplify the advantage ε).

- Recall the binary GCD algorithm:
  - $\gcd(0, s) = gcd(s, 0) = s$
  - $\gcd(2a, 2b) = 2 \cdot \gcd(a, b)$
  - $\gcd(2a, 2b + 1) = \gcd(a, 2b + 1)$ and $\gcd(2a + 1, 2b) = \gcd(2a + 1, b)$.
  - $\gcd(2a + 1, 2b + 1) = \gcd(((2a + 1) - (2b + 1))/2, 2b + 1) = \gcd(a - b, 2b + 1)$ if $a \geq b$, and otherwise $\gcd(b - a, 2a + 1)$

  Given any two integers $z_1 = q_1 \cdot p + 2r_1 + m_1$ and $z_2 = q_2 \cdot p + 2r_2 + m_2$ with $r_1, r_2 \ll p$, we can compute $z^* = s \cdot p + r^*$ where $r^* \ll p$ and s is the final odd part returned by the binary GCD algorithm (in the step $\gcd(0, s) = s$). At each step the algorithm only needs to know the last bits of the input numbers, which can be done using the LSB subroutine from above. It can also be shown that we can disregard the noise during the comparison, subtraction and division operations.

- Notice that the number s from $z^* = s \cdot p + r^*$ will always be odd. Moreover, with probability at least $\pi/6 \approx 0.6$ it will be 1 (i.e. z_1 and z_2 are co-prime), in which case z^* will be equal to the value of $\tilde{z} = 1 \cdot p + r$ for a small error component r. It then remains to run the binary GCD algorithm on numbers $z_1 = q_1 \cdot p + r_1$ and $\tilde{z} = 1 \cdot p + r$ to get the binary representation of q_1 (by watching the parity of the z_1-related number throughout the iterations of the binary GCD algorithm). We restore p by computing the quotient of the division z_1/q_1 (i.e. $\lfloor z_1/q_1 \rceil$). □

Notice that in Section 4.1 we separately introduced the noise parameter $\rho = \omega(\log \lambda)$ and the secondary noise parameter $\rho' = \rho + \omega(\log \lambda)$. We have to do this because the security reduction above involves a loss of parameters. We basically convert an attack

on the ciphertext with ρ' noise into an ability to solve the approximate GCD problem with ρ noise.

It is known that the approximate GCD problem is hard with arbitrarily many samples available to the adversary [22, Section 3.4]. The currently best known exhaustive attack on the scheme with x_0 being a multiple of the secret key p runs in time $\tilde{\mathcal{O}}(2^{\rho/2})$ [14]. If the public key contains no exact multiples of p, then the exhaustive attack requires time $\tilde{\mathcal{O}}(2^{\rho})$ [16, Section 6]. Additionally, [49] demonstrates a CCA1 attack against the vDGHV scheme which requires $O(\lambda^2)$ queries to the decryption oracle. The same attack can be applied to the somewhat homomorphic and fully homomorphic variants of this scheme. The trivial attack against the fully homomorphic variant would require γ queries, one for every encrypted bit of the private key[3].

4.4 Managing the Noise

In the previous sections we described a secure somewhat homomorphic scheme. In order to make it fully homomorphic we would like to show this scheme can be made bootstrappable. In this section we try to estimate what circuits can be evaluated homomorphically using this scheme.

Recall that when reducing some number z modulo p we get a result in the interval $(-p/2, p/2]$. This basically mimics the behaviour of the nearest neighbour decoding. We can guarantee that the ciphertext will be decoded correctly as long as the absolute value of the noise always remains less than $p/2$.

Consider the increase of the noise when computing the arithmetic operations on two ciphertexts:

- $(q_1 p + \underline{2r_1 + m_1}) + (q_2 p + \underline{2r_2 + m_2}) = (q_1 + q_2)p + (\underline{2r_1 + m_1} + \underline{2r_2 + m_2})$
- $(q_1 p + \underline{2r_1 + m_1}) \cdot (q_2 p + \underline{2r_2 + m_2}) = (q_1 q_2 p + 2r_1 + m_1 + 2r_2 + m_2)p + (\underline{2r_1 + m_1}) \cdot (\underline{2r_2 + m_2})$

When adding or multiplying the ciphertexts, the corresponding noise also gets added or multiplied respectively. So computing a function f on ciphertexts $c_1, ..., c_\ell$ can be written as $f(c_1, ..., c_\ell) = f(2r_1 + m_1, ..., 2r_\ell + m_\ell) + kp$ for some integer k. We will reduce this expression modulo p and modulo 2 during the decryption, so we want that the result of $[f(2r_1 + m_1, ..., 2r_\ell + m_\ell) + kp]_p = f(2r_1 + m_1, ..., 2r_\ell + m_\ell)$ is within $(-p/2, p/2]$. Then it will also hold after reducing it modulo 2.

[3] The size of the public key gets increased to γ during the circuit squashing process in Section 4.5, so that the scheme can become bootstrappable.

Moreover, the encryption scheme can handle the function f if and only if $|f(2r_1 + m_1, 2r_2 + m_2, ..., 2r_\ell + m_\ell)| < p/2$ holds for *every* possible set of noise values $(2r_1 + m_1, ..., 2r_\ell + m_\ell)$. Namely, it is not sufficient that one specific parameter set satisfies this inequality, because some of the intermediate results might have exceeded $p/2$ which generally means we can no longer trust the function output.

Now consider a multivariate polynomial $z \in \mathbb{Z}[X]$ of degree d which corresponds to the processed function f (or circuit C). If the noise of the fresh ciphertexts is bounded by some value B, then an encryption scheme can handle the polynomial z if

$$|\mathbf{v_z}|_1 \cdot B^d < p/2 \tag{4.1}$$

where $|\mathbf{v_z}|$ is the l_1 norm of the coefficient vector or z. This requirement is sufficient but not necessary, as we build this inequality for the worst case when all monomials of z have degree d. If all input ciphertexts have the biggest allowed noise B then the output noise of the polynomial is $|\mathbf{v_z}|_1 \cdot B^d$. We need this output noise to be less than $p/2$ in order to decrypt it correctly.

In [47, Section 3.2] it is shown that the fresh ciphertexts contain at most $B = 2^{\rho'+2}$. Additionally, they only allow the output of Evaluate to have noise at most $2^{(\eta-4)} < p/8$ instead of $p/2$ because the evaluation of the decryption circuit will introduce a small precision error which needs to be compensated for (see Section 4.5). By plugging these numbers in the equation (4.1) above and expressing it differently, they get $d \leq \frac{\eta - 4 - \log |\mathbf{v_p}|_1}{\rho' + 2}$. Further on, it is assumed that $\log |\mathbf{v_p}|_1$ is small in relation to η and that we need to support polynomials of degree up to $\alpha\lambda \log^2 \lambda$ for some constant α. This yields the requirement of $\eta = \rho' \cdot \Theta(\lambda \log^2 \lambda)$.

