université PARIS-SACLAY

NNT : 2018SACLV020

Thèse de doctorat

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

Vers l'efficacité et la sécurité du chiffrement homomorphe et du cloud computing

Thèse de doctorat de l'Université Paris-Saclay
préparée à l'Université de Versailles Saint-Quentin-en-Yvelines

École doctorale n°580 Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat: Informatique

Thèse présentée et soutenue à Versailles, le 17 Mai 2018, par

Ilaria Chillotti

Composition du Jury :

M. Jean-Sébastien CORON Professeur assistent, Université de Luxembourg	Examinateur
Mme Caroline FONTAINE Chargé de recherche, IMT Atlantique	Examinatrice
M. Nicolas GAMA Maître de conférences, Université de Versailles Saint-Quentin-en-Yvelines & Inpher	Co-encadrant
M. Louis GOUBIN Professeur, Université de Versailles Saint-Quentin-en-Yvelines	Directeur de thèse
M. Daniele MICCIANCIO Professeur, University of California, San Diego	Rapporteur
M. Pascal PAILLIER CEO et senior security expert, CryptoExperts	Invité
M. Renaud SIRDEY Directeur de recherche, CEA LIST	Président du Jury
M. Damien STEHLÉ Professeur, École Normale Supérieure de Lyon	Rapporteur
Mme Vanessa VITSE Maître de conférences, Institut Fourier Grenoble	Examinatrice

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Vers l'efficacité et la sécurité du chiffrement homomorphe et du cloud computing

Mots clefs : chiffrement homomorphe, cloud computing, learning with errors, cryptologie, vote électronique, calcul multi-partite.

Résumé : Cette thèse étudie le chiffrement homomorphe, nouvelle famille de schémas de chiffrement qui permet de faire des calculs sur les messages chiffrés. Le chiffrement homomorphe compte un grand nombre d'applications pratiques : vote électronique, calculs sur des données sensibles, cloud computing, etc.. Dans la thèse, on propose un nouveau schéma de chiffrement homomorphe, basé sur la construction GSW et ses variantes, appelé TFHE. TFHE améliore soit la technique de bootstrapping, utilisée pour rafraîchir les chiffrés bruités, soit les calculs homomorphes non bootstrappés, en proposant des nouvelles techniques de packing des données et d'évaluation via automates pondérés. Le schéma a été implémenté pendant la thèse et il est disponible en open source. La thèse présente aussi des travaux complémentaires : la construction théorique d'un schéma de vote électronique post-quantique basé sur le chiffrement homomorphe, l'analyse de sécurité du chiffrement homomorphe dans le cas d'une implémentation dans le cloud, et une nouvelle solution pour le calcul sécurisé basée sur le calcul multi-partite.

Title : Towards efficient and secure Fully Homomorphic Encryption and cloud computing

Keywords : fully homomorphic encryption, cloud computing, learning with errors, cryptology, electronic voting, multi-party computation.

Abstract : This thesis studies fully homomorphic encryption (FHE), a new family of encryption schemes that allows to perform computations on encrypted data. FHE has a large number of applications: electronic voting, computations on sensitive data, cloud computing, etc.. In this thesis, we propose a new homomorphic scheme based on the GSW construction and its variants, that we call TFHE. TFHE improves both the bootstrapping technique, used to refresh noisy ciphertexts, and the homomorphic computations without bootstrapping, by presenting new packing techniques and an evaluation via weighted automata. The scheme has been implemented during the thesis and is available in open source. Furthermore, this thesis presents additional work: the theoretical construction of a post-quantum electronic-voting scheme based on homomorphic encryption, a security analysis of homomorphic encryption in a practical cloud implementation scenario, and a new solution for secure computing based on multi-party computation.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

Acknowledgements

T.b.a.

Contents

Chapter 1

Introduction

We live in the era of the cloud, an ambiguous entity able of storing and processing our data, accessible from everywhere in the world via internet connection, that materializes in a huge number of servers located somewhere on the planet. The cloud is incredibly practical: it is sufficient to have a portable device to access its agenda, photos, documents, contacts, emails, etc. Several cloud services are proposed by different companies and most of the time their products are "free". It is sufficient to create an account with a login and a password (and sometimes answer to a bunch of questions) and one can access all these services.

Of course, nothing comes for free, and the cloud is not an exception. Even if our data seems protected by a password that only we possess, the cloud is the real final recipient of all the information. Accessing this information gives the cloud a huge amount of very valuable details about ourselves. The cloud knows our tastes and our needs, and it can sell all this information to third parties (and gain a lot). The risk is that we might finally become the product to exploit.

In this scenario, it becomes important to find efficient solutions to protect private data, without losing at the same time the benefits brought by the cloud. A valuable solution in this context comes from fully homomorphic encryption.

Fully homomorphic encryption.

When it comes to protect data, cryptology is surely a good answer.
Already the ancient Greeks, with the scytale, and Romans, with Caesar's cipher, made use of cryptology, and after them, kings and queens, intellectuals and mainly armies protected their communications by using a huge variety of systems, more or less resistant. Until the 70's, all the cryptosystems were symmetric: the sender and the receiver share the same secret key and they use it to encrypt and decrypt messages. Asymmetric encryption was a real breakthrough. The recipient owns a pair of keys, one private that he keeps for himself, and one public known by

everyone. The two keys are related by mathematical relations, and the knowledge of the public key should not allow the adversary to retrieve the secret. Anyone who has the public key can send messages to the owner of the keys, but only their owner is able to decrypt those messages. The first asymmetric protocols were proposed by Diffie and Hellman in 1976 [DH76], Rivest, Shamir and Adleman in 1978 [RSA78] and ElGamal in 1985 [ElG85]. Nowadays, asymmetric protocols are mainly used to produce signatures or exchange symmetric keys. After the advent of public key constructions, cryptology stopped being used exclusively in the military context and became a tool shared (not always consciously) by all connected people on the earth.

In the cloud context, we can use standard cryptology to protect the data stored on the servers online. But if we want to search, modify or operate on such data, we must download it, decrypt it, transform it, encrypt it again and re-upload it on the cloud. Even if classical encryption solves the privacy problem, it is not enough.

In 1978, Rivest, Adleman and Dertouzos [RAD78] introduced a new concept of encryption schemes, called by them *privacy homomorphisms* and known nowadays as *homomorphic encryption schemes*, able to perform computations on encrypted data. However, they did not have an actual instantiation, and no solution was found for almost 30 years after this publication. To be precise, there existed schemes able to perform computations on encrypted data, but the allowed operations were limited. Indeed, only one kind of operation could be performed, or just a limited number of computations. The first valid fully homomorphic encryption scheme, able to potentially evaluate any function on encrypted data, was proposed in 2009 by Gentry [Gen09]. His scheme was impractical, but he proved that fully homomorphic encryption was possible. After this first construction, several schemes were proposed.

Fully homomorphic schemes have an enormous amount of applications, and one of them is cloud computing. In fact, not only we can encrypt our data on the cloud, but we can also ask the cloud to operate on this data: with fully homomorphic encryption the cloud is not able to extract any information and, sometimes, neither to understand which kind of operation is performed. With this new technology applied on the cloud, we may take advantage of the storage space and the computational power of the cloud, all by keeping our data safe and secret.

Besides cloud computing, homomorphic encryption can be used for electronic voting, for multi-party computation, to make operations on sensitive data (in the medical and financial domain as instance), for biometrics recognition, etc.

This large range of applications has made homomorphic encryption one of the most interesting subjects of study in cryptology of the last few years and, in the particular, the main subject of this PhD thesis.

This thesis.

At this point, new questions arise. Is it possible to make fully homomorphic encryption practical and usable for real world applications? Is it really safe to use fully homomorphic encryption in the cloud context? Are there any other options to perform computations on encrypted data other than homomorphic encryption?
It is in this context that this thesis started, and these are all the questions we aimed at solving with the work done during the past three years. This manuscript is a summary of the answers found, as well as all the new questions that arose in the attempt to find these solutions.

We start this manuscript with a state of the art on homomorphic encryption (Chapter 2), that summarizes the main solutions proposed in the literature. Chapters 3, 4 and 5 explain our improvements on homomorphic encryption, and present in detail our construction, called *TFHE*. These chapters include the results we published in [CGGI16a], [CGGI17a], and submitted as a long version to the Journal of Cryptology last year. With TFHE, we contribute to make homomorphic encryption more practical. Our results in [CGGI16b] won the Best Paper Award to the conference Asiacrypt 2016. However, there is a long road ahead before we can really affirm that homomorphic encryption is efficient for real world applications. The first part of Chapter 6 analyzes the semantic security of TFHE (as in [CGGI16a] and [CGGI17a]) and gives the guidelines for the choice of the parameters for our scheme, a difficult task in the homomorphic encryption context. The need, the wish and the curiosity to see our construction used in practice pushed us to implement it and test it for simple functions. This implementation (today a C/C++ open source library) is publicly available on GitHub [CGGI16d]. We describe our library, the tests we did and the results we obtained in the second part of Chapter 6. Then, a short conclusion summarizes the open questions and possible future works.
The manuscript includes three appendices, presenting some additional works. Appendix A is dedicated to the security of general homomorphic encryption schemes in the cloud context. This part summarizes the results we proposed in [CGG16]. In Appendix B, we present a more complex application of homomorphic encryption: an electronic voting scheme. We proposed this scheme in [CGGI16b]. Finally, Appendix C describes a recent result in multi-party computation, a different solution for secure computing, published in [BCG+18].

The chronological order in which the results were found is a bit different. When this thesis started, the first question we tried to answer was the one about the security of homomorphic encryption in the cloud context. This was a generic question. Just after that we focused on a specific family of homomorphic encryption schemes. We decided to analyze one of the most interesting fully homomorphic encryption schemes at the moment, i.e. the one proposed by Ducas and Micciancio in [DM15]. To better understand their construction and to better express its potential, we started by using

their scheme to produce a real world application. We were particularly interested in electronic voting and we designed the first post quantum electronic voting scheme. Our design was mainly theoretical and we wanted to see if it would run in practice. So we studied the scheme by [DM15] in detail and we tried to improve it. This work led to the design of TFHE. In the end we did not implement the electronic voting scheme, but it would be interesting to see what it looks like with the new TFHE improvements. The multi-party computation project came later, as a valuable secure cloud computing alternative.

What we learned.

Now that this thesis comes to its end, it is time to take stock and see if we were able to give proper answers to the questions we had at the beginning. The answers, that we summarize here, seem quite promising for the future of FHE schemes.

Fully homomorphic encryption is surely a powerful tool, and it is a continuous surprise. When Rivest, Adleman and Dertouzos [RAD78] first had the idea, such schemes seemed just utopic. In 2009, when Gentry showed the first construction, the doubts on the existence were replaced by the doubts on the practicality, since memory requirements and execution timings of Gentry's construction were unachievable (initially, none a lifetime was sufficient to compute a bootstrapping). After that, new constructions demonstrated that unachievable may become achievable, even if not so practical (one bootstrapping in a few minutes). The work we made in the last three years and the results that other researchers produced in this domain proved that homomorphic encryption may be not so impractical, and can be used (at least) for small applications (one bootstrapping is in the order of the milliseconds). In less than 10 years, the confidence on this new technology has increased enormously, and all these results made us believe that soon homomorphic encryption could really be used for real world applications with great results. Furthermore, the schemes proposed are often post-quantum secure, which contributes to add a huge interest in them.

Of course, cloud computing could be one of the real world applications. The potential of homomorphic encryption on the cloud is huge, as well as is the danger. We should imagine the cloud as an enormous arena: our most sensitive data are put in the center of the arena and a huge audience can look at them from the bleachers. The audience may be good, honest but curious, or even malicious. The proven semantic security of homomorphic encryption let us think that the schemes are safe, but it is known that the constructions are malleable. The malleability (i.e. the possibility to transform a ciphertext into another valid ciphertext, and then encrypt a message related to the original one) is the main strength of homomorphic encryption, but it was in general considered as a weakness in cryptology. Many attacks have been made on other cryptosystems because of this vulnerability: the

most famous are fault attacks. Homomorphic encryption schemes are not immune, and a cloud employment may give adversaries the possibility to retrieve many bits of information. There exist countermeasures to this kind of attacks (even if sometimes too expensive to set up), and some small precautions may prevent the loss of information. It is fundamental to be aware of these threats, in particular in a context such as that of the cloud.

It may also be interesting to search for other solutions for secure and private computations. There are already multiple examples, such as multi-party computation, obfuscation, functional encryption, multilinear maps, etc. Many of them may be realized by employing fully homomorphic encryption, but their cost would be too expensive. Better solutions have already been proposed and the research on these domains continues with the same commitment as in the homomorphic encryption domain. We started to investigate only the multi-party computation domain and it would be interesting to go deeper on the subject.
There is not an ideal solution for secure computing, all of the solutions have pros and cons, and a large variety of applications for what they are better suited than the other solutions. Having good constructions for all of these domains would make secure computing more feasible and usable from a larger number of people.

This manuscript is the summary of all the results obtained during these last three years of work. We try to clarify all the complicated and more technical points by using a lot of images and schemes. We guide the reader on this path by regularly summarizing what we did and what remains to do.
We hope that the reading of this manuscript is as pleasant as possible. We hope that the "story" we tell could lighten the doubts on the functionality and the great potential of homomorphic encryption and cloud computing. We finally hope that the reading will induce reflection, and maybe become the starting point for new ideas, as well as all the cited references did for us.

Chapter 2

State of the art on Homomorphic Encryption

By homomorphic encryption, we mean the family of encryption schemes being able to perform homomorphic operations on ciphertexts, without decrypting them. The first time someone introduced such a notion goes back to 1978, when Rivest, Adleman and Dertouzos [RAD78] described the *privacy homomorphisms*.
An homomorphic encryption scheme, as traditional encryption schemes, has a *key generation*, an *encryption* and a *decryption* algorithm, and it has an additional algorithm called *evaluation*.

- **Key generation**: given a security parameter λ, the keys (secret and, if necessary, public) are generated.
- **Decryption**: given a valid message space $\mathcal{M}$ and a valid ciphertext space $\mathcal{C}$, the decryption algorithm takes in input a ciphertext $c \in \mathcal{C}$ and a secret key, and outputs a message $m \in \mathcal{M}$.
- **Encryption**: the encryption algorithm takes in input a message $m \in \mathcal{M}$ and a secret (or public, depending on the scheme) key and outputs a ciphertext $c \in \mathcal{C}$. Encryption is randomized, which means that the same message can be encrypted in different ways by using the same key. The correctness condition imposes that every ciphertext decrypts in the original message.
- **Evaluation**: the evaluation algorithm is used to evaluate (homomorphically) functions or Boolean circuits over ciphertexts. It basically takes in input the description of a function $\phi : \mathcal{M}^k \to \mathcal{M}$ (with $k > 1$ being an integer) and a list of k ciphertexts $c_1, \ldots, c_k \in \mathcal{C}$ corresponding to messages $m_1, \ldots, m_k \in \mathcal{M}$ respectively, and outputs a new ciphertext $c \in \mathcal{C}$ that decrypts in $\phi(m_1, \ldots, m_k)$.

Some schemes possess homomorphic properties, but they can perform just additions, or just multiplications. Known examples are RSA [RSA78], ElGamal [ElG85] and

Paillier's [Pai99] cryptosystems. Some other schemes are homomorphic with respect to both operations, but they are able to perform these operations only a limited number of times. An example is the scheme of Boneh, Goh and Nissim [BGN05], on which it is possible to do unlimited additions with a unique final multiplication (based on elliptic curves and pairings). Such schemes are generally known as *Somewhat (or Partially) Homomorphic*.
Finding a scheme able to evaluate any function, and not just a limited set of operations, has not been an easy task. The first solution has been proposed by Gentry in 2009 [Gen09]. His scheme is based on ideal lattices and it is (potentially) able to evaluate any function. The scheme proposed by Gentry uses noisy ciphertexts, i.e. ciphertexts containing a certain amount of noise for security reasons. The noise grows after every evaluation: if this growth is not controlled, it can lead to incorrect decryption. In order to deal with this problem, Gentry introduced a technique, called *bootstrapping*, that is used to reduce periodically the noise and make the scheme *Fully Homomorphic*. It consists in evaluating the decryption circuit homomorphically by using an encryption of the secret key. More details are given in the next section.