Notice that we can rewrite our decryption formula $m = (c \bmod p) \bmod 2$ by expressing $c \bmod p$ using division and subtraction, to get $m = (c - p \cdot \lfloor c/p \rceil) \bmod 2$. Further on, subtraction modulo 2 can be implemented using $\oplus$ (i.e. logic XOR gate), so $m = [c]_2 \oplus [p \cdot \lfloor c/p \rceil]_2$. Now use the fact that p is odd to get $m = [c]_2 \oplus [\lfloor c/p \rceil]_2$.

Unfortunately, the corresponding decryption circuit is still too big to be evaluated homomorphically. In order to multiply c and $1/p$ we need at least $\log p$ bits of precision. Since $1/p$ is a real number, it can happen that the bit we want to compute (i.e. eventually it is $[\lfloor c/p \rceil]_2$) is one of the most significant bits of the result. By performing the primary-school multiplication algorithm it is easy to notice that each bit of the result depends on the carry bits of all less significant bits. Hence the degree of the polynomial we want to compute is roughly $d = \log p$.

Consider the equation 4.1 again. We choose the smallest possible value for $|\mathbf{v_z}|_1 = 1$ and plug inside the value $d = \log p$ to get the requirement $1 \cdot B^{\log p} < p/2$ where

$B^{\log p} = (2^{\log B})^{\log p} = (2^{\log p})^{\log B} = p^{\log B}$. Hence it is required that $p^{\log B} < p/2$. At this point it is clear that a secure scheme cannot satisfy this requirement.

4.5 Squashing the Decryption Circuit

All of the early fully homomorphic encryption schemes (such as [21, 47, 12]) could not evaluate their own decryption circuit naturally. In order to solve this problem, in his initial construction, Gentry suggested to "squash" the decryption circuit.

The idea of squashing consists of placing some "hint" in the public key[4], which helps to simplify the decryption of ciphertexts. The "hint" is some information about the initial private key, hidden as a subset of fixed size inside the bigger set contained in the public key. The private key is then replaced by a characteristic bit vector which indicates the elements of the public key set that belong to the "hint". This requires another assumption that a *sparse subset sum* problem is computationally hard.

Definition 4.5.1 (Sparse Subset Sum Problem [22, Section 5.4])**.** *Given a set of β numbers $\vec{y}$ and another number s, find the sparse (α-element) subset of $\vec{y}$ whose sum is s.*

In the case of the integer scheme, computing the value of c/p is the most computationally complicated step. To avoid this step, we compute the value $1/p$ during the KeyGen algorithm and hide it in the sparse subset inside the public key. Specifically, the sum of the sparse subset should be equal to $1/p$ within a small precision error.

After every encryption or evaluation algorithm call we multiply each of the numbers from the big public key set with the ciphertext, and store the resulting "scaled" set as a part of the ciphertext. It only remains for the decryption algorithm to choose the correct elements of the "scaled" set and sum them together to get c/p. Since the addition produces much less noise than the multiplication, this considerably simplifies the decryption circuit.

In [47] the squashing step introduces three more parameters: set size $\Theta = \omega(\kappa \cdot \log \lambda)$, subset size $\theta = \lambda$, and the required precision of the real numbers in the set $\kappa = \gamma + 2$. Specifically, we add a set $\vec{y} = \{y_1, ..., y_\Theta\}$ to the public key. This set will contain real values in $[0, 2)$ with κ bits of precision. We redefine the public key as $pk^* = (\langle x_0, x_1, ..., x_\tau \rangle, \vec{y})$ where $\langle x_0, x_1, ..., x_\tau \rangle$ is the old public key. We create a subset $S \subset \{1, ..., \Theta\}$ of size θ and require that $\sum_{i \in S} y_i \approx 1/p \pmod 2$. We redefine the private key as $\mathbf{p}^*$, a characteristic bit-vector of length Θ, such that $p_i^* = 1$ if and only if $i \in S$.

[4] In this case the "hint" is needed for both the Encrypt and the Evaluate algorithms, so we will say the "hint" is contained in the *public* key.

The algorithms of the encryption scheme should be changed in the following way:

- KeyGen:

  Define $\beta = \lfloor 2^\kappa / p \rceil$. Create a random private key bit-vector $\mathbf{p}^*$ of size Θ and weight θ, and set $S = \{i : p_i^* = 1\}$.

  Randomly choose $u_i \in \mathbb{Z} \cap [0, 2^{\kappa+1})$ for $i = \{1, ..., \Theta\}$ such that $\sum_{i \in S} u_i = \beta \pmod{2^{\kappa+1}}$. Finally, define a (public key) set $\vec{y} = \{y_1, ..., y_\Theta\}$ where $y_i = u_i / 2^\kappa$.

- Encrypt and Evaluate:

  Compute the output ciphertext c as before. Afterwards compute a set $\vec{z} = \{z_1, ..., z_\Theta\}$ such that $z_i = [c \cdot y_i]_2$ for all $i \in \{1, ..., \Theta\}$. Define a new ciphertext $c^* = (c, \vec{z})$.

- Decrypt:

  Given a ciphertext $c^* = (c, \vec{z})$ and a private key $\mathbf{p}^*$, compute $m = [c - \lfloor \sum_i \mathbf{p}_i^* \cdot z_i \rceil]_2$.

The assumption of the sparse subset sum hardness was not previously studied much, therefore the squashing step is considered to be the main problem of Gentry's construction. In the next chapter we will see the modulus-switching technique which helps to avoid the squashing step.

Chapter 5

Lattice-based Encryption Scheme

Recall that in order for the encryption scheme to be boostrappable, its decryption algorithm needs to have a low circuit complexity. The schemes using lattices is a natural choice because of their very simple decryption circuits, consisting mostly of matrix multiplication with vector, an operation belonging to NC^1.

In this chapter we discuss the construction of the fully homomorphic encryption scheme which rely on the lattice related computational hardness assumptions. The initial scheme by Gentry [21, 20] worked directly with ideal latices. However, similarly as with the sparse subset sum problem, the ideal lattices are also a relatively new topic which did not receive extensive attention before. Instead, in this chapter we focus on the problem which is proven to be as hard as the well-known standard lattice problem. For this purpose we start by defining lattices and the related hard problems.

5.1 Introduction to Lattices

Definition 5.1.1 (Lattice)**.** *Given m linearly independent vectors $\mathbf{b}_1, ..., \mathbf{b}_m \in \mathbb{R}^n$, we define lattice as the set of all their integer combinations $L = \{\sum_{i=1}^{m} x_i \mathbf{b}_i : x_i \in \mathbb{Z}\}$.*

We say that n and m are the *dimension* and the *rank* of the lattice accordingly. The vectors $\mathbf{b}_1, ..., \mathbf{b}_m$ are called a *basis* of the lattice. We represent a basis as a matrix whose columns are $\mathbf{b}_i$, i.e. $\mathbf{B} = (\mathbf{b}_1|...|\mathbf{b}_m) \in \mathbb{R}^{n \times m}$. Then a lattice can be alternatively defined as $L = \{\mathbf{Bx} : x \in \mathbb{Z}^m\}$.

We say that two bases $\mathbf{B}$ and $\mathbf{B}'$ are equivalent if they generate the same lattice. We call a matrix $\mathbf{U} \in \mathbf{Z}^{m \times m}$ *unimodular* if $\det(\mathbf{U}) = \pm 1$. Two bases are equivalent if and only if there exists some unimodular matrix $\mathbf{U} \in \mathbf{Z}^{m \times m}$ such that $\mathbf{B}' = \mathbf{BU}$. It also follows that $\det(\mathbf{B}') = \det(\mathbf{BU}) = \det(\mathbf{B})$, i.e. there is a unique determinant value

corresponding to every lattice. Since any unimodular matrix can be used to construct a new basis, there are obviously infinitely many equivalent bases for each lattice.

Remark 5.1.2 (Notation). *We denote inner product of vectors* $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ *by* $\langle \mathbf{v}, \mathbf{w} \rangle = \sum_{i=1}^{n} \mathbf{v}_i \cdot \mathbf{w}_i$. *We denote Euclidean norm of a vector* $\mathbf{b}_i$ *by* $\|\mathbf{b}_i\| = \sqrt{\langle \mathbf{b}_i, \mathbf{b}_i \rangle}$.

The following classical lattice problems are known to be NP-hard.

Definition 5.1.3 (Shortest Vector Problem (SVP)). *Given a basis* $\mathbf{B}$ *for lattice* L, *find a non-zero lattice vector* $\mathbf{v} \in L$ *such that* $\|\mathbf{v}\|$ *is minimal.*

Definition 5.1.4 (Closest Vector Problem (CVP)). *Given a basis* $\mathbf{B}$ *for lattice* L *and some vector* $\mathbf{w} \in \mathbb{R}^n$, *find a non-zero lattice vector* $\mathbf{v} \in L$ *such that* $\|\mathbf{w} - \mathbf{v}\|$ *is minimal.*

There are "good" bases and "bad" bases. Intuitively, the "good" bases contain short vectors which facilitates solving the above problems. The "bad" bases are then the ones containing long vectors which are far apart from each other. Given a "bad" basis, it is computationally hard to find a "good" basis for the same lattice. Therefore, one can build cryptosystems such as Goldreich-Goldwasser-Halevi (GGH) which use a "good" basis as a private key and a "bad" basis as a public key.

5.2 LWE-based Encryption Scheme

In this section we discuss a cryptosystem based on the "Learning With Errors" (LWE) problem. This problem was first introduced in [37], and it is proven that LWE is as hard as solving particular versions of shortest vector problems in general lattices [35].

Definition 5.2.1 ((Informal) Learning With Errors (LWE) [11, Page 2]). *If* $s \in \mathbb{Z}_q^n$ *is an* n *dimensional "secret" vector, any polynomial number of "noisy" random linear combinations of the coefficients of* $\mathbf{s}$ *are computationally indistinguishable from uniformly random elements in* $\mathbb{Z}_q$. *Specifically,* $\{\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i\}_{i=1}^{poly(n)} \overset{c}{\approx} \{\mathbf{a}_i, u_i\}_{i=1}^{poly(n)}$ *where* $\mathbf{a}_i \in \mathbb{Z}_q^n$ *and* $u_i \in \mathbb{Z}_q$ *are uniformly random, and the "noise"* e_i *is sampled from a noise distribution* χ *that outputs numbers much smaller than* q.

We first define the public-key cryptosystem based on LWE, and then separately verify the decryption correctness and discuss the homomorphic properties.

Definition 5.2.2 (LWE-based Encryption Scheme, adapted from [11, Section 4.1]).

- $\mathsf{KeyGen}(1^\lambda)$:

  outputs public key $(\mathbf{A}, \mathbf{b})$ *and private key* $\mathbf{s}$, *where* $\mathbf{b} = \mathbf{A}\mathbf{s} + 2\mathbf{e}$, $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{m \times n}$, $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$ *and* $\mathbf{e} \xleftarrow{\$} \chi^m$.

- Encrypt(*public key* $(\mathbf{A}, \mathbf{b})$, *message* $m \in \{0,1\}$)*:*

  outputs $c = (\mathbf{A}^T\mathbf{r}, \mathbf{b}^T\mathbf{r} + m)$ *where* $\mathbf{r} \xleftarrow{\$} \{0,1\}^m$.

- Decrypt(*private key* $\mathbf{s}$, *ciphertext* $(\mathbf{v}, w)$)*:*

  outputs $m = (w - \langle \mathbf{v}, \mathbf{s} \rangle \bmod q) \bmod 2$.

First of all, it is easy to check that the decryption correctness property is satisfied:

$$
\begin{aligned}
& m = (w - \langle \mathbf{v}, \mathbf{s} \rangle \bmod q) \bmod 2 \\
&= \left[\mathbf{b}^T\mathbf{r} + m - \langle \mathbf{A}^T\mathbf{r}, \mathbf{s} \rangle\right]_q \bmod 2 \\
&= \left[(\mathbf{As})^T\mathbf{r} + 2\mathbf{e}^T\mathbf{r} + m - (\mathbf{A}^T\mathbf{r})^T\mathbf{s}\right]_q \bmod 2 \\
&= \left[((\mathbf{As})^T\mathbf{r})^T + 2\mathbf{e}^T\mathbf{r} + m - \mathbf{r}^T\mathbf{As}\right]_q \bmod 2 \\
&= \left[\mathbf{r}^T\mathbf{As} + 2\mathbf{e}^T\mathbf{r} + m - \mathbf{r}^T\mathbf{As}\right]_q \bmod 2 \\
&= [2\mathbf{e}^T\mathbf{r} + m]_2 \\
&= [m]_2
\end{aligned}
$$

Secondly, we can express the ciphertext as:

$$
\begin{aligned}
& c = (\mathbf{A}^T\mathbf{r}, \mathbf{b}^T\mathbf{r} + m) \\
&= (\mathbf{A}^T\mathbf{r}, (\mathbf{As})^T\mathbf{r} + 2\mathbf{e}^T\mathbf{r} + m) \\
&= (\mathbf{A}^T\mathbf{r}, \mathbf{s}^T\mathbf{A}^T\mathbf{r} + 2\mathbf{e}^T\mathbf{r} + m) \\
&= (\mathbf{A}^T\mathbf{r}, (\mathbf{s}^T(\mathbf{A}^T\mathbf{r}))^T + 2\mathbf{e}^T\mathbf{r} + m) \\
&= (\mathbf{A}^T\mathbf{r}, (\mathbf{A}^T\mathbf{r})^T\mathbf{s} + 2\mathbf{e}^T\mathbf{r} + m) \\
&= (\mathbf{A}^T\mathbf{r}, \langle \mathbf{A}^T\mathbf{r}, \mathbf{s} \rangle + 2\mathbf{e}^T\mathbf{r} + m)
\end{aligned}
$$

Here $\mathbf{A}^T\mathbf{r}$ is a random subset sum of columns of matrix $\mathbf{A}^T$, and $\mathbf{s}$ is a "secret" key, and $2\mathbf{e}^T\mathbf{r} + m$ is a noise which is considerably smaller than q. One can intuitively see that the security of this cryptosystem relies on LWE.