Gentry's scheme is not used today because it is impractical and broken, but the bootstrapping technique is largely employed. The most promising schemes rely on two lattice-based problems:

- **Approximate GCD** problem: described in 2001 by Howgrave-Graham [HG01]. Some schemes based on this problem are [vDGHV10], [CMNT11], [CNT12], [CLT14], [CS15].
- **Learning With Errors** (LWE) problem: described in 2005 by Regev [Reg05], and its ring variants, described in 2009 by Stehlé, Steinfeld, Tanaka and Xagawa [SSTX09] and in 2010 by Lyubashevsky, Peikert and Regev [LPR10]. Two principal families of homomorphic encryption schemes are based on the LWE problem: the BGV family [BGV12], [BV11a], [BV11b], [BV14a], [Bra12], [FV12], [BV14b], and the GSW family [GSW13], [AP14], [BR15], [DM15], [CGGI16a], [CGGI17a].

A reduction from LWE to Approximate-GCD is described in [CS15]. More details on these schemes are given in next sections.
Unfortunately, bootstrapping still is the most expensive part of the entire homomorphic evaluations. It is for that reason that a lot of the work done in the homomorphic encryption domain aims at improving this technique. Some other works, instead, try to avoid bootstrapping. In fact, for some applications, the function to be evaluated is known in advance. In these cases, it is generally possible to just adapt the parameters of the scheme in order to do the evaluation without the needing of bootstrap. Of course, the size of the parameters is proportional to the depth of the function. Thus, we distinguish two cases:

- **Fully Homomorphic Encryption**: once the parameters are fixed, it is possible to correctly evaluate any function;
- **Leveled Homomorphic Encryption**: once the function is fixed, it is possible to find a set of parameters for the scheme in order to correctly evaluate the function.

In all the existing homomorphic schemes, the evaluation of an homomorphic operation increases the noise in ciphertexts. Generally, the multiplications are more costly than additions in terms of noise. When it comes to the evaluation of a function in leveled mode, it is important to know its multiplicative depth to predict how big have to be the parameters to support the evaluation. The depth often corresponds to the number of *levels* that can be supported by the chosen parameter set.
The main scheme presented in this manuscript gives a different interpretation of the levels and does not always measure the depth of a circuit/function as depending on the multiplicative depth. More details about this subject are given in Chapters 4 and 5.

In the rest of the manuscript, we use the notations SHE, FHE and LHE for Somewhat, Fully and Leveled homomorphic encryption respectively, and we often abbreviate homomorphic encryption by HE.

Symmetric or asymmetric. Homomorphic encryption schemes can be symmetric or asymmetric. In cryptology, it has been a huge challenge to find asymmetric schemes (first public key solutions only appeared in the seventies). Obtaining a symmetric encryption scheme from an asymmetric construction is easy: just keep all the keys secret. The inverse is nearly impossible in general.
With homomorphic encryption schemes, this is quite simple. In fact, as shown in [Rot10], a binary symmetric key homomorphic encryption scheme can be made asymmetric by publishing a list of fresh[1] encryptions of 0 and 1. If the scheme allows trivial ciphertexts or it supports the evaluation of constant functions, then it is sufficient to publish just a list of fresh encryptions of 0.
The main scheme described in this manuscript is presented in its symmetric version. In Appendix B, we present some applications where a public key version is needed.

Applications. Homomorphic encryption is particularly interesting for the enormous quantity of practical applications it offers. We propose just a few examples.

1. **Electronic voting**: considered as the electronic solution that will replace paper voting, it is one of the most interesting applications of HE. Several solutions have already been proposed. The ballots (i.e. the "vote containers"

[1]Shortly, a *fresh* ciphertext is a ciphertext that has just been generated and does not have already been used in a homomorphic evaluation.

of every voter) can be encrypted and summed homomorphically in order to compute the final result of an election. A system of trusted authorities reveals the decryption of the final result. In Appendix B we give more details about a homomorphic based e-voting scheme.

2. **Multi-party computation (MPC)**: consists in a distributed computation between multiple participants, not trusting each other. Everyone owns a part of the data or the function to be computed. The goal is that every participant at the end is able to compute the result (or a share of the result) without knowing the entire set of data. FHE is one of the possible solutions to this problem.

3. **Outsourced computations**: consists in asking to a third entity to perform computations on personal data. In the era of the cloud, the privacy of the data uploaded on public servers is a crucial subject. If we want to hide personal information, encryption becomes necessary. And as homomorphic encryption allows to perform computations on ciphertexts, the clients can take advantage of the storing and computational power of cloud servers, while maintaining their data private.

4. **Computations on sensitive data**: FHE found a large set of applications when sensitive data have to be treated. Examples are data coming from medical centers, hospitals, biometrics data or even financial data. With FHE it is possible to compute statistical balances, do researches, compute a match, study a particular phenomenon, while keeping the privacy of patients or clients safe at the same time.

5. **Circuit privacy**: not only the data, but also the function to be evaluated can be kept secret thanks to FHE. For instance, it is possible to hide the function by evaluating universal circuits or encrypted look-up tables. This may be an interesting property when a secret algorithm has to be evaluated, as instance in the financial domain. Even if the approach is completely different, this may be seen as a sort of *obfuscation* (make the program incomprehensible while keeping the functionality), or as a kind of two-party computation.

2.1 Gentry and DGHV: the first generation

The first FHE scheme proposed by Gentry [Gen09] and the solutions based on the approximate-GCD problem, in particular DGHV [vDGHV10], can be considered part of the *first generation* of homomorphic schemes.
The scheme proposed by Gentry is based on ideal lattices. In this section, we do not describe his scheme, but the bootstrapping idea that he introduced for the first time and that is largely used in FHE schemes. To better understand why the

bootstrapping is so important, we start by describing the approximate-GCD problem and the DGHV scheme, which is the easiest HE existing scheme.

2.1.1 Approximate-GCD and the DGHV scheme

The approximate-GCD (approximate great common divisor) problem has been described in 2001 by Howgrave-Graham [HG01].
Let p be an odd positive integer. Given a list $x_1, \ldots, x_n$ of integers[2] of the form

$$x_i = p \cdot q_i + r_i,$$

with q_i random integers (possibly greater than p) and r_i small random integers (noises), for $i \in [\![1, n]\!]$. The approximate-GCD problem consists in recovering p (search problem). If the noises are equal to 0, the problem is trivial: the GCD can be easily found via Euclid's algorithm. But in the presence of noise, it becomes harder to find the value of p.
To understand why, we can see the approximate-GCD problem as a graduate rule. The long lines in the rule represent the exact multiples of p. If we want to take samples for the approximate-GCD distribution, we draw (red) lines near the long lines in the rule. Instead, if we take uniformly random samples (in green), they appear in random places on the rule. If someone asks to distinguish which one, between the green or the red lines, has been sampled from the approximate-GCD distribution, the answer is immediate (Figure 2.1).

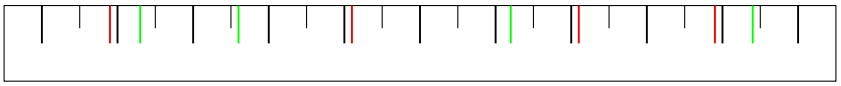

Figure 2.1: *The approximate-GCD distribution is represented by the rule: red lines are samples from this distribution, while green lines are sampled uniformly at random.*

But if the rule disappears (i.e. we do not know the secret), it is hard to distinguish (decisional problem) between the samples from the distribution and the random samples (Figure 2.2).

The DGHV scheme

In 2010, van Dijk, Gentry, Halevi and Vaikuntanathan, proposed in [vDGHV10] a scheme based on the approximate-GCD problem, that we call DGHV, after the authors names. The scheme is very easy to describe and it is homomorphic with respect to both addition and multiplication.

[2] In [HG01] $n = 2$, but the problem can be generalized for arbitrarily many samples.

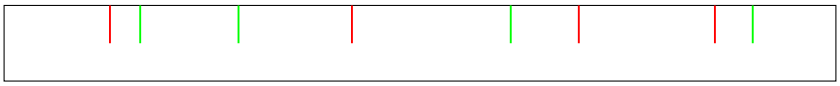

Figure 2.2: *The approximate-GCD distribution is hidden: red lines are samples from this distribution, while green lines are taken uniformly at random. They all appear as random samples.*

- **Key generation**: let λ be a fixed security parameter. The secret key is chosen as an odd random η-bit integer p (with $\eta = \tilde{O}(\lambda^2)$).
- **Encryption**: the message space $\mathcal{M}$ is the binary space $\mathbb{B}$. To encrypt a message $\mu \in \mathbb{B}$, one choses a random integer $q \in [\![0, 2^\gamma/p[\![$ (with $\gamma = \tilde{O}(\lambda^5)$) and a small random integer $r \in]\!] - 2^\rho, 2^\rho [\![$ (with $\rho = \lambda$). The encryption of μ is

$$c = p \cdot q + 2 \cdot r + \mu.$$

- **Decryption**: the decryption consists in reducing the ciphertext modulo p (in the zero centered interval), then modulo 2. Without knowing p, to retrieve the message m is like trying to *find a needle in a haystack*, without knowing if the needle is even there or not.
- **Evaluation**: Let's take two encryptions c_1, c_2 of two messages $m_1, m_2 \in \mathbb{B}$ respectively. If we add or multiply c_1 and c_2, we obtain two new ciphertexts encrypting the sum or the product of m_1 and m_2:

$$c_+ = c_1 + c_2 = p \cdot (q_1 + q_2) + 2 \cdot (r_1 + r_2) + (m_1 + m_2) \tag{2.1}$$

$$\begin{aligned} c_\times = c_1 \cdot c_2 = p \cdot (q_1 q_2 + 2q_1 r_2 + 2q_2 r_1 + q_1 m_2 + q_2 m_1) \\ + 2 \cdot (2r_1 r_2 + r_1 m_2 + r_2 m_1) + (m_1 m_2). \end{aligned} \tag{2.2}$$

Observe the noise part (Equations (2.1) and (2.2)) produced during the evaluation phase, i.e. the quantity multiplied by 2: it has grown in both cases. During the evaluation of the addition (Equation (2.1)) it has doubled. During the evaluation of the multiplication (Equation (2.2)) it has approximately doubled the square of the original values. The operations remain correct till the noise part is smaller than $p/2$, otherwise decryption may give incorrect results. It is for that reason that a noise reduction technique is needed (or bigger parameters in order to support a leveled evaluation).

Improvements. This scheme has been improved in 2011 by Coron, Mandal, Naccache and Tibouchi [CMNT11], who reduce the public key size and describe the first implementation of the scheme. In 2012 Coron, Naccache and Tibouchi [CNT12]

propose a key compression technique, by keeping the same level of security, and by showing how to adapt [BGV12] techniques, described in next sections, in the DGHV context. In 2013, Cheon et al. [CCK+13] show how to do batching, i.e. process several messages at the same time in an evaluation by packing them in a single ciphertext. The scheme has been generalized for a non-binary message space in 2015 by Nuida and Kurosawa [NK15]. They show that it is possible to encrypt messages in $\mathbb{Z}/P\mathbb{Z}$ for any P prime integer. Further improvements are presented in [CS15]. DGHV can be used in both leveled or bootstrapped mode.

2.1.2 Gentry's bootstrapping

Bootstrapping is the technique introduced by Gentry [Gen09] in order to reduce the noise in ciphertexts. In fact, after evaluating operations homomorphically, the noise increases (as shown for instance in the previous section) and, if not controlled, it could cause some decryption problems.
Imagine the practical scenario of a client storing his data on a cloud server, and asking the cloud to perform some computations on these data. In order to hide the information, the client encrypts his data with homomorphic encryption and asks the cloud to perform the computations on the ciphertexts. This is possible thanks to the properties of HE schemes. But if during the computations the noise grows too much, the result of the evaluation can become incorrect. The cloud could of course periodically send the ciphertexts to the client, this latter could decrypt and (freshly) re-encrypt them to let the cloud continue the computations correctly. Nevertheless, this is absolutely impractical. If the cloud was able to refresh the noisy ciphertexts itself without the constant intervention of the client, it would be much better.

Bootstrapping solves this problem. The easiest and trivial way to suppress the noise is to decrypt the ciphertext. But the client cannot reveal his secret key. Instead, he gives to the cloud an encryption of his secret key and asks him to homomorphically decrypt the noisy ciphertexts periodically.
Figure 2.3 illustrates the original bootstrapping idea. We start with an initial noisy ciphertext encrypting the message μ with a secret key sk_1 (straight line boxes). In order to reduce the noise, we homomorphically evaluate the decryption circuit. To do that, we add a supplementary layer of encryption (dashed line boxes), by using the public key corresponding to the secret sk_2. We also provide an encryption of the first secret key sk_1 with respect to the secret key of the second encryption scheme sk_2, and we call it *bootstrapping key* (BK). Then, inside the second layer of encryption, we evaluate the decryption circuit of the first scheme. As a result, the first layer of encryption disappears (and its noise too), and the message is still encrypted, but this time only with the second secret key sk_2. The noise in the output ciphertext is independent of the input noise. In particular, if the input noise was

large, bootstrapping reduces the noise to a fixed amount, so it is possible to continue evaluating operations. Every time the noise reaches a critical level, the bootstrapping is repeated.

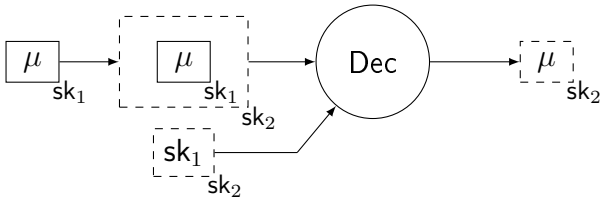

Figure 2.3: Original bootstrapping idea - *Straight-line boxes represent the first encryption layer, with secret key* sk_1*, while the dashed-line boxes represent the second encryption layer, with secret key* sk_2*. In order to reduce the noise, the decryption circuit with respect to the key* sk_1 *is evaluated homomorphically. Thanks to the second layer of homomorphic encryption, no information about the message or the secret key* sk_1 *is revealed. The result is a new ciphertext encrypting the initial message, but with less noise. The encryption of* sk_1 *with the new key* sk_2 *is called bootstrapping key* BK*.*

After the construction proposed by Gentry, bootstrapping has been simplified. In particular, for the (binary) schemes allowing trivial ciphertexts, the additional second layer added to the first encryption of the message is not necessary. It is possible to directly evaluate the decryption circuit on the bits of the noisy ciphertext, by using the bootstrapping key. The result is still a new encryption of the message μ with the second secret key sk_2 and with less noise. Roughly speaking, during the evaluation of the decryption circuit, the second layer of encryption propagates on μ, thanks to the way the bootstrapping key is constructed. Figure 2.4 illustrates the idea we just described.

So, if an HE scheme is able to homomorphically evaluate its own decryption function plus another small operation (at least), and if publishing the bootstrapping keys does not compromise the security of the scheme, then it is bootstrappable and it becomes fully homomorphic.

2.2 Learning with errors: the second and third generation

In 2005, Regev [Reg05] described the Learning With Errors (LWE) problem. Nowadays this problem and its variants are largely used in lattice based cryptology, mainly in homomorphic encryption constructions. The problem is considered hard and it is largely used in most of HE constructions. We give more details about its security

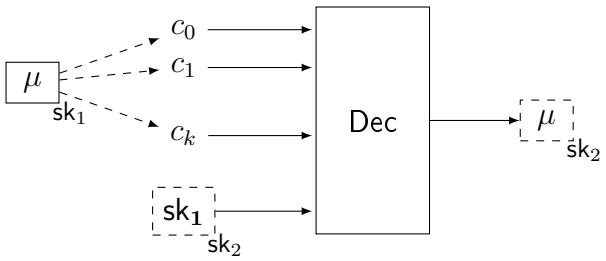

Figure 2.4: *Many improvements to Gentry's bootstrapping principle consisted in removing the re-encryption phase by delaying as much as possible the operations depending on the secret key, and to make them as linear as possible. Instead of re-encrypting the noisy ciphertexts with an additional layer of homomorphic encryption, new bootstrappings evaluate the decryption circuit directly on the bits of the noisy ciphertext by using the encryption of the secret key (i.e. the bootstrapping key).*

in Chapter 6. In this section we describe LWE and its variants. Then we rapidly summarize the main homomorphic constructions based on LWE.