In [11, Section 1.1] the supported homomorphic operations of this cryptosystem are considered by describing the decryption function: $f_{(\mathbf{a},b)} : \mathbb{Z}_q^n \to \mathbb{Z}_q$ such that

$$
f_{(\mathbf{a},b)}(\mathbf{x}) = b - \langle \mathbf{a}, \mathbf{x} \rangle = b - \sum_{i=1}^{n} \mathbf{a}[i] \cdot \mathbf{x}[i] \pmod q
$$

where $(\mathbf{a}, b)$ is a ciphertext and the argument $\mathbf{x}$ is the secret key. We can now consider the homomorphic operations with respect to this decryption function.

First of all, $f_{\mathbf{a},b}$ is a linear function in its input argument $\mathbf{x}$, hence the homomorphic addition is trivial: $f_{(\mathbf{a},b)} + f_{(\mathbf{a}',b')} = f_{(\mathbf{a}+\mathbf{a}',b+b')}$.

Now we try the same approach for the homomorphic multiplication:

$$
\begin{aligned}
& f_{(\mathbf{a},b)}(\mathbf{x}) \cdot f_{(\mathbf{a}',b')}(\mathbf{x}) \\
&= (b - \sum \mathbf{a}[i]\mathbf{x}[i]) \cdot (b' - \sum \mathbf{a}'[i]\mathbf{x}[i]) \pmod q
\end{aligned}
$$

$$= h_0 + \sum h_i \cdot \mathbf{x}[i] + \sum h_{i,j} \cdot \mathbf{x}[i]\mathbf{x}[j] \pmod{q} \tag{5.1}$$

where $h_0, h_i, h_{i,j}$ are some coefficients containing values of $\mathbf{a}, \mathbf{a}', b, b'$ and obtainable by opening the parentheses above.

Each of the initial ciphertexts contains $n+1$ values, i.e. n values in vector $\mathbf{a}$ and one more value is the scalar b. One can observe that by multiplying two ciphertexts we get a result with $1+n+n\cdot(n-1)/2$ coefficients. So every multiplication can basically square the size of the ciphertext, and overall this size can increase exponentially in the number of multiplications.

5.2.1 Re-Linearization Technique

In order to avoid the ciphertext size blow-up, the "re-linearization" technique is offered in [11, Section 1.1]. It is suggested to use the "encryptions" of the private key $\mathbf{s}$ under some other key $\mathbf{t} \in \mathbb{Z}_q^n$. These "encryptions" should be pre-computed and included in the evaluation key.

More precisely, we want to express both the linear variables $\mathbf{x}[i]$ and the quadratic variables $\mathbf{x}[i]\mathbf{x}[j]$ from Equation (5.1) as some functions $g_i(\mathbf{t}) \approx \mathbf{x}[i]$ and $g_{i,j}(\mathbf{t}) \approx \mathbf{x}[i]\mathbf{x}[j]$, which are linear in their input argument $\mathbf{t}$. After performing a multiplication, we can then transform Equation (5.1) into a ciphertext of size $n+1$ again, by replacing $\mathbf{x}[i]$ and $\mathbf{x}[i]\mathbf{x}[j]$ with $g_i(\mathbf{t})$ and $g_{i,j}(\mathbf{t})$ respectively. We can think about functions $g_i(\mathbf{t})$ and $g_{i,j}(\mathbf{t})$ as "encryptions" of the values $\mathbf{x}[i]$ and $\mathbf{x}[i]\mathbf{x}[j]$.

Specifically, the bits of the private key $\mathbf{s}$ should be "encrypted" under another secret key $\mathbf{t}$ and stored in the evaluation key in the following way:

- $(\mathbf{a}_i, b_i)$ where $b_i = \langle \mathbf{a}_i, \mathbf{t} \rangle + 2e_i + \mathbf{s}[i] \pmod{q}$ for all $i \in \{1, ..., n\}$
- $(\mathbf{a}_{i,j}, b_{i,j})$ where $b_{i,j} = \langle \mathbf{a}_{i,j}, \mathbf{t} \rangle + 2e_{i,j} + \mathbf{s}[i]\mathbf{s}[j] \pmod{q}$ for all $i \leq j$ from $\{1, ..., n\}$

where $\mathbf{a}_i, \mathbf{a}_{i,j} \in \mathbb{Z}_q^n$ are uniformly random, and $e_i, e_{i,j} \in \chi'$ are small error vectors. Then we define our symbolic functions from above as $g_i(\mathbf{t}) \equiv b_i - \langle \mathbf{a}_i, \mathbf{t} \rangle = 2e_i + \mathbf{s}[i] \approx \mathbf{s}[i]$ and $g_{i,j}(\mathbf{t}) \equiv b_{i,j} - \langle \mathbf{a}_{i,j}, \mathbf{t} \rangle = 2e_{i,j} + \mathbf{s}[i]\mathbf{s}[j] \approx \mathbf{s}[i]\mathbf{s}[j]$.

The LWE hardness assumption obviously holds for these pairs of values, however they are not really valid ciphertexts in our cryptosystem because the values $\mathbf{s}[i] \in \mathbb{Z}_q$ and $\mathbf{s}[i]\mathbf{s}[j] \pmod{q} \in \mathbb{Z}_q$ do not belong to the plaintext space $\{0, 1\}$.

Now we can rewrite (5.1) as a linear function in $\mathbf{t}$:

$$h_0 + \sum h_i(b_i - \langle \mathbf{a}_i, \mathbf{t} \rangle) + \sum_{i,j} h_{i,j} \cdot (b_{i,j} - \langle \mathbf{a}_{i,j}, \mathbf{t} \rangle) \pmod{q}. \tag{5.2}$$

Hence we can "compress" the resulting ciphertext back to the initial size $n+1$ using the additional information from the evaluation key.

We can repeatedly perform re-linearization after every level of topologically sorted gates of the circuit. For a circuit of depth L we need to store encryptions of L subsequent keys, implicitly changing our private key from initial $\mathbf{s}$ into $\mathbf{t}$, and then into $\mathbf{t_1}$, $\mathbf{t_2}$, ..., $\mathbf{t_{L-1}}$. We only need to know the key $\mathbf{t_{L-1}}$ so that we can decrypt the result after the evaluation of the circuit is finished. However, this scheme is just somewhat homomorphic, not leveled fully homomorphic, because the ciphertext noise keeps increasing. Unlike the Recrypt algorithm we saw in Section 3.3, the re-linearization process does not remove the noise, it only allows to avoid the exponential growth of the ciphertext size and, in fact, also slightly increases the noise.

However, there's still a problem with the Equation (5.2). Notice that $(b_{i,j} - \langle \mathbf{a}_{i,j}, \mathbf{t} \rangle) = 2e_{i,j} + \mathbf{s}[i]\mathbf{s}[j]$. When we multiply $h_{i,j} \cdot (b_{i,j} - \langle \mathbf{a}_{i,j}, \mathbf{t} \rangle) = h_{i,j} \cdot (2e_{i,j} + \mathbf{s}[i]\mathbf{s}[j])$, the resulting noise component $h_{i,j} \cdot 2e_{i,j}$ might become too big (same applies to the multiplication $h_i \cdot 2e_i$).

In [11, Section 4.1] it is handled by considering the binary representation $h_{i,j} = \sum_{\tau=0}^{\lfloor \log q \rfloor} 2^\tau \cdot h_{i,j,r}$ where $h_{i,j,\tau} \in \{0,1\}$. It is suggested to compute $(\mathbf{a}_{i,j,\tau}, b_{i,j,\tau})$ for each value of τ from $\{0, ..., \lfloor \log q \rfloor\}$, where $b_{i,j,\tau} = \langle \mathbf{a}_{i,j,\tau}, \mathbf{t} \rangle + 2e_{i,j,\tau} + 2^\tau \mathbf{s}[i] \cdot \mathbf{s}[j]$ With this change we now manage to keep the noise under control:

$$h_{i,j} \cdot \mathbf{s}[i]\mathbf{s}[j] = \sum_{\tau=0}^{\lfloor \log q \rfloor} h_{i,j,\tau} 2^\tau \mathbf{s}[i]\mathbf{s}[j] \approx \sum_{\tau=0}^{\lfloor \log q \rfloor} h_{i,j,\tau} (b_{i,j,\tau} - \langle \mathbf{a}_{i,j,\tau}, \mathbf{t} \rangle).$$

In other words, we implicitly handle the multiplication $h_{i,j} \cdot (2e_{i,j} + \mathbf{s}[i]\mathbf{s}[j])$ bit-by-bit with respect to the binary representation of $h_{i,j}$. For every 2^τ bit-component of $h_{i,j}$, we artificially bring this power of 2 inside the brackets, but we "forget" to multiply it with the noise: $h_{i,j,\tau} 2^\tau \cdot (2e_{i,j} + \mathbf{s}[i]\mathbf{s}[j])$ gets transformed into $h_{i,j,\tau} \cdot (2e_{i,j} + 2^\tau \mathbf{s}[i]\mathbf{s}[j])$. This trick reduces the noise increase during the re-linearization process, however it also increases the size of the evaluation key by a magnitude of $\tau + 1 \approx \log q + 1$ times.

5.2.2 Dimension-Modulus Reduction Technique

Having introduced the re-linearization process, we could now squash the decryption circuit to make the scheme bootstrappable and hence fully homomorphic. However, we will instead use the re-linearization process in a way which helps to avoid squashing the circuit, as described in [11, Section 1.2].

The decryption function $f_{(\mathbf{a},b)}(\mathbf{x})$ of the encryption scheme is a linear function in its

input argument $\mathbf{x}$, which is a vector of size roughly n. And we perform all computations modulo q. Therefore, it is safe to say that the depth of the decryption circuit depends on the values n and $\log q$, e.g. its depth is at least $\max(n, \log q)$. In order to simplify the decryption complexity, we will to reduce both of these values now.

The first observation is that the size k of the new re-linearization key $\mathbf{t} \in \mathbb{Z}_q^k$ from the previous section should not definitely be equal to n (the size of the initial key $\mathbf{s}$). Instead, we can choose any size we want, including one which is considerably smaller than n. It is proven in [3, Section 3.1] that LWE is hard even when the size of its secret key is small (e.g. chosen from the error distribution).

Secondly, in order to decrease the value of $\log q$, we can simply scale down the modulus by applying the re-linearization to any ciphertext. The evaluation key should then also contain pairs $(\mathbf{a}_{i,\tau}, b_{i,\tau}) \in \mathbb{Z}_p^k \times \mathbb{Z}_p$ such that

$$b_{i,\tau} = \langle \mathbf{a}_{i,\tau}, \mathbf{t} \rangle + e + \left\lfloor \frac{p}{q} \cdot 2^\tau \cdot \mathbf{s}[i] \right\rceil$$

where p is the new modulus, $p < q$ and $p \equiv q \pmod 2$[1]. We can also express the equation above as:

$$2^\tau \cdot \mathbf{s}[i] \approx \frac{q}{p} \cdot (b_{i,\tau} - \langle \mathbf{a}_{i,\tau}, \mathbf{t} \rangle).$$

So we can basically swap the initial key $\mathbf{s}$ into a new key $\mathbf{t}$, while also scaling down both the ciphertext size and modulus from $\mathbb{Z}_q$ into $\mathbb{Z}_p$[2].

Therefore, we are now able to reduce the values n and $\log q$ down to some k and $\log p$. We then use the somewhat homomorphic scheme as described in the section above, swapping L keys to evaluate any circuit of depth L. Once we are finished computing the circuit, we have accumulated a lot of noise and would like to reset it. We can then scale down the modulus (greatly reducing the depth of the decryption circuit), hence making the scheme bootstrappable (and eventually fully homomorphic).

Remark 5.2.3 (Chimeric Fully Homomorphic Encryption)**.** *The dimension-modulus reduction technique is one of the two entirely different ways known to avoid the step of squashing the decryption circuit (which depends on the relatively little studied and hence undesirable "sparse subset sum" assumption). The other known way to avoid the squashing was offered in [23] and is sometimes referred to as "Chimeric Fully Homomorphic Encryption".*

[1] The reason we use noise value e instead of $2e$ is (roughly) because, in order to preserve an even error vector, we divide the ciphertext by 2 before applying re-linearization, and multiply it back by 2 afterwards.

[2] Rather than explaining this transformation in details, our goal is to provide an overall idea before discussing a further and simpler improvement in the next section.

It expresses the decryption function as a depth-3 arithmetic circuit (including addition and multiplication, i.e. $\sum\prod\sum$). In order to compute the products, the construction temporarily switches to some multiplicative homomorphic encryption scheme (MHE), such as Elgamal. It does not require to homomorphically evaluate its own decryption circuit, neither it needs to perform any computations to switch to MHE. It only needs to be able to homomorphically evaluate the decryption circuit of the MHE scheme after the homomorphic computation of products is finished.

5.3 Modulus-Switching Technique

In the previous section we discussed the dimension-modulus reduction technique [11] which allowed to scale down the size of both the ciphertext and the modulus. It was used to simplify the decryption circuit and hence make the LWE-based encryption scheme bootstrappable.