Definition 2.2.1 (LWE). *Let λ be a security parameter (all the following parameters depend on λ). Let n be an integer (called dimension) and q be an integer (called modulus). Let χ be a probability distribution over $\mathbb{Z}$. Sample a uniformly random secret vector $\mathbf{s}$ from $\mathbb{Z}_q^n$. We define the* LWE *distribution, and we call it $\mathcal{D}_{\mathbf{s},\chi}^{\mathsf{LWE}}$, as the distributions that samples pairs $(\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$, such that $\mathbf{a}$ is chosen uniformly at random from $\mathbb{Z}_q^n$ and $b = \mathbf{a} \cdot \mathbf{s} + e$, with $e \leftarrow \chi$. Then, we distinguish two problems.*

- *Decision problem: given arbitrarily many samples $(\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$, decide if they come from the distribution $\mathcal{D}_{\mathbf{s},\chi}^{\mathsf{LWE}}$ or from a uniform distribution over $\mathbb{Z}_q^{n+1}$, for a fixed $\mathbf{s} \leftarrow \mathbb{Z}_q^n$.*
- *Search problem: given arbitrarily many samples $(\mathbf{a}, b)$ from the distribution $\mathcal{D}_{\mathbf{s},\chi}^{\mathsf{LWE}}$, find $\mathbf{s}$.*

The LWE search and decision problems are reducible one to each other, and their average case is asymptotically as hard as worst-case lattice problems, as shown by Regev in [Reg05]. In practice, both problems are intractable: their hardness depends on the choice of parameters. The same observation can be done for the variants of LWE described in the following. We give more details about the subject in Chapter 6. In 2009, Stehlé, Steinfeld, Tanaka and Xagawa [SSTX09] and in 2010, Lyubashevsky, Peikert and Regev [LPR10] proposed a variant of LWE over the rings, called RingLWE.

Definition 2.2.2 (RingLWE). *Let λ be a security parameter (all the following parameters depend on λ). Let N be an integer (size of the ring, generally a power of 2) and define $R = \mathbb{Z}[X]/(f_N(X))$, were $f_N(X)$ is a monic polynomial, irreducible over*

$\mathbb{Z}[X]$, *of degree* N *(generally* $f_N(X) = X^N + 1$*). Let* q *be a large integer (called modulus) and* $R_q = R/qR$*. Let* χ *be a probability distribution over* R*. Sample uniformly at random a secret polynomial* s *from* R_q*. We define the* RingLWE *distribution, and we call it* $\mathcal{D}_{s,\chi}^{\mathsf{RingLWE}}$*, as the distribution that samples pairs* $(a, b) \in R_q^2$*, such that* a *is chosen uniformly at random from* R_q *and* $b = a \cdot s + e$*, with* $e \leftarrow \chi$*. Then, we distinguish two problems.*

- *Decision problem: given arbitrarily many samples* $(a, b) \in R_q^2$*, decide if they come from the distribution* $\mathcal{D}_{s,\chi}^{\mathsf{RingLWE}}$ *or from a uniform distribution over* R_q^2*, for a fixed* $s \leftarrow R_q$*.*
- *Search problem: given arbitrarily many samples* (a, b) *from the distribution* $\mathcal{D}_{s,\chi}^{\mathsf{RingLWE}}$*, find* s*.*

The definition we give is more similar to the IdealLWE problem described in [SSTX09] and later renamed PolynomialLWE. In the PolynomialLWE, a single polynomial a is sampled: the others are obtained by multiplying a by X^i for $0 < i < N$ and reducing the result modulo $f_N(X)$. The definition of [LPR10] is generalized for the ring of integers in any number field. The two versions match if the quotient polynomial is chosen as a power of 2 cyclotomic, as in our case.

The ring version of LWE seems more attractive than the original one, mainly for efficiency reasons. One sample in RingLWE contains the same amount of information as N LWE samples. As a consequence, the size of ciphertexts and public keys is reduced. Additionally, the operations with polynomials can be very fast, if a FFT technique is used. On the other side, the ring structure may bring more weaknesses from the security point of view.

In [BGV12], a generalized definition, including both the LWE and the RingLWE instances, is given and it is called the general LWE problem, and noted GLWE. In the literature, GLWE is often called ModuleLWE [LS15].

Definition 2.2.3 (GLWE (as in [BGV12]))**.** *Let* λ *be a security parameter (all the following parameters depend on* λ*). Let* N *be a power of* 2 *(size of the ring) and define* $R = \mathbb{Z}[X]/(X^N + 1)$*: if* $N = 1$*,* $R = \mathbb{Z}$*. Let* q *be a large integer (called modulus) and* $R_q = R/qR$*. Let* k *be an integer and let* χ *be a probability distribution over* R*. Sample uniformly at random a secret polynomial* $\mathbf{s}$ *from* R_q^k*. We define the* GLWE *distribution, and we call it* $\mathcal{D}_{\mathbf{s},\chi,N,k}^{\mathsf{GLWE}}$*, as the distributions that samples couples* $(\mathbf{a}, b) \in R_q^{k+1}$*, such that* $\mathbf{a}$ *is chosen uniformly at random from* R_q^k *and* $b = \mathbf{a} \cdot \mathbf{s} + e$*, with* $e \leftarrow \chi$*. Then, we distinguish two problems.*

- *Decision problem: given arbitrarily many samples* $(\mathbf{a}, b) \in R_q^{k+1}$*, decide if they come from the distribution* $\mathcal{D}_{\mathbf{s},\chi,N,k}^{\mathsf{GLWE}}$ *or from a uniform distribution over* R_q^{k+1}*, for a fixed* $\mathbf{s} \leftarrow R_q^k$*.*

- *Search problem: given arbitrarily many samples* $(\mathbf{a}, b)$ *from the distribution* $\mathcal{D}^{\mathsf{GLWE}}_{\mathbf{s},\chi,N,k}$*, find* $\mathbf{s}$*.*

If $N = 1$, the GLWE problem reduces to the LWE instance. If instead $k = 1$, the GLWE problem reduces to the RingLWE instance.
The TFHE scheme, described in the main chapters of this manuscript, is based on the LWE problem. In order to make the description more general, we give in Chapter 3 another generalized definition of the LWE problem, which is similar to the GLWE definition.

2.2.1 Second Generation: BGV

One of the first homomorphic encryption schemes based on the LWE problem and on its RingLWE variant is the scheme proposed by Brakerski, Gentry and Vaikuntanathan in 2012 [BGV12], and shortly called BGV after the authors names. It is inspired and uses techniques from the homomorphic schemes proposed in [BV11a] and [BV11b]. BGV is a LWE/RingLWE-based LHE scheme, able to evaluate L-leveled arithmetic circuits.
In order to keep the noise constant, they use a technique called *modulus switching*. The modulus switching reduces the noise magnitude and the modulus size at the same time, but does not decrease their ratio. To be able to perform the modulus switching L times, i.e. to evaluate a L-leveled circuit, they initially set L different decreasing modulus $q_{L-1}, \ldots, q_0$ and switch from one to another by keeping the noise essentially constant. They also use a *key-switching* technique, in order to switch from one key to another. This is crucial during the evaluation of multiplications.
Bootstrapping could be used, but it is seen just as an optimization, and it does not impact the L-level evaluation. With the ring version of BGV, it is possible to do batching, i.e. to pack several messages in a single ciphertext and evaluate the operations on all the messages at the same time (with the same efficiency as if the same operations were evaluated on a single input).

The BGV scheme

In this section, we describe the generic BGV construction, based on the GLWE problem. This way, both the scalar and the ring versions are summarized.
Let λ be a security parameter. The dimension of all the parameters for this encryption scheme depend on λ. Let n be an integer, called the dimension and let k and N (ring dimension, used to define $R = \mathbb{Z}[X]/(X^N + 1)$) be two integers such that the product kN is equal to n (if $N = 1$ and k large we have the LWE instance, if N large and $k = 1$ we have the RingLWE instance). Let $m = \lceil (2k + 1) \log q \rceil$ be an integer. Let q be an integer, called the modulus, R_q denotes R/qR and R_2 denotes $R/2R$ (plaintext space)[3]. Let χ be a probability distribution over R (that samples

[3]The plaintext space can be generalized for a larger modulus p, instead of 2.

small coefficients). The BGV scheme is composed by the following algorithms.

- **Secret key generation**: sample a random secret vector $\mathbf{s} \leftarrow \chi^k$. Then the secret key of the scheme is

$$\mathsf{sk} = (1, -\mathbf{s}) \in R_q^{k+1}.$$

  The LWE problem is still hard, even if the secret is sampled from the χ distribution [ACPS09].

- **Public key generation**: given the secret key $\mathsf{sk} = (1, \mathbf{s})$, we generate uniformly at random a matrix $A \leftarrow R_q^{m \times k}$ and a random vector of errors/noises $\mathbf{e} \leftarrow \chi^m$. Then we set $\mathbf{b} = A\mathbf{s} + 2\mathbf{e}$. The public key is

$$\mathsf{PK} = [\mathbf{b}|A] \in R_q^{m \times (k+1)}.$$

  The public key generation also includes the generation of eventual evaluation keys, such as key-switching keys and bootstrapping keys.

- **Encryption**: let $\mu \in R_2$ be a message. To encrypt it we sample a random vector $\mathbf{r} \in R_2^m$. The ciphertext is

$$\mathbf{c} = (\mu, 0, \ldots, 0) + \mathsf{PK}^T \mathbf{r} \in R_q^{k+1}.$$

- **Decryption**: the decryption consists in a scalar product between the ciphertext $\mathbf{c}$ and the secret key sk reduced modulo q and then modulo 2.

- **Evaluation of addition**: the evaluation of the homomorphic addition is performed as an addition of two ciphertexts.

- **Evaluation of multiplication**: the evaluation of the multiplication consists in a multiplication of two ciphertexts, but it changes the nature of the output ciphertext. In fact, this latter results to be encrypted with respect to the tensor product of the secret key sk with itself (the key has changed). To supply to this, a *relinearization* process (using key switching) is needed.

- **Key switching**: used to switch from a key to another. Let $\mathbf{c}$ be a ciphertext encrypted with the public key PK, corresponding to the secret sk. The key switching produces a new ciphertext $\mathbf{c}'$ encrypting the same message, with respect to a secret sk'. To do that, it makes use of a *key-switching key*, noted KS, which is an encryption of the secret key sk (times some powers of 2) with the public key PK'. Then, the key switching consists in a product between the vector containing the bit decomposition of $\mathbf{c}$ times KS. All the secret keys of the scheme and the key-switching keys are generated at the beginning, during the key generation phase.

- **Modulus switching**: used to switch from a level to another and to keep the noise constant. It takes in input a ciphertext $\mathbf{c}$ with large modulus q and outputs a new ciphertext $\mathbf{c}'$ encrypting the same message, with a smaller modulus q' and smaller noise. The procedure consists in multiplying $\mathbf{c}$ times q'/q and to round the result to the nearest integer polynomial such that $\mathbf{c} = \mathbf{c}' \mod 2$. The modulus levels are generated at the beginning and their number corresponds to the multiplicative depth of the function to be evaluated.

Variants. There exists a *scale invariant* version of the BGV scheme, described in [Bra12] and [FV12]. A homomorphic scheme is said to be scale invariant if its properties depend only on the ratio between the modulus q and the initial noise level, not on their absolute value [Bra12]. In the scale invariant scheme, the message is multiplied by a constant factor (rounded ratio of the modulus q and the plaintext modulus) before encryption. In scale invariant schemes, the messages are encoded in the most significant bits (instead of the least significant ones, as done by Regev in 2005) and the noise analysis is simpler.
Another BGV-based construction that is worth mentioning is HEAAN (Homomorphic Encryption for Arithmetic of Approximate Numbers). This new promising construction has been presented by Cheon et al. in 2017 [CKKS17] and bootstrapped in 2018 [CHK+18]. The scheme supports approximate addition and multiplication of encrypted messages with a predetermined precision, it allows batched operations and a fast bootstrapping of packed ciphertexts. The scheme has been implemented and it is available in open-source.

2.2.2 Third Generation: GSW

In 2013 Gentry, Sahai and Waters propose a new FHE scheme, called GSW after their names. GSW is the principal representative of the third generation of FHE schemes. It is a LWE-based scheme and uses what the authors call the *approximate eigenvector method* to perform multiplications: the ciphertexts are matrices, the secret key is an approximate eigenvector of the ciphertext matrix and the message is the corresponding eigenvalue.
The GSW scheme is originally described as a LWE based scheme [GSW13], but it can be presented in its ring version. To follow the same footstep as in previous section, we describe GSW with respect to the generalized problem GLWE.
As for the BGV scheme, GSW can be used in both leveled and bootstrapped mode.

The GSW scheme

Let λ be a security parameter. As for BGV, all the parameters for the GSW encryption scheme depend on λ.
Again, let n an integer dimension and let k and N be two integers such that the product kN is equal to n. As $R = \mathbb{Z}[X]/(X^N + 1)$, if $N = 1$ and k is large we have

the LWE instance, if N is large and $k = 1$ we have the RingLWE instance. Let q the integer modulus (possibly a power of 2), R_q denotes R/qR. Let $\ell = \lfloor \log q \rfloor + 1$ and $m = (k+1)\ell$ be two integers. Let χ be a probability distribution over R_q. In the original description of the scheme the authors define four functions they use in different parts of the construction: BitDecomp (an exact decomposition in base 2), $\mathsf{BitDecomp}^{-1}$ (the inverse of BitDecomp), Flatten (composition of BitDecomp and $\mathsf{BitDecomp}^{-1}$) and Powersof2 (multiplication by different powers of 2). Instead we describe the scheme by using the gadget approach, as in [AP14].
The *gadget matrix* H is defined as the block diagonal matrix from Equation (2.3), where the coefficients in the diagonal blocks are increasing powers of 2:

$$H = \left(\begin{array}{c|c|c} 2^0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 2^{\ell-1} & \dots & 0 \\ \hline \vdots & \ddots & \vdots \\ \hline 0 & \dots & 2^0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 2^{\ell-1} \end{array}\right) \in \mathcal{M}_{h\ell,h}(R_q). \tag{2.3}$$

The BGV's operation Powersof2 on a vector $\mathbf{a} \in R_q^h$ corresponds to multiply $H \cdot \mathbf{a}$. The decomposition of a vector with respect to the gadget matrix is the BitDecomp operation. We note the decomposition with respect to H as Dec_H. Since the elements in the columns of the gadget matrix are superincreasing, this operation is efficient. If a decomposed vector $\mathbf{a}'$ is multiplied on the left with the gadget matrix, the operation corresponds to the inverse of the decomposition, i.e. the $\mathsf{BitDecomp}^{-1}$ operation. This means that $H \cdot Dec_H(\mathbf{a}) = \mathbf{a}$. The gadget representation comes from [MP12] and [AP14], and it can be generalized for a gadget basis different from 2 (generally a power of 2).
In the following, h is equal to $k+1$. The GSW scheme is composed by the following algorithms.

- **Secret key generation**: sample a random secret vector $\mathbf{s} \leftarrow R_q^k$. Then the secret key of the scheme is

$$\mathsf{sk} = (1, -\mathbf{s}) \in R_q^{k+1}.$$

- **Public key generation**: given the secret key $\mathsf{sk} = (1, \mathbf{s})$, we generate uniformly at random a matrix $A \leftarrow R_q^{m \times k}$ and a random vector of errors/noises $\mathbf{e} \leftarrow \chi^m$. Then we set $\mathbf{b} = A\mathbf{s} + \mathbf{e}$. The public key is

$$\mathsf{PK} = [\mathbf{b}|A] \in R_q^{m \times (k+1)}.$$

  The public key generation also includes the generation of eventual evaluation keys, such as key-switching keys and bootstrapping keys.

- **Encryption**: let $\mu \in R_q$ be a message. To encrypt it we sample a random matrix $R \in \mathbb{B}^{m \times m}$. The ciphertext is

$$C = R \cdot \mathsf{PK} + \mu \cdot H \in R_q^{k+1}.$$

- **Decryption**: let $\mathbf{v} = \mathsf{Powersof2}(\mathsf{sk})$. The first ℓ elements of $\mathbf{v}$ are equals to $1, 2, \ldots, 2^{\ell-1}$. Let $i \in [\![0, \ell-1]\!]$. The decryption of the ciphertext C is computed as

$$\mu' = \left\lfloor \frac{C_i \cdot \mathbf{v}}{v_i} \right\rceil.$$

  This algorithm works for small messages. For general decryption, the authors propose to use an idea from [MP12]: each one of the bits of the message is recovered from one of the first ℓ elements of the product $C \cdot \mathbf{v}$, starting from the least significant bits.