In this section we present a major improvement of dimension-modulus reduction idea, which at the same time also became a lot simpler and very intuitive [10]. The new technique is intended to manage the noise by repeatedly reducing the modulus after every topologically sorted circuit level. Similar to the dimension-modulus approach, it also helps to avoid the circuit squashing step, but it does not require to use the evaluation key. Moreover, it can also be used to avoid the bootstrapping process entirely.

Given a bound on the l_1-norm of the secret key, the modulus-switching technique allows to reduce the ciphertext and the corresponding modulus by simply scaling both of them down and rounding the resulting ciphertext appropriately. Below we provide the lemma which describes the modulus-switching technique, along with its exact proof.

Lemma 5.3.1 ([10, Lemma 1])**.** *Let p and q be two odd moduli, and let $\mathbf{c}$ be an integer vector. Define $\mathbf{c}'$ to be the integer vector closest to $(p/q)\cdot\mathbf{c}$ such that $\mathbf{c}' = \mathbf{c} \bmod 2$. Then, for any $\mathbf{s}$ with $|[\langle\mathbf{c},\mathbf{s}\rangle]_q| < q/2 - (q/p)\cdot l_1(\mathbf{s})$, we have*

$$[\langle\mathbf{c}',\mathbf{s}\rangle]_p = [\langle\mathbf{c},\mathbf{s}\rangle]_q \bmod 2 \text{ and } |[\langle\mathbf{c}',\mathbf{s}\rangle]_p| < (p/q)\cdot|[\langle\mathbf{c},\mathbf{s}\rangle]_q| + l_1(\mathbf{s})$$

where $l_1(\mathbf{s})$ is the l_1-norm of $\mathbf{s}$.

Proof. ([10, Page 3]) For some integer k, we have $[\langle\mathbf{c},\mathbf{s}\rangle]_q = \langle\mathbf{c},\mathbf{s}\rangle - kq$. For the same k, let $e_p = \langle\mathbf{c}',\mathbf{s}\rangle - kp \in \mathbb{Z}$. Since $\mathbf{c} = \mathbf{c}'$ and $p = q$ modulo 2, we have $e_p = [\langle\mathbf{c},\mathbf{s}\rangle]_q \bmod 2$. Therefore, to prove the lemma, it suffices to prove that $e_p = [\langle\mathbf{c}',\mathbf{s}\rangle]_p$ and that it has small enough norm. We have $e_p = (p/q)[\langle\mathbf{c},\mathbf{s}\rangle]_q + \langle\mathbf{c}' - (p/q)\mathbf{c},\mathbf{s}\rangle$, and therefore $|e_p| \le (p/q)[\langle\mathbf{c},\mathbf{s}\rangle]_q + l_1(\mathbf{s}) < p/2$. The latter inequality implies $e_p = [\langle\mathbf{c}',\mathbf{s}\rangle]_p$. □

Notice that the ratio between the noise and the modulus does not change during the modulus switching (besides a very minor rounding error). However, by scaling down the ciphertext, we reduce the absolute value of the noise, which is also an important indicator. For a moment, for the sake of simplicity assume that $l_1(\mathbf{s}) = 0$. Assume that a fresh ciphertext of some cryptosystem has noise level at most B, and the initial modulus of the cryptosystem is $q_0 = B^9$ (hence the noise threshold is $\approx q_0/2 = B^9/2$). We will consider the growth of the noise in two different situations, without and with the modulus switching:

- (Without the modulus switching). After the first multiplication the noise can increase up to B^2, after the second multiplication up to B^4, after the third multiplication up to B^8, and we can no longer use such ciphertexts in any more multiplications. As already seen before, there are roughly $\log_2 \log_B q_0$ consecutive homomorphic multiplications possible, before the noise becomes too big.

- (With the modulus switching). After the first multiplication the noise can increase up to B^2. We then scale both the noise and the modulus down by a magnitude of B, hence acquiring a new noise value $B^2/B = B$ and a new modulus $q_1 = q_0/B = B^8$ (more formally, we multiply everything by $q_1/q_0 = B^8/B^9 = 1/B$).

  After the second multiplication the noise can increase up to B^2 again, so we continue rescaling: $B^2/B = B$ and $q_2 = q_1/B = B^7$, etc.

  After the seventh multiplication noise can increase up to B^2. We scale down the noise $B^2/B = B$ and the modulus $q_7 = q_6/B = B^2$.

  No more multiplications are allowed because the current noise threshold is $\approx q_7/2 = B^2/2$ and the noise after the next multiplication could become as big as B^2. So it is possible to perform roughly $\log_B q_0$ consecutive homomorphic multiplications when using the modulus switching, which is an exponential improvement in the noise management!

Since $l_1(\mathbf{s}) \neq 0$, we actually need to choose the moduli more careful, so that the inequality $|[\langle \mathbf{c}, \mathbf{s} \rangle]_q| < q/2 - (q/p) \cdot l_1(\mathbf{s})$ is always satisfied. But it can still be done, for example, if third party is provided with an upper bound of the possible $l_1(\mathbf{s})$ value.

Similar to some other techniques, the modulus switching should be applied after processing each of the topologically sorted circuit levels. If we know the circuit depth beforehand, we can choose the initial modulus big enough so that we can compute the whole circuit without ever having to refresh the ciphertext using the bootstrapping. Alternatively, it is also possible to use the modulus switching approach *together* with bootstrapping so that the scheme can be converted in a fully homomorphic scheme.

The modulus switching is a general technique that can be applied to various somewhat homomorphic encryption schemes, some of them listed below:

- The LWE-based encryption scheme [11] defined in Section 5.2.
- The RLWE-based ("*Ring Learning With Errors*") encryption scheme [12], which is basically the LWE scheme which operates over polynomial rings instead of vectors.
- The NTRU encryption scheme [32], which is also a noisy scheme based on lattices.
- The vDGHV encryption scheme [47] from Chapter 4, for which the modulus switching was described in [16]. Since the modulus in this scheme is a private key, the switching process is more complicated.

All of the LWE, RLWE and NTRU schemes from above also benefit from the relinearization technique.

Remark 5.3.2 (Scale-Invariant Fully Homomorphic Encryption)**.** *Finally, the recent results in [9] further improve the noise management, comparing even to the modulus-switching technique. It is shown how to achieve a linear growth of noise after multiplication, without switching the modulus.*

Chapter 6

Implementing Homomorphic Encryption Schemes

The main efficiency bottleneck of fully homomorphic encryption schemes is obviously the bootstrapping process. In order to refresh a ciphertext, it is required to run the decryption circuit homomorphically. Moreover, the encrypted private key bits are required to be passed to the input wires of this circuit. Each of such encrypted bits are represented by a ciphertext of a big size. And the properties of the somewhat homomorphic encryption schemes (which are used to build the fully homomorphic encryption schemes) add even more complexity related to the bootstrapping process. According to the analysis provided in [10, Section 1.1], normally the computational cost of running the bootstrapping a single time is comparable to λ^4, and in the best case could be comparable to λ^3. Finally, it is safe to assume we need to run the bootstrapping procedure for each of the circuit gates, all of which operate on a single (encrypted) bit.