- **Evaluation of addition**: let C_1 and C_2 two GSW ciphertexts, the addition consists in a simple addition of the ciphertexts.

- **Evaluation of multiplication**: let C_1 and C_2 two GSW ciphertexts, the multiplication consists in multiplying the first ciphertext times the decomposition of the second one with respect to the gadget: $C_1 \cdot Dec_H(C_2)$.

The scheme can be bootstrapped. Further improvements are proposed in [AP14]: here the bootstrapping has almost quartic asymptotic complexity with respect to the the security parameter.
In 2015, Ducas and Micciancio [DM15] proposed a new fast bootstrapping with quasi-quadratic asymptotic complexity. Their bootstrapping (very fast compared to previous ones) is performed after every `NAND` gate evaluation.
Additional improvements to this scheme are proposed in [BR15], where the scheme of [DM15] is generalized to support a bigger message space (not just binary).
The TFHE construction described in the main chapters of this manuscript is a GSW based construction. Our construction is an improvement of the scheme and of the techniques proposed by [DM15].

Additional homomorphic constructions based on NTRU

Additional homomorphic encryption schemes are presented in the literature. One family that deserves to be mentioned is the one based on NTRU. NTRU is a lattice based cryptosystem that has been presented for the first time at the Crypto rump session in 1996 by Hoffstein, Pipher and Silverman [HPS98], and modified later by Stehlé and Steinfeld in [SS11] for security reasons.
Two known homomorphic encryption schemes based on NTRU are YASHE (Yet Another SHE), presented by Bos, Lauter, Loftus and Naehrig in [BLLN13], and LTV, presented by López-Alt, Tromer and Vaikuntanathan in [LTV12].

YASHE is scale invariant and its performances have been largely compared with the ones of the BGV scale invariant version FV ([LN14], [CS16]). LTV is a multi-key FHE scheme, principally presented as a valuable multi-party computation solution. Recent subfield lattice attacks by [ABD16] have compromised the asymptotic security for both YASHE and LTV schemes.

Chapter 3

LWE and GSW over the Torus

The homomorphic encryption scheme TFHE described in the main chapters of this manuscript is a GSW based scheme. The "T" letter in TFHE stands for the real torus, noted $\mathbb{T}$ and defined as the quotient $\mathbb{R}/\mathbb{Z}$, or simply as the reals modulo 1. But what has the torus to do with homomorphic encryption or with the GSW construction?
As we explained in Chapter 2, GSW is a LWE-based scheme. LWE is defined over $\mathbb{Z}_q$, but it can also be redefined over the torus. The idea is that integers modulo q can be re-scaled by dividing them by q and reducing modulo $q/q = 1$. This reduces the problem to a discrete subset of the real numbers between 0 and 1. Instead of working in a discrete subset, we can redefine the problem in the continuous interval $\mathbb{T}$. Working on the torus is quite practical: the scheme is scale invariant, the security analysis and the study of noise growth are simplified, and it allows to better abstract the problems with which we are dealing. Additionally, the torus has a nice $\mathbb{Z}$-module structure, that helps to clarify the complex set of morphisms, operations and more generally interactions in the plaintext space.
The torus structure has already been used in previous works on LWE: already Regev [Reg05] used it in its paper about LWE (the right hand side of the LWE samples is in the torus), and, for example, after him Cheon and Stehle [CS15] defined the scale invariant version of LWE completely over the torus.

In this chapter we give a generalized definition (similar to the GLWE definition) for the scale invariant LWE problem and we generalize it over $\mathbb{T}$. Then, by following this footstep, we redefine and re-formalize the GSW construction over the torus.

3.1 Preliminary notions

We start this chapter by fixing the notations we use in the rest of the manuscript and by rapidly recalling some basic concepts about modules, distributions, distances and norms.

Basic notations

We always denote the security parameter by λ. We note by $\mathbb{B}$ the set $\{0,1\}$ (without any structure), by $\mathbb{Z}$ the set of integers and by $\mathbb{T}$ the real Torus $\mathbb{R}/\mathbb{Z}$, i.e. the set of real numbers modulo 1.
We denote by $\mathbb{Z}_N[X]$ the ring of polynomials $\mathbb{Z}[X]/(X^N+1)$. $\mathbb{T}_N[X]$ denotes $\mathbb{R}[X]/(X^N+1) \mod 1$ and $\mathbb{B}_N[X]$ denotes the polynomials in $\mathbb{Z}_N[X]$ with binary coefficients.
Generally, N is taken as a power of 2, so X^N+1 is the $2N$-th cyclotomic polynomial[1]. Using a power of 2 becomes very practical when it comes to implementing the schemes.
We denote by E^p the set of vectors of dimension p with entries in E and by $\mathcal{M}_{p,q}(E)$ the set of $(p \times q)$-size matrices with elements in E.

3.1.1 Modules

The real torus has a $\mathbb{Z}$-module structure. Roughly speaking, $\mathbb{T}$ has a commutative group structure with respect to the addition, and it owns an external product with integer coefficients. In practice, the internal product $\times$ between torus elements is not defined (i.e. the torus is not a ring). Observe as instance, the product $0 \times \frac{1}{2} = 0$ and the product $1 \times \frac{1}{2} = \frac{1}{2}$, where 0, 1 and $\frac{1}{2}$ are seen as elements of $\mathbb{T}$. The two products should produce the same result in $\mathbb{T}$ since 0 and 1 are equivalent on the torus, but they do not. Instead, the external product $\cdot$ between an element of $\mathbb{Z}$ and an element in $\mathbb{T}$ is correctly defined and produces a torus element. To use the same example as before, $0 \cdot \frac{1}{2}$, where $0 \in \mathbb{Z}$ and $\frac{1}{2} \in \mathbb{T}$, is equal to $0 \in \mathbb{T}$, and $1 \cdot \frac{1}{2}$, where $1 \in \mathbb{Z}$ and $\frac{1}{2} \in \mathbb{T}$, is equal to $\frac{1}{2} \in \mathbb{T}$.
We give a proper definition of modulus.

Definition 3.1.1 (R-module)**.** *Let $(R,+,\times)$ be a commutative ring[2]. We say that a set M is a R-module when $(M,+)$ is an Abelian group, and when there exists an external operation $\cdot$ (product) which is bi-distributive and homogeneous. Namely, $\forall r,s \in R$ and $x,y \in M$ the following properties are satisfied:*

- *Unitary: $1_R \cdot x = x$, where 1_R denotes the unity of R,*
- *Left distributive: $(r+s)\cdot x = r\cdot x + s\cdot x$,*
- *Right distributive: $r\cdot(x+y) = r\cdot x + r\cdot y$,*
- *Homogeneous: $(r\times s)\cdot x = r\cdot(s\cdot x)$.*

[1] The n-th cyclotomic polynomial is the monic polynomial having as roots the primitive n-th roots of unity $e^{2\pi i k/n}$, with $0<k<n$ prime with n.

[2] If the ring is not commutative, we have to distinguish between right and left R-modules, depending on which side the product acts.

A R-module M shares many arithmetic operations and constructions with vector spaces. The R-module structure propagates to vectors M^p or matrices $\mathcal{M}_{p,q}(M)$: the left dot product with a vector in R^p or the left matrix product in $\mathcal{M}_{k,p}(R)$ (respectively) are both well defined.
As a consequence, for all positive integers N and k, we recall that $(\mathbb{T}_N[X]^k, +, \cdot)$ is a $\mathbb{Z}_N[X]$-module.

3.1.2 Probability distributions

In most FHE schemes, a small amount of noise from a Gaussian distribution is added during the encryption for security reasons. In the literature, this noise is quantified by using a noise parameter, that is proportional to the standard deviation of the Gaussian distribution: let σ be the standard deviation, the noise parameter is equal to $\sqrt{2\pi}\sigma$. The factor $\sqrt{2\pi}$ in the noise parameter is related to the Fourier parameter and it is used in the security analysis, but it is often a source of confusion in concrete implementations.
In the following, we always quantify the noise via its standard deviation or its variance, which lead to noise propagation formulas that are more natural.

Gaussian Distributions

Let $k \geq 1$ be an integer, $\sigma \in \mathbb{R}^+$ and $\mathbf{c} \in \mathbb{R}^k$. We say that a random variable $X \in \mathbb{R}^k$ has Gaussian (or Normal) distribution of center (or mean) $\mathbf{c}$ and standard deviation σ if and only if its probability density function is

$$\mathcal{D}^G_{\sigma,\mathbf{c}}(\mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\|\mathbf{x}-\mathbf{c}\|^2}{2\sigma^2}\right) \text{ for all } \mathbf{x} \in \mathbb{R}^k.$$

In the literature, the Gaussian distribution is noted by $\mathcal{N}(\mathbf{c}, \sigma^2)$ (σ^2 is the variance of the distribution). In this manuscript we note it as $\mathcal{D}^G_{\sigma,\mathbf{c}}$: the letter $\mathcal{D}$ is our notation for the probability distributions and the suffix G is used to distinguish the Gaussian one. If $\mathbf{c}$ is omitted, then it is implicitly set to $\mathbf{0}$.
Let S be a subset of $\mathbb{R}^k$. Then $\mathcal{D}^G_{\sigma,\mathbf{c}}(S)$ denotes $\sum_{x\in S} \mathcal{D}^G_{\sigma,\mathbf{c}}(\mathbf{x})$, if S discrete, or $\int_{x\in S} \mathcal{D}^G_{\sigma,\mathbf{c}}(\mathbf{x}) \cdot d\mathbf{x}$, if S is measurable.
For all closed (continuous or discrete) additive subgroup $M \subseteq \mathbb{R}^k$, $\mathcal{D}^G_{\sigma,\mathbf{c}}(M)$ is finite, and defines a *(restricted) Gaussian Distribution* $\mathcal{D}^G_{M,\sigma,\mathbf{c}}$ of standard deviation σ and center $\mathbf{c}$ over M, with the density function $\mathcal{D}^G_{M,\sigma,\mathbf{c}}(\mathbf{x}) = \mathcal{D}^G_{\sigma,\mathbf{c}}(\mathbf{x})/\mathcal{D}^G_{\sigma,\mathbf{c}}(M)$. Let L be a discrete subgroup of M, then the *Modular Gaussian distribution* $\mathcal{D}^G_{M/L,\sigma,\mathbf{c}}$ over M/L exists and is defined by the density $\mathcal{D}^G_{M/L,\sigma,\mathbf{c}}(\mathbf{x}) = \mathcal{D}^G_{M,\sigma,\mathbf{c}}(\mathbf{x}+L)$.

Remark 3.1.1. *A random variable $X \in \mathbb{R}$ is σ-SubGaussian if and only if it satisfies the Laplace-transformation bound. Namely, for all $t \in \mathbb{R}$ the expectation verifies $\mathbb{E}(\exp(tX)) \leq \exp(\sigma^2 t^2/2)$. Observe that a Gaussian distribution of standard deviation σ is also a σ-SubGaussian distribution. Let X and X' be two independent σ and*

σ'-SubGaussian random variables (respectively). Then, for all $\alpha, \beta \in \mathbb{R}$, the random variable obtained as the combination $\alpha X + \beta X'$ is $\sqrt{\alpha^2\sigma^2 + \beta^2\sigma'^2}$-SubGaussian. We often use this property in the analysis of the noise growth presented in following sections.

Concentrated distribution on the Torus

In general, distributions over the torus do not have expectation nor variance: for instance, it would be impossible to define the expectation of the uniform distribution over $\mathbb{T}$. However, when the support of the distribution is concentrated on a small interval, it is still possible to uniquely define these notions.
We say that a distribution $\mathcal{X}$ on the torus is *concentrated* if and only if its support is included in a ball of radius $\frac{1}{4}$ of $\mathbb{T}$, except for negligible probability. In this case, we define the *variance* $\mathsf{Var}(\mathcal{X})$ and the expectation $\mathbb{E}(\mathcal{X})$ of $\mathcal{X}$ as respectively

- $\mathsf{Var}(\mathcal{X}) = \min_{\bar{x}\in\mathbb{T}} \left(\sum_x p(x)|x - \bar{x}|^2 \right)$,
- $\mathbb{E}(\mathcal{X})$ as the position $\bar{x} \in \mathbb{T}$ which minimizes the variance expression.

In the variance formula, we make an abuse of notation by using the norm symbol $|x - \bar{x}|$: actually, it represents the distance of the two points over the torus. This definition of expectation by an optimization formula yields the same result, up to a negligible amount, as if we lift the distribution over any real interval of length $< \frac{1}{2}$, and compute its real expectation modulo 1.
By extension, we say that a distribution $\mathcal{X}$ over $\mathbb{T}^n$ or $\mathbb{T}_N[X]^k$ is concentrated if and only if each coefficient has an independent concentrated distribution on the torus. Then the expectation $\mathbb{E}(\mathcal{X})$ is the vector of expectations of each coefficient, and $\mathsf{Var}(\mathcal{X})$ denotes the maximum of each coefficient variance.

These expectation and variance over $\mathbb{T}$ follow the same linearity rules than their classical equivalent over the reals.

Remark 3.1.2. *Let $\mathcal{X}_1, \mathcal{X}_2$ be two independent concentrated distributions on either $\mathbb{T}, \mathbb{T}^n$ or $\mathbb{T}_N[X]^k$, and let $e_1, e_2 \in \mathbb{Z}$ such that $\mathcal{X} = e_1 \cdot \mathcal{X}_1 + e_2 \cdot \mathcal{X}_2$ remains concentrated. Then*

- $\mathbb{E}(\mathcal{X}) = e_1 \cdot \mathbb{E}(\mathcal{X}_1) + e_2 \cdot \mathbb{E}(\mathcal{X}_2)$,
- $\mathsf{Var}(\mathcal{X}) \leq e_1^2 \cdot \mathsf{Var}(\mathcal{X}_1) + e_2^2 \cdot \mathsf{Var}(\mathcal{X}_2)$.

up to a negligible amount.

Observe that SubGaussian distributions with small enough parameters are necessarily concentrated. Every distribution $\mathcal{X}$ on either $\mathbb{T}, \mathbb{T}^n$ or $\mathbb{T}_N[X]^k$ where each coefficient is σ-SubGaussian, with $\sigma \leq 1/\sqrt{32\ln(2)(\lambda+1)}$, is a concentrated distribution: a fraction $\geq 1 - 2^{-\lambda}$ of its mass is in the interval $[-\frac{1}{4}, \frac{1}{4}]$.

Remark 3.1.3. *A better solution might be to take truncated distributions (as suggested in [MP12]) in order to avoid all the negligible amounts appearing in the probability formulas.*

3.1.3 Distance and Norms

We denote as $\|\cdot\|_p$ and $\|\cdot\|_\infty$ the standard norms for scalars and vectors over the real field or over the integers. By extension, the norms $\|P(X)\|_p$ and $\|P(X)\|_\infty$ of a real or integer polynomial P are the norms of its coefficient vector. If P is a polynomial mod $X^N + 1$, we take the norm of its unique representative of degree $\leq N - 1$.
If $\mathbf{x}$ is an vector in $\mathbb{T}^k$, we note $\|\mathbf{x}\|_p = \min_{\mathbf{u} \in \mathbf{x} + \mathbb{Z}^k} (\|\mathbf{u}\|_p)$ is the p-norm of the representative of $\mathbf{x}$ with all coefficients in $]-\frac{1}{2}, \frac{1}{2}]$. It satisfies the separation and the triangular inequalities, but it is not a norm because it lacks homogeneity[3], and $\mathbb{T}^k$ is not a vector space either. Instead, it is sub-homogeneous, i.e. it satisfies the property $\|m \cdot \mathbf{x}\|_p \leq |m| \cdot \|\mathbf{x}\|_p$, $\forall m \in \mathbb{Z}$. By extension, we define $\|P(X)\|_p$ for a polynomial $P \in \mathbb{T}_N[X]$ as the p-norm of its unique representative in $\mathbb{R}[X]$ of degree $\leq N - 1$ and with coefficients in $]-\frac{1}{2}, \frac{1}{2}]$.

Definition 3.1.2 (Infinity norm over $\mathcal{M}_{p,q}(\mathbb{T}_N[X])$)**.** *Let $A \in \mathcal{M}_{p,q}(\mathbb{T}_N[X])$. We define the infinity norm of A as*

$$\|A\|_\infty = \max_{\substack{i \in [\![1,p]\!] \\ j \in [\![1,q]\!]}} \|a_{i,j}\|_\infty.$$

3.2 The Learning With Errors problem revisited

In this section, we revisit the LWE problem (and its variants) by abstracting over the real torus. A full definition of the LWE problem over the torus is given in [CS15]. They call the instance the Scale Invariant LWE problem.

Definition 3.2.1 ((Scale-Invariant) LWE (adapted from [CS15]))**.** *Let $n \geq 1$ be an integer, $\mathbf{s}$ be in $\mathbb{Z}^n$ and $\mathcal{X}$ a distribution over $\mathbb{R}$. We define $\mathsf{SILWE}_{\mathbf{s},\mathcal{X}}$ as the distribution over $\mathbb{T}^n \times \mathbb{T}$ obtained by sampling a pair $(\mathbf{a}, b)$, where the left member $\mathbf{a} \in \mathbb{T}^n$ is chosen uniformly random and the right member $b = \mathbf{a} \cdot \mathbf{s} + e$. The error e is a sample from the distribution $\mathcal{X}$. Let $\mathcal{S}$ be a distribution over $\mathbb{Z}^n$. We can define the two following problems.*

- *Decision problem: given arbitrarily many independent samples, distinguish between $\mathsf{SILWE}_{\mathbf{s},\mathcal{X}}$ samples and uniformly random samples from $\mathbb{T}^n \times \mathbb{T}$, for a fixed $s \leftarrow \mathcal{S}$.*

[3] Mathematically speaking, a more accurate notion would be $\mathsf{dist}_p(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_p$, which is a distance. However, the norm symbol is clearer for almost all practical purposes.

- *Search problem: given arbitrarily many independent* $\mathsf{SILWE}_{\mathbf{s},\mathcal{X}}$ *samples, find* $s \leftarrow \mathcal{S}$.

As for LWE, the scale invariant problems are considered hard. The SILWE problem can be easily used to encrypt messages from a discrete message space $\mathcal{M} \subset \mathbb{T}$. The idea can be easily explained with images (as shown in [CGGI16b]) and reflects the idea of Regev's cryptosystem. Formal details are given later.
Figure 3.1 represents the pair $(\mathbf{a}, b) \in \mathbb{T}^n \times \mathbb{T}$. The left part $\mathbf{a}$, also called *mask*, is indicated inside the ring, while the right part b is represented by the ring itself. The ring can be seen as the continuous real torus, with 0 in the bottom and the other values of the interval following in counterclockwise. We chose a discrete message space $\mathcal{M} \subset \mathbb{T}$: in the figure we represent $\mathcal{M} = \{0, 1/3, 2/3\}$.

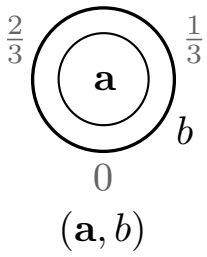

Figure 3.1: The idea*: the circle represents the real torus, and in particular the right part of the sample* $b \in \mathbb{T}$*. The left part is indicated inside the ring by* $\mathbf{a} \in \mathbb{T}^n$*. In this example, the discrete message space* $\mathcal{M} \subset \mathbb{T}$ *is set to* $\{0, 1/3, 2/3\}$*: the* 0 *is in the bottom and the other messages follow in counterclockwise.*

Now, if we want to encrypt a message $\mu \in \mathcal{M}$ ($\mu = 1/3$ in the example shown in Figure 3.2), we start by computing a quantity called the phase, noted φ, and by sampling a random mask $\mathbf{a} \in \mathbb{T}^n$. The phase can be computed in two ways:

1. As the addition between the message and a random small error from the distribution $\mathcal{X}$, that is generally chosen to be the Gaussian distribution (centered in 0),

2. As a random sample from a Gaussian distribution centered in the message.

The pair $(\mathbf{a}, \varphi)$ is called the *private representation*: the phase is in clear and it reveals the message. By knowing the message space $\mathcal{M}$, the message can be retrieved from the phase in two ways:

1. Rounding the phase to the nearest message possible in $\mathcal{M}$,

2. Mathematically evaluate the expectation of its Gaussian distribution (in a probability space Ω we describe later). In practice, we never use this method because an unique sample is not sufficient to compute the expectation.

In order to hide the phase, we produce the *public representation* by using a secret key $\mathbf{s} \in \mathbb{Z}^n$ (or more commonly in $\mathbb{B}^n$). We compute $b = \varphi + \mathbf{a} \cdot \mathbf{s}$ and we reveal the pair $(\mathbf{a}, b)$ (Figure 3.3). Thanks to the hardness of the SILWE problem, this is indistinguishable from a random sample in $\mathbb{T}^n \times \mathbb{T}$, so the message is hidden. A special case is the one where the mask $\mathbf{a}$ is chosen as the vector of zeros. In this case the public and private representations are equal and the phase is in clear: we call such samples *trivial* and they never occur spontaneously.

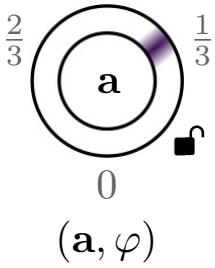

Figure 3.2: Private representation*: the phase is in clear and the message can be easily found.*

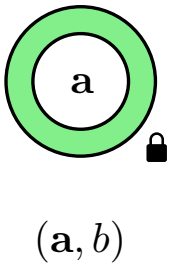

Figure 3.3: Public representation*: indistinguishable from a random sample in $\mathbb{T}^n \times \mathbb{T}$.*

To decrypt the message, we "unlock" the public representation by using the secret key: we compute the phase as $\varphi = b - \mathbf{a} \cdot \mathbf{s}$ and we retrieve the message from the phase as explained before.
The LWE samples have the additive homomorphic property, as shown in Figure 3.4. When we add homomorphically two samples (or more), we simply add the right and left part, and at the same time, we are adding the phases. From what we explained in Remarks 3.1.1 and 3.1.2, as the message is the expectation of the phase, the message of the resulting ciphertext is the sum of the original messages, but the standard deviation σ, representing the noise of ciphertexts, grows sublinearly.

Remark 3.2.1. *This encryption scheme has also an asymmetric variant: the public key is set as a list of random fresh encryptions of* 0*. To encrypt a message $\mu \in \mathcal{M}$, one chooses a small random subset of the elements of the public key, and sums it to the trivial sample $(\mathbf{0}, \mu)$.*

The same idea presented in the figures can be extended to the ring version of LWE. In order to unify, generalize and abstract all the instances, we propose in next section a new definition of the LWE problem, over the torus, that we call TLWE. The definition follows the same footstep of Definition 3.2.1 (as in [CS15]) and Definition 2.2.3 (as in [BGV12]). Then, we extend the same formalism to the description of the GSW construction, and we call it TGSW. The new formalism helps to better clarify the

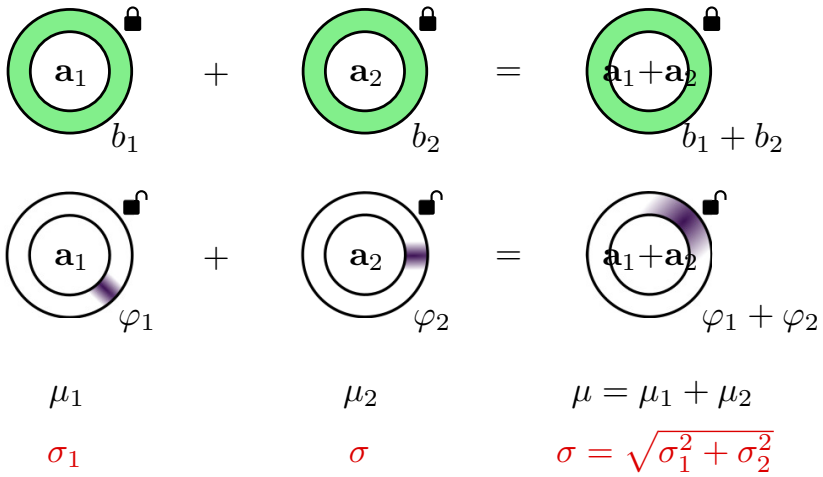

Figure 3.4: Additive homomorphic property*: the homomorphic addition between* LWE *ciphertexts is performed by simply adding the right and the left parts. At the same time, phases and messages (expectation of the phase) are added while the standard deviation grows sublinearly.*

intricate-at-first-sight structure of the constructions and allows to explicit all the operations between the messages in the different plaintext spaces (and so between ciphertexts).

3.2.1 TLWE

In this section we clarify and formalize the ideas proposed in the previous section and we give a generalized definition of LWE over the torus, that we call TLWE.
In the previous example, we talked about the phase as a quantity used during encryption and decryption. In TLWE, the phase is defined as a proper function depending on the secret key and becomes the central building block. All other notions are deduced from the algebraic properties of this linear function: message space, ciphertext space, encryption, decryption. In particular, this abstraction allows to unify scale invariant FHE schemes, based not only on LWE, RingLWE, GLWE, but also on other problems like Approx-GCD or NTRU.

In this section and in the following one we distinguish two types of definitions, "abstract" and "canonical". The abstract definitions (the term *abstract* is explicit) are interesting from a theoretical point of view: they are generalized and include the different instances of scale invariant problems/schemes. Fixing the parameters in the abstract definitions gives the different instances.
Our TFHE scheme uses only scalar and ring LWE instances. It is for this reason that

we also give explicit classical definitions (again over the torus): these are the ones which are sufficient for the comprehension of TFHE.

Definition 3.2.2 (Abstract TLWE problems)**.** *Let I be an ideal of $\mathbb{Z}[X]$, we call $\mathfrak{R} = \mathbb{Z}[X]/I$ and $\mathbb{T}_I[X] = \mathbb{T}[X]/I$. A* phase function *is a lipschitzian*[4] *$\mathfrak{R}$-module morphism from a $\mathfrak{R}$-module M to $\mathbb{T}_I[X]$. The abstract* TLWE *problem is parametrized by an error distribution $\mathcal{X}$ on M, and a family $(\varphi_s)_{s\in\mathcal{S}}$ of phase functions, indexed by a secret s. The* homogeneous abstract TLWE distribution *for the secret s is $\mathcal{U}_{\ker(\varphi_s)}+\mathcal{X}$ (sum of the uniform distribution over $\ker(\varphi_s)$ and an error from $\mathcal{X}$). Abstract* TLWE *is λ-secure if neither of the following two problems can be solved in less than 2^λ bit operations, or with advantage $2^{-\lambda}$ by any PPT*[5] *adversary:*

- *Abstract* TLWE *decision problem: given arbitrarily many samples in M, distinguish if they come from the uniform distribution on M or from $\mathcal{U}_{\ker(\varphi_s)}+\mathcal{X}$ for a particular but unknown secret phase φ_s.*
- *Abstract* TLWE *search problem: given arbitrarily many samples from $\mathcal{U}_{\ker(\varphi_s)}+\mathcal{X}$ for a particular secret phase φ_s, find s.*

If for all secret s, the distributions $\mathcal{U}_{\ker\varphi_s}+\mathcal{X}$ is concentrated, Regev's cryptosystem can be abstracted as follows:

- The message space is the image of φ_s in $\mathbb{T}_I[X]$.
- The ciphertext space is the domain M of φ_s.
- The encryption of μ is an approximation of a random preimage $\varphi_s^{-1}(\mu)$. Abstractly, a sample from $\mathcal{U}_{\varphi_s^{-1}(\mu)}+\mathcal{X}$.
- The (approximate) decryption of a ciphertext $\mathbf{c}$ is its image $\varphi_s(\mathbf{c})$.

Different choices of the ideal I, of the module M and of the phase function φ_s, explicit different instances of the abstract TLWE problem.
If we instantiate this definition with $I = (X + 1)$, then we get $\mathfrak{R} = \mathbb{Z}$ and $\mathbb{T}_I[X] = \mathbb{T}$, and obtain scalar schemes. Setting $M = \mathbb{T}^{n+1}$ and the phase as $\varphi_s(\mathbf{a}, b) = b - \mathbf{a}\cdot\mathbf{s}$, we retrieve the scale-invariant LWE from Definition 3.2.1. By choosing instead $M = (\mathbb{Z}/q\mathbb{Z})^{n+1}$ with phase $\varphi_s(a, b) = (b - sa)/q$ and discrete Gaussian error, we retrieve the well known LWE mod q (described in Chapter 2). If we set $M = \mathbb{T}$ and take $\varphi_s(x) = p.x$ where p is a secret integer, then the TLWE problem consists in recognizing approximations of multiples of $1/p$, so the TLWE abstraction can express cryptosystems based on the (dual) approx-GCD problem. Now, if we take a different ideal, for instance $I = (X^N + 1)$, then the

[4] In short, a lipschizian function is a function for which the ratio between the variation of the ordinates and the variation of the abscissas is limited by a constant positive factor.

[5] Probabilistic Polynomial Time.

canonical choice for a phase $\varphi_s(\mathbf{a}, b) = b - \mathbf{a} \cdot \mathbf{s}$ expresses RingLWE, depending on the dimension of $\mathbf{a}$. Many other choices of phases are possible, for instance $\varphi_{(f,g)} : \mathbb{T}_N[X]^2 \to \mathbb{T}_N[X], (x, y) \mapsto fx - gy$ for small secret polynomials f, g, would allow to build FHE over scale invariant version of NTRU.

In the rest of the manuscript, we are mainly interested in the "canonical" TLWE problem (that we just call the TLWE problem): we give a specific definition for that.

Definition 3.2.3 (TLWE problem)**.** *Let $k \geq 1$ be an integer, N be a power of 2 and $\sigma \in \mathbb{R}_{\geq 0}$ be a standard deviation. Let the secret key space $\mathcal{S}$ be composed by the binary vectors $\mathbf{s} \in \mathbb{B}_N[X]^k$ that we assume to be uniformly chosen with $n \approx kN$ bits of entropy[6]. The phase $\varphi_{\mathbf{s}}$ is defined over $M = \mathbb{T}_N[X]^k \times \mathbb{T}_N[X]$ by $\varphi_{\mathbf{s}}((\mathbf{a}, b)) = b - \mathbf{a} \cdot \mathbf{s}$. By definition it is $(kN + 1)$-lipschitzian for the ℓ_∞ distance. The error distribution $\mathcal{X}$ is $(\mathbf{0}, \mathcal{D}^G_{\mathbb{T}_N[X],\sigma})$, where $\mathcal{D}^G_{\mathbb{T}_N[X],\sigma}$ is the modular Gaussian distribution of standard deviation σ over $\mathbb{T}_N[X]$. By definition, a homogeneous* TLWE *sample can be constructed as $(\mathbf{a}, \mathbf{a} \cdot \mathbf{s} + e)$ where $\mathbf{a}$ is uniformly sampled in $\mathbb{T}_N[X]^k$ (or in a sufficiently dense submodule[7]) and $e \leftarrow \mathcal{D}^G_{\mathbb{T}_N[X],\sigma}$. As before, we can define two problems:*

- TLWE *decision problem: for a particular but unknown secret phase $\varphi_{\mathbf{s}}$, distinguish between the uniform distribution on $\mathbb{T}_N[X]^k \times \mathbb{T}_N[X]$ and the samples from the* TLWE *distribution.*
- TLWE *search problem: given arbitrarily many samples from the* TLWE *distribution for a particular secret phase $\varphi_{\mathbf{s}}$, find $\mathbf{s}$.*

Definition 3.2.3 considers a continuum between the LWE (for $N = 1$) and the RingLWE (for $k = 1$) instances, but in the following we only consider these two instances for our applications.

Furthermore, we define as *trivial* the samples having the mask $\mathbf{a} = \mathbf{0}$ and as *noiseless* the samples having the standard deviation $\sigma = 0$. To encrypt a message $\mu \in \mathbb{T}_N[X]$ (i.e. produce a *fresh* ciphertext of μ) we sum the trivial sample $(\mathbf{0}, \mu) \in \mathbb{T}_N[X]^k \times \mathbb{T}_N[X]$ with a homogeneous TLWE sample.