This far there were several attempts to implement the initial Gentry's fully homomorphic encryption scheme from [21, 20] and the fully homomorphic encryption scheme over the integers from [47].

6.1 Gentry's Encryption Scheme

The initial encryption scheme proposed by Gentry in [21, 20] was based on ideal lattices. The first attempt to implement this scheme was done by Smart and Vercauteren in [42]. They implemented the somewhat homomorphic variant of the scheme, but they did not manage to transform it into a fully homomorphic scheme. Their squashed decryption polynomial was estimated to have a degree of a few hundreds, and for that they needed

the lattice dimension to be at least 2^{27}. They also required the determinant of the lattice to be a prime number which resulted in a very inefficient key generation algorithm, so the biggest lattice dimension they could get in practice was 2^{11}.

The next attempt in [24] offered a range of optimizations (including security trade-offs) which resulted in a complete implementation of fully homomorphic encryption for the first time. Amongst other things they eliminate the requirement that the determinant of the lattice be prime, and also show a way to suffice with a decryption polynomial of degree 15. The resulting implementation was tested with lattice dimensions 2^9, 2^{11}, 2^{13} and 2^{15}. According to [13], this corresponds to 52-bit, 61-bit, 72-bit and 100-bit security levels respectively. The achieved public key sizes and the running times of KeyGen and Recrypt algorithms on a strong desktop computer can be seen in Table 6.1.

Security Level	Dimension	Public Key Size	KeyGen	Recrypt
52-bit	2^9	17 MB	2.5 sec	6 sec
61-bit	2^{11}	69 MB	41 sec	32 sec
72-bit	2^{13}	284 MB	8.4 min	2.8 min
100-bit	2^{15}	2.25 GB	2.2 hour	31 min

Table 6.1: The implementation details of the fully homomorphic scheme from [24]

Considering that the Recrypt algorithm might need to be run as often as after every multiplication (and separately for every encrypted bit), it is safe to say that the resulting implementation is highly inefficient.

6.2 Encryption Scheme over the Integers

The first implementation of the fully homomorphic vDGHV scheme [47] was presented in [15]. It adapted various optimizations from [24], and the main improvement of this scheme comparing to the initial vDGHV scheme is outlined in Section 4.2. Its performance is similar to the one achieved by the above implementation of the Gentry's scheme from [24]. However, [14] showed that the chosen security levels of this implementation are actually lower than it was initially expected.

The next implementation by [16] gave a better estimate of expected security levels and added more improvements. This scheme used the public key of size $\tilde{\mathcal{O}}(\lambda^5)$ (see 4.2 for details) and also implemented the modulus-switching technique. Two variants of implementation were tested. The key size and algorithm running speed of the implementations with and without the modulus-switching can be seen in Table 6.2 and Table 6.3 respectively. Although modulus-switching allows to compute a circuit of arbitrary

depth (fixed in advance), the bootstrapping algorithm was still implemented for the corresponding variant. The computer used for taking the measurements had a similar hardware configuration to the one from [24].

Security Level	Public Key Size	KeyGen	Recrypt
42-bit	77 KB	0.06 sec	0.41 sec
52-bit	437 KB	1.3 sec	4.5 sec
62-bit	2207 KB	28 sec	51 sec
72-bit	10.3 MB	10 min	11 min 34 sec

Table 6.2: The implementation details of the fully homomorphic scheme without modulus-switching from [16]

Security Level	Public Key Size	KeyGen	Recrypt
42-bit	354 KB	0.36 sec	8.8 sec
52-bit	1690 KB	5.4 sec	101 sec
62-bit	7.9 MB	1 min 12 sec	32 min 38 sec
72-bit	18 MB	6 min 18 sec	2 hours 27 min

Table 6.3: The implementation details of the fully homomorphic scheme with modulus-switching from [16]

It can be observed that the public key size of both implementations is a lot better than in the above implementation of Gentry's scheme. However, the speed of the Recrypt algorithm is noticeably slower than in the implementation by [24]. The key generation times are roughly similar amongst all three implementations. Despite the 4-times slower speed of Recrypt (even in the implementation without modulus-switching), there is some hope that the conceptually simpler scheme based on the integers can successfully compete with the initial (Gentry's) scheme.

6.3 Further Optimizations

It is possible to perform homomorphic operations on several ciphertexts in parallel, which can considerably speed up fully homomorphic encryption. The related research was started in [43] and followed by [25]. In [25] it is shown how to operate with encryption of plaintext vectors, including a way to permute the data bits inside these vectors. However, these constructions were not tested in any practical implementations yet.

Chapter 7

Conclusions

This dissertation provided an overview of fully homomorphic encryption which deals with the problem of computing arbitrary functions over encrypted data. We started out by informally discussing the general idea behind homomorphic encryption and gradually made our way through to more complicated topics.

This work also focused on such major topics as the general design of fully homomorphic encryption schemes, in particular the details concerning two such schemes based on integers and lattices were discussed, and the efficiency of current practical implementations.

An attempt was made at focusing on concepts which can be found in various contexts of the theory of fully homomorphic encryption, rather than on the ones which are particular to a specific scheme. As such, the general ideas of examining the noise level and squashing the decryption circuit were introduced while discussing the fully homomorphic encryption scheme over the integers. The chapter on the lattice-based scheme explained the re-linearization technique that can be applied to the all known lattice-based schemes, as well as the modulus switching technique which can also be applied to the approximate-GCD based scheme.

Finally, the idea of the noisy encryption schemes is discussed throughout the work, starting from the Polly Cracker cryptosystem whose vulnerabilities introduced the need of adding the noise, and continuing with the Noisy Polly Cracker scheme, the approximate-GCD based scheme and the LWE-based scheme. These schemes demonstrate the common idea behind a bigger number of the fully homomorphic encryption schemes (including the Gentry's scheme over the ideal lattices, the RLWE-based scheme and the NTRU scheme all of which were not defined in this work).

The dissertation ended with a discussion of attempted implementations of fully homomorphic encryption. It was seen that we are still far from achieving truly practical

cryptosystems in this area. However, it should be noted that only 3 years have passed since Gentry's breakthrough work provided the first viable design of a fully homomorphic encryption scheme. Since then, significant improvements have been made. It is altogether plausible that there eventually there will appear results which will make fully homomorphic encryption practical.