From a practical point of view, the fact that the phase is lipschitzian makes this decryption resilient to numerical errors, and allows to work with approximations. Plus, thanks to the linearity of the phase, the cryptosystem is homomorphic with

[6] An equivalence between LWE and binLWE, i.e. LWE with binary secret has been proven in [BLP+13, Mic18]. The same reduction for the Ring variant of LWE is still an open problem.

[7] A submodule G is sufficiently dense if there exists an intermediate submodule H such that $G \subseteq H \subseteq \mathbb{T}^n$, the relative smoothing parameter $\eta_{H,\varepsilon}(G)$ is $\leq \sigma$, and H is the orthogonal in $\mathbb{T}^n$ of at most $n - 1$ vectors of $\mathbb{Z}^n$. This definition allows to convert any (Ring)-LWE with non-binary secret to a TLWE instance via binary decomposition.

respect to the addition. However, it is a noisy cryptosystem, in a sense that after encrypting and decrypting a message μ, the result is not exactly μ, but a close approximation $\mu + e$, where $e \leftarrow \mathcal{X}$ is a small error.
There are use-cases, like floating point computations [CKKS17] or in general differential privacy, where these approximations of the plaintext are considered valid. However, if we need an exact result, we have two options. The first one is the historical choice in Regev cryptosystem: restrict the message space to a discrete subset, whose packing radius is larger than the amplitude of the distribution $\mathcal{X}$, and retrieve the exact plaintext by rounding the phase. If rounding is easy to set-up in practice, its non-linearity complicates the correctness analysis, especially when the current sample is not fresh, but rather a linear combination of previous samples. Also, restricting the message space prevents some floating point applications and bounds plaintext operations to just small abelian groups. The second option, consists in taking $\mathbb{E}(\mathcal{X}) = 0$, and thus, the plaintext becomes the expectation of the phase. This option does not require to restrict the message space and works with infinite precision over the continuous one. Furthermore, the continuity and linearity of the expectation ease the analysis of morphism properties and of the noise propagation, but it requires to properly define a probability space, we call Ω.

Definition 3.2.4 (The Ω-probability space)**.** *Since samples are either independent (random, noiseless, or trivial) fresh* $\mathbf{c} \leftarrow \mathsf{TLWE}_{\mathbb{T}_N[X],\mathbf{s},\sigma}(\mu)$*, or linear combination* $\tilde{\mathbf{c}} = \sum_{i=1}^{p} e_i \cdot \mathbf{c_i}$ *of other samples, the probability space* Ω *is the product of the probability spaces of each individual fresh samples* $\mathbf{c}$ *with the* TLWE *distributions defined in definition 3.2.3, and of the probability spaces of all the coefficients* $(e_1, \ldots, e_p) \in \mathbb{Z}_N[X]^p$ *or* $\mathbb{Z}^p$ *that are obtained with randomized algorithm.*

In other words, instead of viewing a TLWE sample as a fixed value which is the result of one particular event in Ω, we will consider all the possible values at once, and make statistics on them.

To conclude this section, we define some important functions on TLWE samples: message, error, noise variance, and noise norm. These functions are well defined mathematically, and can be used in the analysis of various algorithms. However, they cannot be directly computed or approximated in practice: in fact we want to compute these functions on single samples from the random variable, but to compute variance and expectation a single sample is not sufficient.

Definition 3.2.5. *Let* $\mathbf{c}$ *be a random variable* $\in \mathbb{T}_N[X]^{k+1}$*, which we interpret as a* TLWE *sample. All probabilities are on the* Ω*-space. We say that* $\mathbf{c}$ *is a* valid TLWE *sample if and only if there exists a key* $\mathbf{s} \in \mathbb{B}_N[X]^k$ *such that the distribution of the phase* $\varphi_{\mathbf{s}}(\mathbf{c})$ *is concentrated. If* $\mathbf{c}$ *is trivial, all keys* $\mathbf{s}$ *are equivalent, else the mask of* $\mathbf{c}$ *is uniformly random, so* $\mathbf{s}$ *is unique. We then define:*

- *The* message *of* $\mathbf{c}$*, denoted as* $\mathsf{msg}(\mathbf{c}) \in \mathbb{T}_N[X]$*, as the expectation of* $\varphi_{\mathbf{s}}(\mathbf{c})$*.*

- *The* error, *denoted* $\mathsf{Err}(\mathbf{c})$, *is equal to* $\varphi_{\mathbf{s}}(\mathbf{c}) - \mathsf{msg}(\mathbf{c})$.
- $\mathsf{Var}(\mathsf{Err}(\mathbf{c}))$ *denotes the variance of* $\mathsf{Err}(\mathbf{c})$, *which is by definition also equal to the variance of* $\varphi_{\mathbf{s}}(\mathbf{c})$.
- *Finally,* $\|\mathsf{Err}(\mathbf{c})\|_\infty$ *denotes the maximum amplitude of* $\mathsf{Err}(\mathbf{c})$ *(possibly with overwhelming probability)*[8].

Unlike the classical decryption algorithm, the message function can be viewed as an ideal black box decryption function, which works with infinite precision even if the message space is continuous. Provided that the noise amplitude remains smaller than $\frac{1}{4}$, the message function is perfectly linear. Using these intuitive and intrinsic functions will considerably ease the analysis of all algorithms in this paper.

Remark 3.2.2. *Let* $\mathbf{c_1}, \ldots, \mathbf{c_p}$ *be* p *valid and independent* TLWE *samples under the same key* $\mathbf{s}$, *and let* $e_1, \ldots, e_p \in \mathfrak{R}$ *be* p *integer polynomials. If the linear combination* $\mathbf{c} = \sum_{i=1}^{p} e_i \cdot \mathbf{c_i}$ *is a valid* TLWE *sample, it satisfies* $\mathsf{msg}(\mathbf{c}) = \sum_{i=1}^{p} e_i \cdot \mathsf{msg}(\mathbf{c}_i)$ *with*

- *Variance:* $\mathsf{Var}(\mathsf{Err}(\mathbf{c})) \leq \sum_{i=1}^{p} \|e_i\|_2^2 \cdot \mathsf{Var}(\mathsf{Err}(\mathbf{c_i}))$,
- *Noise amplitude:* $\|\mathsf{Err}(\mathbf{c})\|_\infty \leq \sum_{i=1}^{p} \|e_i\|_1 \cdot \|\mathsf{Err}(\mathbf{c_i})\|_\infty$.

If the last bound is $< \frac{1}{4}$, *then* $\mathbf{c}$ *is necessarily a valid* TLWE *sample (under the same key* $\mathbf{s}$*).*

3.2.2 TGSW

As presented in previous section, TLWE samples can be linearly combined to obtain a new sample encrypting the linear combination of the messages. But when it comes to non linear operations on the samples, TLWE seems to lack some properties. To remedy, several schemes based on the different variants of LWE have been proposed. Between them, the most known solutions are the BGV constructions [BGV12] and the GSW constructions [GSW13], described in Chapter 2. The TFHE scheme is based on the latter construction and on the improvements proposed in [AP14] and [DM15]. To follow the same footstep as in the previous section, we start by abstracting and generalizing the GSW problem over the torus. We call this version TGSW: it is scale invariant and it includes both the scalar and the ring modes. The scheme relies on a gadget decomposition function: the novelty we propose in our construction is that our decomposition function performs an *approximate decomposition*, up to some precision parameter. This allows to improve running time and memory requirements for a small amount of additional noise.
We give the abstract definition of the gadget decomposition and of the TGSW construction. Then, as we did for TLWE, we concentrate on the canonical definitions,

[8] Talking about maximum amplitude is still an abuse of notation. As we said in Remark 3.1.3, a more correct approach would be to use a truncated distribution.

which are the ones necessary to the comprehension of the TFHE scheme and of all the ideas behind.

Observe that, if we decompose with respect to a gadget H, the "inverse" operation is the multiplication between the decomposed vector and the gadget H itself, which is linear.

Definition 3.2.6 (Abstract Gadget Decomposition)**.** *Let M be a $\mathfrak{R}$-module (as in Definition 3.2.2). We say that an efficient algorithm $Dec_{H,\beta,\epsilon}(\mathbf{v})$ is a valid decomposition on the gadget $H \in M^{\ell'}$ with quality $\beta \in \mathbb{R}_{>0}$ and precision $\epsilon \in \mathbb{R}_{>0}$ if and only if, for any (abstract)* TLWE *sample $\mathbf{v} \in M$, it efficiently and publicly outputs a small vector $\mathbf{u} \in \mathfrak{R}^{\ell'}$ such that $\|\mathbf{u}\|_\infty \leq \beta$ and $\|\mathbf{u} \cdot H - \mathbf{v}\|_\infty \leq \epsilon$. Furthermore, the expectation of $\mathbf{u} \cdot H - \mathbf{v}$ must to be equal to 0 when $\mathbf{v}$ is uniformly distributed in M.*

To fix the ideas, we give an efficient (canonical) example of gadget decomposition, whose purpose is to decompose TLWE ciphertexts: this is the one used in TFHE. We fix the module $M = \mathbb{T}_N[X]^k \times \mathbb{T}_N[X]$. The gadget is a block diagonal matrix on which each column-block on the diagonal contains a geometric decreasing sequence of constant polynomials in $\mathbb{T} \subseteq \mathbb{T}_N[X]$, and the corresponding decomposition function is the greedy algorithm. Decomposing with respect to a super-decreasing/increasing basis is an easy operation. A trivial every-day-life example is the decomposition in euros. We are able to easily pay in cash when we go on the supermarket, because we are able to decompose the amount of money we have to pay with respect to the base composed by the different banknote/coin denominations.
In theory, decomposition algorithms should be randomized to ensure that the distribution of all error coefficients remain independent. In practice, our average case theorems already rely on an independence Heuristic 3.2.1 that we describe later in this section, which explains why we use a deterministic canonical decomposition.

Lemma 3.2.1 (Gadget Decomposition)**.** *Let the module $M = \mathbb{T}_N[X]^{k+1}$ be the domain of* TLWE*, and ℓ and B_g (called the base of the gadget) be two positive integers. The (canonical) gadget is the list of $\ell' = (k+1)\ell$ rows of the matrix $H \in \mathcal{M}_{(k+1)\ell,k+1}(\mathbb{T}_N[X])$ as in* (3.1)*.*

$$H = \left(\begin{array}{c|c|c} 1/B_g & \dots & 0 \\ \vdots & \ddots & \vdots \\ 1/B_g^\ell & \dots & 0 \\ \hline \vdots & \ddots & \vdots \\ \hline 0 & \dots & 1/B_g \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1/B_g^\ell \end{array}\right) \in \mathcal{M}_{(k+1)\ell,k+1}(\mathbb{T}_N[X]). \tag{3.1}$$

Then for $\beta = \frac{B_g}{2}$ and $\epsilon = \frac{1}{2B_g^\ell}$, Algorithm 1 is a valid $Dec_{H,\beta,\epsilon}$.

Algorithm 1 Gadget Decomposition of a TLWE sample

Input: A TLWE sample $(\mathbf{a}, b) = (a_1, \ldots, a_k, b = a_{k+1}) \in \mathbb{T}_N[X]^k \times \mathbb{T}_N[X]$
Output: A combination $[u_{1,1}, \ldots, u_{k+1,\ell}] \in \mathbb{Z}_N[X]^{(k+1)\ell}$

1: For each a_i choose the unique representative $\sum_{j=0}^{N-1} a_{i,j} X^j$, with $a_{i,j} \in \mathbb{T}$, and set $\bar{a}_{i,j}$ the closest multiple of $\frac{1}{B_g^\ell}$ to $a_{i,j}$
2: Decompose each $\bar{a}_{i,j}$ uniquely as $\sum_{p=1}^{\ell} \bar{a}_{i,j,p} \frac{1}{B_g^p}$ where each $\bar{a}_{i,j,p} \in [\![-B_g/2, B_g/2[\![$
3: **for** $i = 1$ **to** $k+1$
4: **for** $p = 1$ **to** ℓ
5: $u_{i,p} = \sum_{j=0}^{N-1} \bar{a}_{i,j,p} X^j \in \mathbb{Z}_N[X]$
6: Return $(u_{i,p})_{i,p}$

Proof. Let $\mathbf{v} = (\mathbf{a}, b) = (a_1, \ldots, a_k, b = a_{k+1}) \in \mathbb{T}_N[X]^{k+1}$ be a TLWE sample, given as input to Algorithm 1. Let $\mathbf{u} = [u_{1,1}, \ldots, u_{k+1,\ell}] \in \mathbb{Z}_N[X]^{(k+1)\ell}$ be the corresponding output: by construction $\|\mathbf{u}\|_\infty \leq B_g/2 = \beta$.
Let $\epsilon^{\mathbf{dec}} = \mathbf{u} \cdot H - \mathbf{v}$. For all $i \in [\![1, k+1]\!]$ and $j \in [\![0, N-1]\!]$, we have by construction

$$\epsilon^{\mathbf{dec}}{}_i = \sum_{p=1}^{\ell} u_{i,p} \cdot \frac{1}{B_g^p} - a_i = \sum_{j=0}^{N-1} (\bar{a}_{i,j} X^j) - a_i$$

and so

$$\epsilon^{\mathbf{dec}}{}_{i,j} = \bar{a}_{i,j} - a_{i,j}.$$

Since $\bar{a}_{i,j}$ is defined as the nearest multiple of $\frac{1}{B_g^\ell}$ on the torus, we have $|\bar{a}_{i,j} - a_{i,j}| \leq \frac{1}{2B_g^\ell} = \epsilon$.
The decomposition error $\epsilon_{\mathbf{dec}}$ has therefore a concentrated distribution when $\mathbf{v}$ is uniform. We now verify that it is zero-centered. We call f the function from $\mathbb{T}$ to $\mathbb{T}$ which rounds an element x to its closest multiple of $\frac{1}{B_g^\ell}$ and g the (symmetry) function defined by $g(x) = 2f(x) - x$ on the torus. We can easily verify that the $\mathbb{E}(\epsilon^{\mathbf{dec}}{}_{i,j})$ is equal to $\mathbb{E}(a_{i,j} - f(a_{i,j}))$ when $a_{i,j}$ has uniform distribution, which is equal to $\mathbb{E}(g(a_{i,j}) - f(g(a_{i,j})))$ when $g(a_{i,j})$ has uniform distribution, also equal to $\mathbb{E}(f(a_{i,j}) - a_{i,j}) = -\mathbb{E}(\epsilon^{\mathbf{dec}}{}_{i,j})$. Thus, the expectation of $\epsilon^{\mathbf{dec}}$ is 0. □

We are now ready to define (abstract) TGSW samples, and to extend the notions of phase of valid sample, message and error of the samples. Observe that, while in TLWE samples the messages were torus polynomials of $\mathbb{T}_N[X]$, here we encrypt integer polynomials in $\mathbb{Z}_N[X]$.

Definition 3.2.7 (Abstract TGSW samples)**.** *Consider the* TLWE *cryptosystem of error distribution $\mathcal{X}$ and of secret phase φ_s on the $\mathfrak{R}$-module M, and its associated gadget decomposition $Dec_{H,\beta,\epsilon}$ over $H \in M^{\ell'}$. We say that $C \in M^{\ell'}$ is a fresh* TGSW *sample of $\mu \in \mathfrak{R}$ if and only if $C = Z + \mu \cdot H$ where each element of $Z \in M^{\ell'}$ is an Homogeneous* TLWE *sample (of 0) and error $\mathcal{X}$.*

Reciprocally, we say that an element $C \in M^{\ell'}$ is a valid TGSW *sample for the key s if and only if there exists a unique polynomial $\mu \in \mathfrak{R}$ (modulo $H \cdot \mathfrak{R}$), such that each row of $C - \mu \cdot H$ is a valid* TLWE *sample of 0 for the key $\mathbf{s}$. We call the polynomial μ the message of C, and we denote it by $\mathsf{msg}(C)$. By extension, the phase of C denoted as $\varphi_{\mathbf{s}}(C) \in \mathbb{T}_I[X]^{\ell'}$ is the vector of the ℓ'* TLWE *phases of each line of C. In the same way, we define the error of C, denoted $\mathsf{Err}(C)$, as the list of the ℓ'* TLWE *errors of each row of C.*

As we did for TLWE samples, we give the (canonical) definitions of TGSW samples, which is the one used in the TFHE scheme. If one instantiate the previous definition with the TLWE samples from Definition 3.2.3 and the decomposition algorithm from Lemma 3.2.1, one obtains the (canonical) TGSW samples over $\mathbb{T}_N[X]^{(k+1)\ell}$, of binary key $\mathbf{s} \in \mathbb{B}_N[X]^k$, and Gaussian error of standard deviation σ.
Fresh TGSW samples of a message $\mu \in \mathbb{Z}_N[X]$ are denoted $\mathsf{TGSW}_{\mathbf{s},\sigma}(\mu)$. Since TGSW samples are essentially vectors of TLWE samples, they are naturally compatible with linear combinations, and both phase and message functions remain linear.

Definition 3.2.8 (TGSW samples)**.** *Let ℓ and $k \geq 1$ be two integers, $\sigma \geq 0$ be a standard deviation and H the gadget defined in Equation* (3.1)*. Let $\mathbf{s} \in \mathbb{B}_N[X]^k$ be a* TLWE *key, we say that $\mathbf{C} \in \mathcal{M}_{(k+1)\ell,k+1}(\mathbb{T}_N[X])$ is a fresh* TGSW *sample of $\mu \in \mathbb{Z}_N[X]$ with standard deviation σ if and only if $\mathbf{C} = \mathbf{Z} + \mu \cdot H$ where each row of $\mathbf{Z} \in \mathcal{M}_{(k+1)\ell,k+1}(\mathbb{T}_N[X])$ is an Homogeneous* TLWE *sample (of 0) with Gaussian standard deviation σ (Definition 3.2.3).*
Reciprocally, we say that an element $\mathbf{C} \in \mathcal{M}_{(k+1)\ell,k+1}(\mathbb{T}_N[X])$ is a valid TGSW *sample if and only if there exists a unique polynomial $\mu \in \mathbb{Z}_N[X]$ and a unique key $\mathbf{s}$ such that each row of $\mathbf{C} - \mu \cdot H$ is a valid* TLWE *sample of 0 for the key $\mathbf{s}$. We call the polynomial μ the message of $\mathbf{C}$, and we denote it by $\mathsf{msg}(C)$.*

We now define the phase and the error for TGSW. As a TGSW ciphertext is a list of TLWE encryptions, the phase and the error are the list of phases and errors of each of the composing TLWE ciphertexts.

Definition 3.2.9 (Phase and Error)**.** *Let $C \in \mathcal{M}_{(k+1)\ell,k+1}(\mathbb{T}_N[X])$ be a* TGSW *sample for a secret key $\mathbf{s} \in \mathbb{B}_N[X]^k$ and standard deviation $\sigma \geq 0$.*
We define the phase of C, denoted as $\varphi_{\mathbf{s}}(C) \in \mathbb{T}_N[X]^{(k+1)\ell}$, as the list of the $(k+1)\ell$ TLWE *phases of each line of C. In the same way, we define the error of C, denoted $\mathsf{Err}(C)$, as the list of the $(k+1)\ell$* TLWE *errors of each line of C.*

As a consequence, the phase is still $(1+kN)$-lipschitzian for the infinity norm.

Remark 3.2.3. *For all $C \in \mathcal{M}_{p,k+1}(\mathbb{T}_N[X])$, $\|\varphi_{\mathbf{s}}(C)\|_\infty \leq (1+kN)\|C\|_\infty$.*

Finally, we observe the noise amplitude and variance growth, when many TGSW samples are linearly combined.

Remark 3.2.4. *Given p valid* TGSW *samples $C_1, \dots, C_p$ of messages $\mu_1, \dots, \mu_p$ under the same key, and with independent error coefficients, and given p integer polynomials $e_1, \dots, e_p \in fR$, the linear combination $C = \sum_{i=1}^{p} e_i \cdot C_i$ is a* TGSW *sample of $\mu = \sum_{i=1}^{p} e_i \cdot \mu_i$, with*

- *Variance:* $\mathsf{Var}(C) = (\sum_{i=1}^{p} \|e_i\|_2^2 \cdot \mathsf{Var}(C_i))^{1/2}$,
- *Noise amplitude:* $\|\mathsf{Err}(C)\|_\infty = \sum_{i=1}^{p} \|e_i\|_1 \cdot \|\mathsf{Err}(C_i)\|_\infty$.

Heuristic

A natural question that comes to mind is what happens to the distribution of the samples that have been linearly combined. In the worst-case, the behaviour is provable, while the average case is less clear.
In order to characterize the average case behaviour of our homomorphic operations, we shall rely on the independence heuristic Assumption 3.2.1. This heuristic will only be used for practical average-case bounds: our worst-case theorems and lemmas based on the infinite norm do not use it at all.

Assumption 3.2.1 (Independence Heuristic)**.** *All the coefficients of the errors of* TLWE *or* TGSW *samples that occur in all the linear combinations we consider are independent and concentrated. More precisely, they are σ-SubGaussian where σ is the standard deviation.*

This assumption allows us to bound the variance of the noise instead of its norm, and to provide realistic average-case bounds which often correspond to the square root of the worst-case ones. The error can easily be proved SubGaussian, since each coefficients are always obtained by involving Gaussians or zero-centered bounded uniform distributions. What remains heuristic is the independence between all the coefficients. Indeed, dependencies between coefficients may affect the variance of their combinations in both directions. The independence of coefficients can be proved if we add enough entropy in the decomposition algorithm (and if we increase all the other parameters to compensate), but as noticed in [DM15], this work-around seems just to be a proof artifact, and is experimentally not needed. Since our average-case corollaries should reflect practical results, we leave the independence of SubGaussian samples as a heuristic assumption.
This independence assumption has been experimentally validated by the TFHE implementation: in Chapter 6 we give more details on the subject.

3.2.3 Products

The interest of using TGSW samples instead of just TLWE samples is that we can perform an internal product on TGSW, instead of just linear combinations. In fact, linear operations are not sufficient to achieve Fully Homomorphic Encryption. The

original definition of GSW in [GSW13] proposed a construction to achieve a homomorphic internal product between the integer messages of two GSW ciphertexts (which live in the ring $\mathbb{Z}_N[X]$). As already described in Chapter 2, the product is performed by decomposing one of the two ciphertexts with respect to the gadget, and by multiplying this decomposition with the other ciphertext. The choice of the ciphertext to decompose does not change the final decrypted result but can produce different noise amounts in the resulting ciphertext. Thus, the GSW product is asymmetric in the sense of the noise propagation, as already noticed in [GVW15], [BV14b] and [AP14], and GSW ciphertexts are particularly suited to evaluate long chains of products, or branching programs. In [BP16] and [CGGI16a], the authors noticed that for these circuits, a large part of the computations in the GSW internal product was subsequently unused, if the final goal was just to decrypt the message. Not performing these computations yields a huge polynomial speed-up.
In this section, we provide an intrinsic explanation for the correctness of these partial computations, by defining a TLWE *external product* between a TGSW ciphertext and a TLWE sample, corresponding in clear to the external $\mathbb{Z}_N[X]$-module product between the two plaintexts (i.e and integer polynomial in $\mathbb{Z}_N[X]$ and a torus polynomial in $\mathbb{T}_N[X]$ respectively). Then, we provide a direct comparison between the external [GSW13, AP14, GINX14, DM15] and internal products, by re-defining the internal product as a list of $\ell' = (k+1)\ell$ independent external products. For both products, we analyze the noise growth in the worst and average cases.

Definition 3.2.10 (External product)**.** *We define the product* $\boxdot$ *as*

$$\begin{aligned} \boxdot\colon \mathsf{TGSW} \times \mathsf{TLWE} &\longrightarrow \mathsf{TLWE} \\ (A, \mathbf{b}) &\longmapsto A \boxdot \mathbf{b} = Dec_{H,\beta,\epsilon}(\mathbf{b}) \cdot A, \end{aligned}$$

where $Dec_{H,\beta,\epsilon}$ *is the gadget decomposition described in Algorithm 1.*

The formula is almost identical to the classical product defined in the original GSW scheme in [GSW13], except that only one vector needs to be decomposed. For this reason, the following theorem shows that we get almost the same noise propagation formula, with an additional term that comes from the approximations in the decomposition.

Theorem 3.2.1 (Worst-case External Product)**.** *Let* A *be a valid* TGSW *sample of message* μ_A *and let* $\mathbf{b}$ *be a valid* TLWE *sample of message* $\mu_{\mathbf{b}}$*. Then* $A \boxdot \mathbf{b}$ *is a* TLWE *sample of message* $\mu_A \cdot \mu_{\mathbf{b}}$ *and*

$$\|\mathsf{Err}(A \boxdot \mathbf{b})\|_\infty \leq (k+1)\ell N\beta \|\mathsf{Err}(A)\|_\infty + \|\mu_A\|_1(1+kN)\epsilon + \|\mu_A\|_1\|\mathsf{Err}(\mathbf{b})\|_\infty$$

in the worst case, where β *and* ϵ *are the parameters used in the decomposition* $Dec_{\mathbf{h},\beta,\epsilon}(\mathbf{b})$*. If* $\|\mathsf{Err}(A \boxdot \mathbf{b})\|_\infty \leq 1/4$ *we are guaranteed that* $A \boxdot \mathbf{b}$ *is a valid* TLWE *sample.*

Proof. As $A = \mathsf{TGSW}(\mu_A)$, then by definition it is equal to $A = Z_A + \mu_A \cdot H$, where Z_A is a TGSW encryption of 0 and H is the gadget matrix. In the same way, as $\mathbf{b} = \mathsf{TLWE}(\mu_{\mathbf{b}})$, then by definition it is equal to $\mathbf{b} = \mathbf{z_b} + (\mathbf{0}, \mu_{\mathbf{b}})$, where $\mathbf{z_b}$ is a TLWE encryption of 0. Let

$$\begin{cases} \|\mathsf{Err}(A)\|_\infty = \|\varphi_{\mathbf{s}}(Z_A)\|_\infty = \eta_A \\ \|\mathsf{Err}(\mathbf{b})\|_\infty = \|\varphi_{\mathbf{s}}(\mathbf{z_b})\|_\infty = \eta_{\mathbf{b}}. \end{cases}$$

Let $\mathbf{u} = Dec_{H,\beta,\epsilon}(\mathbf{b}) \in \mathbb{Z}_N[X]^{(k+1)\ell}$. By definition $A \boxdot \mathbf{b}$ is equal to

$$\begin{aligned} A \boxdot \mathbf{b} &= \mathbf{u} \cdot A \\ &= \mathbf{u} \cdot Z_A + \mu_A \cdot (\mathbf{u} \cdot H). \end{aligned}$$

From Lemma 3.2.1, we have that $\mathbf{u} \cdot H = \mathbf{b} + \epsilon_{\mathbf{dec}}$, where $\|\epsilon_{\mathbf{dec}}\|_\infty = \|\mathbf{u} \cdot H - \mathbf{b}\|_\infty \leq \epsilon$. So

$$\begin{aligned} A \boxdot \mathbf{b} &= \mathbf{u} \cdot Z_A + \mu_A \cdot (\mathbf{b} + \epsilon_{\mathbf{dec}}) \\ &= \mathbf{u} \cdot Z_A + \mu_A \cdot \epsilon_{\mathbf{dec}} + \mu_A \cdot \mathbf{z_b} + (\mathbf{0}, \mu_A \cdot \mu_{\mathbf{b}}). \end{aligned}$$

Then the phase (linear function) of $A \boxdot \mathbf{b}$ is

$$\varphi_{\mathbf{s}}(A \boxdot \mathbf{b}) = \mathbf{u} \cdot \mathsf{Err}(A) + \mu_A \cdot \varphi_{\mathbf{s}}(\epsilon_{\mathbf{dec}}) + \mu_A \cdot \mathsf{Err}(\mathbf{b}) + \mu_A \mu_{\mathbf{b}}.$$

Taking the expectation, we get that $\mathsf{msg}(A \boxdot \mathbf{b}) = 0 + 0 + 0 + \mu_A \mu_{\mathbf{b}}$, and so $\mathsf{Err}(A \boxdot \mathbf{b}) = \varphi_{\mathbf{s}}(A \boxdot \mathbf{b}) - \mu_A \mu_{\mathbf{b}}$. Then thanks to Remark 3.2.3, we have

$$\begin{aligned} \|\mathsf{Err}(A \boxdot \mathbf{b})\|_\infty &\leq \|\mathbf{u} \cdot \mathsf{Err}(A)\|_\infty + \|\mu_A \cdot \varphi(\epsilon_{\mathbf{dec}})\|_\infty + \|\mu_A \cdot \mathsf{Err}(\mathbf{b})\|_\infty \\ &\leq (k+1)\ell N \beta \eta_A + \|\mu_A\|_1 (1 + kN) \|\epsilon_{\mathbf{dec}}\|_\infty + \|\mu_A\|_1 \eta_{\mathbf{b}}. \end{aligned}$$

The result follows. □

We similarly obtain the more realistic average-case noise propagation, based on the independence heuristic Assumption 3.2.1, by bounding the Gaussian variance instead of the amplitude.

Corollary 3.2.1 (Average-case External Product)**.** *Under the same conditions of Theorem 3.2.1 and under Heuristic 3.2.1, we have that*

$$\mathsf{Var}(\mathsf{Err}(A \boxdot \mathbf{b})) \leq (k+1)\ell N \beta^2 \mathsf{Var}(\mathsf{Err}(A)) + (1 + kN)\|\mu_A\|_2^2 \epsilon^2 + \|\mu_A\|_2^2 \mathsf{Var}(\mathsf{Err}(\mathbf{b})).$$

Proof. Let $\vartheta_A = \mathsf{Var}(\mathsf{Err}(A)) = \mathsf{Var}(\varphi_{\mathbf{s}}(Z_A))$ and $\vartheta_{\mathbf{b}} = \mathsf{Var}(\mathsf{Err}(\mathbf{b})) = \mathsf{Var}(\varphi_{\mathbf{s}}(\mathbf{z_b}))$. By using the same notations as in the proof of theorem 3.2.1 we have that the error of $A \boxdot \mathbf{b}$ is $\mathsf{Err}(A \boxdot \mathbf{b}) = \mathbf{u} \cdot \mathsf{Err}(A) + \mu_A \cdot \varphi_{\mathbf{s}}(\epsilon_{\mathbf{dec}}) + \mu_A \cdot \mathsf{Err}(\mathbf{b})$ and thanks to Assumption 3.2.1 and Remark 3.2.3, we have :

$$\begin{aligned} \mathsf{Var}(\mathsf{Err}(A \boxdot \mathbf{b})) &\leq \mathsf{Var}(\mathbf{u} \cdot \mathsf{Err}(A))) + \mathsf{Var}(\mu_A \cdot \varphi(\epsilon_{\mathbf{dec}})) + \mathsf{Var}(\mu_A \cdot \mathsf{Err}(\mathbf{b})) \\ &\leq (k+1)\ell N \beta^2 \vartheta_A + (1 + kN)\|\mu_A\|_2^2 \epsilon^2 + \|\mu_A\|_2^2 \vartheta_{\mathbf{b}}. \end{aligned}$$

□

The last corollary describes exactly the classical internal product between two TGSW samples, already presented in [GSW13, AP14, GINX14, DM15] with adapted notations. As we mentioned before, it consists in $(k+1)\ell$ independent computations of the external product $\boxdot$. As for the external product, we analyze the noise growth in both worst and average case.