Bibliography

[1] N. Ahituv, Y. Lapid, and S. Neumann. Processing encrypted data. *Communications of the ACM*, 30(9):777–780, Sept. 1987. 17

[2] M. R. Albrecht, P. Farshim, J.-C. Faugère, and L. Perret. Polly cracker, revisited. In D. H. Lee and X. Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 179–196. Springer, 2011. 16, 23

[3] B. Applebaum, D. Cash, C. Peikert, and A. Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In S. Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618. Springer, 2009. 43

[4] F. Armknecht and A.-R. Sadeghi. A new approach for algebraically homomorphic encryption. *IACR Cryptology ePrint Archive*, 2008:422, 2008. 17

[5] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In J. Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2001. 12

[6] D. A. M. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. In J. Hartmanis, editor, *STOC*, pages 1–5. ACM, 1986. 17

[7] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In J. Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005. 16

[8] D. Boneh and R. J. Lipton. Algorithms for black-box fields and their application to cryptography (extended abstract). In N. Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 283–297. Springer, 1996. 18

[9] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical gapSVP. *IACR Cryptology ePrint Archive*, 2012:78, 2012. 19, 20, 46

[10] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:111, 2011. 44, 47

[11] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In IEEE, editor, *Proceedings: 2011 IEEE 52nd Annual IEEE Symposium on Foundations of Computer Science: 22–25 October 2011, Palm Springs, California, USA*, pages 97–106, pub-IEEE:adr, 2011. IEEE Computer Society Press. 19, 20, 21, 22, 26, 27, 39, 40, 41, 42, 44, 46

[12] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In P. Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 505–524. Springer, 2011. 36, 46

[13] Y. Chen and P. Q. Nguyen. BKZ 2.0: Better lattice security estimates. In D. H. Lee and X. Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011. 48

[14] Y. Chen and P. Q. Nguyen. Faster algorithms for approximate common divisors: Breaking fully-homomorphic-encryption challenges over the integers. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 502–519. Springer, 2012. 34, 48

[15] J.-S. Coron, A. Mandal, D. Naccache, and M. Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In P. Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 487–504. Springer, 2011. 31, 48

[16] J.-S. Coron, D. Naccache, and M. Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 446–464. Springer, 2012. 32, 34, 46, 48, 49

[17] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, 1985. 16

[18] M. Fellows and N. Koblitz. Combinatorial cryptosystems galore! In *Finite fields: theory, applications, and algorithms (Las Vegas, NV, 1993)*, volume 168 of *Contemp. Math.*, pages 51–61. Amer. Math. Soc., Providence, RI, 1994. 14, 15, 16

[19] C. Fontaine and F. Galand. A survey of homomorphic encryption for nonspecialists. *EURASIP J. Information Security*, 2007, 2007. 16

[20] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`. 5, 6, 16, 17, 19, 21, 22, 23, 25, 27, 38, 47

[21] C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009. 1, 6, 17, 19, 25, 36, 38, 47

[22] C. Gentry. Computing arbitrary functions of encrypted data. *Communications of the ACM*, 53(3):97–105, Mar. 2010. 18, 19, 29, 34, 36

[23] C. Gentry and S. Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In R. Ostrovsky, editor, *FOCS*, pages 107–109. IEEE, 2011. 43

[24] C. Gentry and S. Halevi. Implementing gentry's fully-homomorphic encryption scheme. In K. G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2011. 48, 49

[25] C. Gentry, S. Halevi, and N. P. Smart. Fully homomorphic encryption with polylog overhead. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2012. 49

[26] S. Goldwasser and S. Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In ACM, editor, *Proceedings of the fourteenth annual ACM Symposium on Theory of Computing, San Francisco, California, May 5–7, 1982*, pages 365–377, pub-ACM:adr, 1982. ACM Press. 10

[27] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. See also preliminary version in 14th STOC, 1982. 16

[28] K. Henry. The theory and applications of homomorphic cryptography, 2008. 16, 17

[29] Y. Ishai and A. Paskin. Evaluating branching programs on encrypted data. In S. P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594. Springer, 2007. 17

[30] A. Kerckhoffs. La cryptographie militaire. *Journal des Sciences Militaires*, IX, 1883.

[31] J. Loftus, A. May, N. P. Smart, and F. Vercauteren. On CCA-secure somewhat homomorphic encryption. In A. Miri and S. Vaudenay, editors, *Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 55–72. Springer, 2011. 17

[32] A. López-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In H. J. Karloff and T. Pitassi, editors, *STOC*, pages 1219–1234. ACM, 2012. 46

[33] C. A. Melchor, P. Gaborit, and J. Herranz. Additively homomorphic encryption with d -operand multiplications. In T. Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 138–154. Springer, 2010. 17

[34] M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In C. Cachin and T. Ristenpart, editors, *CCSW*, pages 113–124. ACM, 2011. 24

[35] C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In ACM, editor, *STOC '09: proceedings of the 2009 ACM International Symposium on Theory of Computing, Bethesda, Maryland, USA, May 31–June 2, 2009*, pages 333–342, pub-ACM:adr, 2009. ACM Press. 39

[36] D. K. Rappe. Homomorphic cryptosystems and their applications. *IACR Cryptology ePrint Archive*, 2006:1, 2006. 5

[37] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In ACM, editor, *STOC '05: proceedings of the 37th Annual ACM Symposium on Theory of Computing: Baltimore, Maryland, USA, May 22–24, 2005*, pages 84–93, pub-ACM:adr, 2005. ACM Press. 39

[38] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, Feb. 1978. 9

[39] R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. In R. DeMillo, D. Dobkin, A. Jones, and R. Lipton, editors, *Foundations of Secure Computation*, pages 169–180. Academic Press, 1978. 1, 5, 18

[40] R. Rothblum. Homomorphic encryption: From private-key to public-key. In Y. Ishai, editor, *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 219–234. Springer, 2011. 15

[41] T. Sander, A. Young, and M. Yung. Non-interactive CryptoComputing for NC^1. In *Proceedings of the 40th Symposium on Foundations of Computer Science (FOCS)*, pages 554–567, New York, NY, USA, Oct. 1999. IEEE Computer Society Press. 17

[42] N. P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In P. Q. Nguyen and D. Pointcheval, editors, *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010. 47

[43] N. P. Smart and F. Vercauteren. Fully homomorphic SIMD operations. *IACR Cryptology ePrint Archive*, 2011:133, 2011. 49

[44] D. Stehlé and R. Steinfeld. Faster fully homomorphic encryption. In M. Abe, editor, *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 377–394. Springer, 2010. 27

[45] V. Vaikuntanathan. Computing blindfolded: New developments in fully homomorphic encryption. In R. Ostrovsky, editor, *FOCS*, pages 5–16. IEEE, 2011. 6, 16, 19, 20, 21

[46] W. van Dam, S. Hallgren, and L. Ip. Quantum algorithms for some hidden shift problems. In *SODA*, pages 489–498. ACM/SIAM, 2003. 18

[47] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In H. Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010. 6, 19, 29, 30, 31, 32, 35, 36, 46, 47, 48

[48] Y. Yu, J. Leiwo, and A. B. Premkumar. A study on the security of privacy homomorphism. In *ITNG*, pages 470–475. IEEE Computer Society, 2006. 17

[49] Z. Zhang, T. Plantard, and W. Susilo. On the CCA-1 security of somewhat homomorphic encryption over the integers. In M. D. Ryan, B. Smyth, and G. Wang, editors, *ISPEC*, volume 7232 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2012. 34