Corollary 3.2.2 (Internal Product)**.** *Let the product*

$$\boxtimes\colon \mathsf{TGSW} \times \mathsf{TGSW} \longrightarrow \mathsf{TGSW}$$

$$(A, B) \longmapsto A \boxtimes B = \begin{bmatrix} A \boxdot \mathbf{b_1} \\ \vdots \\ A \boxdot \mathbf{b_{(k+1)\ell}} \end{bmatrix} = \begin{bmatrix} Dec_{H,\beta,\epsilon}(\mathbf{b_1}) \cdot A \\ \vdots \\ Dec_{H,\beta,\epsilon}(\mathbf{b_{(k+1)\ell}}) \cdot A \end{bmatrix},$$

with A and B two valid TGSW *samples of messages μ_A and μ_B respectively and $\mathbf{b_i}$ corresponding to the i-th line of B. Then $A \boxtimes B$ is a* TGSW *sample of message $\mu_A \cdot \mu_B$ and*

$$\|\mathsf{Err}(A \boxdot B)\|_\infty \leq (k+1)\ell N\beta \|\mathsf{Err}(A)\|_\infty + \|\mu_A\|_1(1+kN)\epsilon + \|\mu_A\|_1\|\mathsf{Err}(B)\|_\infty$$

in the worst case. If $\|\mathsf{Err}(A \boxdot B)\|_\infty \leq 1/4$ we are guaranteed that $A \boxdot B$ is a valid TGSW *sample.*
Furthermore, by assuming the heuristic 3.2.1, we have that

$$\mathsf{Var}(\mathsf{Err}(A \boxdot B)) \leq (k+1)\ell N\beta^2 \mathsf{Var}(\mathsf{Err}(A)) + (1+kN)\|\mu_A\|_2^2\epsilon^2 + \|\mu_A\|_2^2 \mathsf{Var}(\mathsf{Err}(\mathbf{b}))$$

in the average case.

Proof. Let A and B be two TGSW samples, and μ_A and μ_B their message. Let $\mathbf{h_i}$ denote the i-th row of the gadget matrix H. By definition, the i-th row of B encodes $\mu_B \cdot \mathbf{h_i}$, so the i-th row of $A \boxtimes B$ encodes $(\mu_A\mu_B) \cdot \mathbf{h_i}$. This proves that $A \boxtimes B$ encodes $\mu_A\mu_B$. Since the internal product $A \boxtimes B$ consists in $(k+1)\ell$ independent runs of the external products $A \boxdot \mathbf{b_i}$, the noise propagation formula directly follows from Theorem 3.2.1 and Corollary 3.2.1. □

3.2.4 CMux gate

With the homomorphic operations described until now, i.e. the linear combinations and the products, it is possible to construct small homomorphic circuits. To ease these constructions, we now define the controlled selector gate (or `CMux` gate, where `C` stands for controlled), which can be considered as the bridge between the external product arithmetic, and high level circuits. The classical `MUX` gate (Figure 3.5) has 3 input bits, c, d_0 and d_1, and it selects one value between d_0 and d_1 depending on the value of the selector bit c. It is generally noted by the formula $c?d_1{:}d_0$.
The `CMux` gate (Figure 3.6) is the homomorphic version of the traditional `MUX` gate. As this latter, it has three input slots and one output slot: one *control input* slot

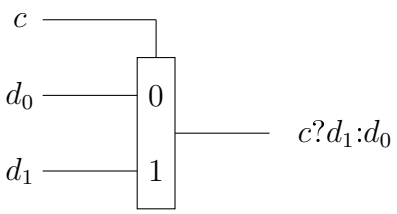

Figure 3.5: MUX gate - *depending on the value of the selector bit c, it chooses between* d_0 *and* d_1.

represented by a TGSW sample on the integer message space (here restricted to $\{0,1\}$), two *data input* slots each carrying a TLWE sample on the continuous message space $\mathbb{T}_N[X]$, and one *data output* slot, also of type TLWE. The controlled MUX gate $\texttt{CMux}(C, \mathbf{d_1}, \mathbf{d_0})$ homomorphically outputs either the message of $\mathbf{d_1}$ or $\mathbf{d_0}$ depending on the Boolean value in C, without decrypting any of the three cipertexts. In practice, it returns

$$\texttt{CMux}(C, \mathbf{d_1}, \mathbf{d_0}) = C \boxdot (\mathbf{d_1} - \mathbf{d_0}) + \mathbf{d_0}.$$

In leveled circuits, the rule to build valid circuits using CMux gates is that all control wires (TGSW) are freshly generated by the user, and the data input ports of our gates can be either freshly generated or connected to a data output of another gate. Observe that, if we use the internal product instead of the external one in the construction of CMux gates, they are composable (inputs and outputs are all TGSW ciphertexts). On the other hand, they are slower than external product CMux gates and the noise reduction via bootstrapping on TGSW ciphertexts is very costly. Instead, in Chapter 5, we propose an efficient way to transform a TLWE sample in a TGSW sample, in order to make the circuits entirely composable (and so relax the condition requiring freshly generated control wires).
In the following, unless specified, the CMux gates we use are constructed only with the external product.
We now analyze the noise growth after the evaluation of a CMux gate.

Lemma 3.2.2 (CMux gate)**.** *Let* $\mathbf{d_0}, \mathbf{d_1} \in \mathsf{TLWE}_{\mathbf{s}}(\mathbb{T}_N[X])$ *and* $C \in \mathsf{TGSW}_{\mathbf{s}}(\{0,1\})$. *Then* $\mathsf{msg}(\texttt{CMux}(C, \mathbf{d_1}, \mathbf{d_0})) = \mathsf{msg}(C)?\mathsf{msg}(\mathbf{d_1}):\mathsf{msg}(\mathbf{d_0})$. *Furthermore*

- $\|\mathsf{Err}(\texttt{CMux}(C, \mathbf{d_1}, \mathbf{d_0}))\|_\infty \leq \max(\|\mathsf{Err}(\mathbf{d_0})\|_\infty, \|\mathsf{Err}(\mathbf{d_1})\|_\infty) + \eta(C)$,
- $\mathsf{Var}(\mathsf{Err}(\texttt{CMux}(C, \mathbf{d_1}, \mathbf{d_0}))) \leq \max(\mathsf{Var}(\mathsf{Err}(\mathbf{d_0})), \mathsf{Var}(\mathsf{Err}(\mathbf{d_1}))) + \vartheta(C)$, *in the conditions of Assumption 3.2.1,*

where $\eta(C) = (k+1)\ell N\beta\|\mathsf{Err}(C)\|_\infty + (kN+1)\epsilon$ *and* $\vartheta(C) = (k+1)\ell N\beta^2\mathsf{Var}(\mathsf{Err}(C)) + (kN+1)\epsilon^2$.

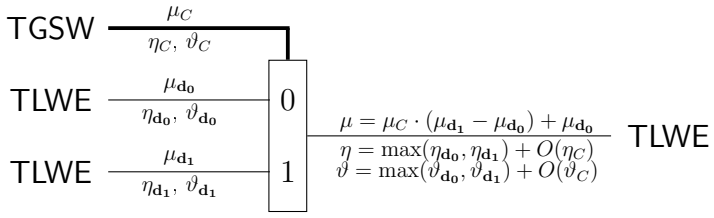

Figure 3.6: `CMux` gate - *The* `CMux` *gate takes in input a* TGSW *sample* C *with message* μ_C*, a* TLWE *sample* $\mathbf{d_0}$ *with message* $\mu_{\mathbf{d_0}}$ *and a* TLWE *sample* $\mathbf{d_1}$ *with message* $\mu_{\mathbf{d_1}}$*. It outputs a* TLWE *sample with message* $\mu = \mu_C \cdot (\mu_{\mathbf{d_1}} - \mu_{\mathbf{d_0}}) + \mu_{\mathbf{d_0}}$*. The* η*'s and* ϑ*'s represent respectively the noise in the worst and average case, for both the inputs and the output.*

Proof. The formulas for the noise in the worst and average cases are a consequence of Theorem 3.2.1 and Corollary 3.2.1. However, we need to explain why there is a max instead of the sum we would obtain by blindly applying these results. Let $\mathbf{d} = \mathbf{d_1} - \mathbf{d_0}$, recall that in the proof of Theorem 3.2.1, the expression of $C \boxdot \mathbf{d}$ is $\mathrm{Dec}_{H,\beta,\epsilon}(\mathbf{d}) \cdot Z_C + \mu_C \epsilon_{\mathrm{dec}} + \mu_C \mathbf{z_d} + (\mathbf{0}, \mu_C \cdot \mu_d)$, where $C = Z_C + \mu_C \cdot H$ and $\mathbf{d} = \mathbf{z_d} + (\mathbf{0}, \mu_d)$, Z_C and $\mathbf{z_d}$ are respectively TGSW and TLWE samples of 0, and $\|\epsilon_{\mathrm{dec}}\|_\infty \leq \epsilon$. Thus, $\texttt{CMux}(C, \mathbf{d_1}, \mathbf{d_0})$ is the sum of four terms:

- $\mathrm{Dec}_{H,\beta,\epsilon}(\mathbf{d}) \cdot Z_C$ of norm $\leq (k+1)\ell N \beta \eta_C$;
- $\mu_C \epsilon_{\mathrm{dec}}$ of norm $\leq (kN+1)\epsilon$;
- $z_{d_0} + \mu_C(z_{d_1} - z_{d_0})$, which is either z_{d_1} or z_{d_0}, depending on the value of μ_C;
- $(\mathbf{0}, \mu_{d_0} + \mu_C \cdot (\mu_{d_1} - \mu_{d_0}))$, which is the trivial sample of the output message $\mu_C ? \mu_{d_1} : \mu_{d_0}$, and is not part of the noise.

Thus, summing the three terms concludes the proof. For the average case, the formula is proven in the same way by using the results of Corollary 3.2.1 and replacing all norm inequalities by variance inequalities. □

Notations

The goal of this chapter was to be as general as possible, so we used the notations TLWE and TGSW to include all the instances and prove their properties once and for all. In the rest of the manuscript instead, the notation TLWE is used to denote the canonical scalar TLWE problem (i.e. the scale invariant LWE problem from Definition 3.2.1). To distinguish it from the Ring mode, we introduce the notation TRLWE. The TGSW samples are only used in ring mode, but we use the notation TRGSW to keep uniformity with the TRLWE notation.

In order to explicit and keep track of the techniques we introduce all along the manuscript, we use Figure 3.7 and we update it every time a new technique is described. The figure will represent all the morphisms and operations that can be performed between the different plaintext spaces. From a high level prospective, all these plaintext operations have an homomorphic equivalent over ciphertexts. This way, it is enough to reason on the plaintext to design complex homomorphic applications. The small circles placed on the bottom right of the message spaces indicate the internal operations that can be performed inside the specific message spaces. The arrows instead indicate the operations between different plaintext spaces.

	Message	Ciphertext	Secret key
TLWE	$\mathbb{T}$	$\mathbb{T}^{n+1}$	$\mathbb{B}^n$
TRLWE	$\mathbb{T}_N[X]$	$\mathbb{T}_N[X]^{k+1}$	$\mathbb{B}_N[X]^k$
TRGSW	$\mathbb{Z}_N[X]$	$((k+1)\ell)$-vector of TRLWE	$\mathbb{B}_N[X]^k$

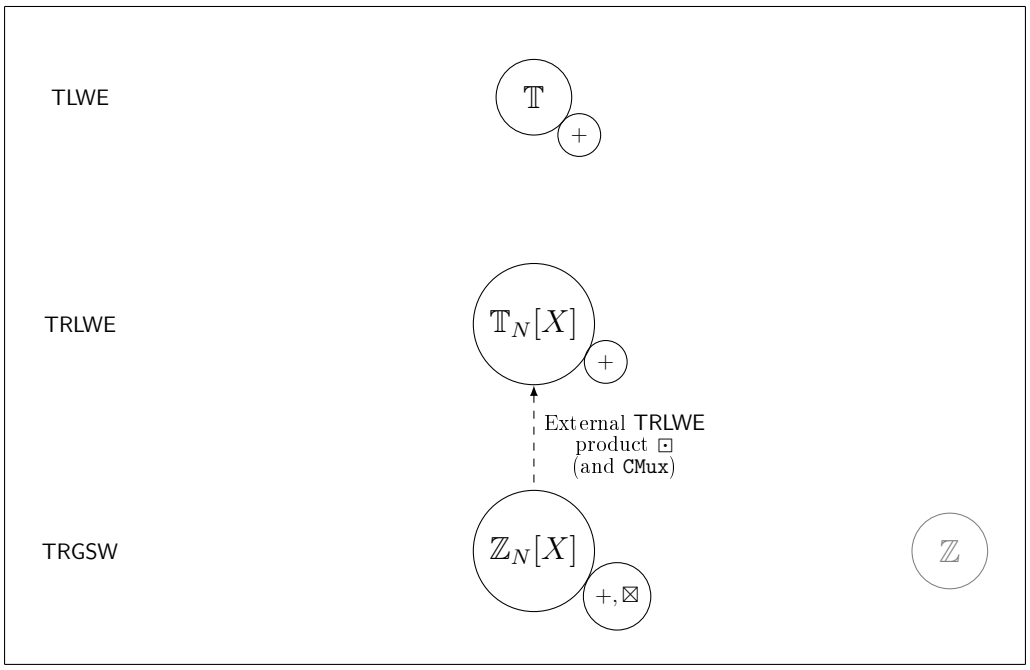

Figure 3.7: *Homomorphic operations: view from the message spaces.*

Furthermore, we distinguish the TLWE keys from the TRLWE keys by using the respective notations $\mathfrak{K}$ and K, instead of the generic $\mathbf{s}$ used until now. We also use the following convention in the rest of the manuscript: for all $n = kN$, a binary vector $\mathfrak{K} \in \mathbb{B}^n$ can be interpreted as a TLWE key, or alternatively as a TRLWE key $K \in \mathbb{B}_N[X]^k$ having the same sequence of coefficients.
Namely, K_i is the polynomial $\sum_{j=0}^{N-1} \mathfrak{K}_{N(i-1)+j+1} X^j$. In this case, we say that K is the TRLWE interpretation of $\mathfrak{K}$, and $\mathfrak{K}$ is the TLWE interpretation of K:

Figure 3.8: *Double interpretation of the keys K and $\mathfrak{K}$: the example shows the* TRLWE *key polynomial K_1 as a container for the first N bits of the* TLWE *key $\mathfrak{K}$.*

TLWE key		TRLWE key
$\mathfrak{K} \in \mathbb{B}^n$	$\sim$	$K \in \mathbb{B}_N[X]^k$
$(\mathfrak{K}_1, \ldots, \mathfrak{K}_n)$		$(K_1, \ldots, K_k)$ such that $K_i = \sum_{j=0}^{N-1} \mathfrak{K}_{N(i-1)+j+1} X^j$

This interpretation is quite natural, as polynomials can be seen as coefficient containers (Figure 3.8). The same idea is also used and better detailed in following chapters.

Chapter 4

TFHE: building blocks and leveled constructions

In the previous chapters, we introduced the bases for the construction of TFHE, by re-formalizing and generalizing LWE and GSW over the torus (TLWE and TGSW). We also described the basic operations (linear combinations, products and the `CMux` homomorphic gate) that can be homomorphically performed on ciphertexts. In this chapter, we construct at first some building blocks, starting from these basic operations, and then we explain how to use them to build more complex leveled homomorphic constructions.

In our construction, we redefine the concept of *levels*. In previous works, the number of levels corresponded in general to the multiplicative depth of the function/circuit to be homomorphically evaluated. In our case, the number of levels corresponds instead to the number of compositions of automata.

4.1 Building blocks for TFHE

In this section we describe three homomorphic building blocks, constructed by using the basic operations described in Chapter 3.
The first building block is the *key switching*. This technique has been introduced for the second generation of homomorphic schemes, and largely used in the following. The key switching we describe is a re-visitation of the original technique. We observe that it can be used not only to switch from a key to another in different parameter sets, but also to homomorphically evaluate lipschitzian morphisms of $\mathbb{Z}$-modules $f : \mathbb{T}^p \to \mathbb{T}_N[X]$. Furthermore, the morphisms can be public or private, which makes us distinguish two kinds of different key switchings.
The second building block is the sample *extraction*: TRLWE ciphertexts encrypt torus polynomials and they can be seen as homomorphic containers. The sample extraction is the operation allowing to extract one of the coefficients from the container. It can

be seen also as the inverse operation to the key switching.
The third building block we describe is the *blind rotation*. As the name says, it consists in a rotation of the coefficients inside the TRLWE container by an encrypted amount of positions. This can be done thanks to the external product and the `CMux` gate.
Each one of them is described in detail in next sections.

4.1.1 Key Switching revisited

We revisit the well-known key-switching procedure, largely described in the literature. The principal interest of key switching, as the name suggests, is to switch between keys in different parameter sets.
We show that this procedure has a larger potential. It allows to switch between the scalar and polynomial message spaces $\mathbb{T}$ and $\mathbb{T}_N[X]$, and more generally, it has the ability to homomorphically evaluate linear morphisms f from any $\mathbb{Z}$-module $\mathbb{T}^p$ to $\mathbb{T}_N[X]$. We define two key-switching flavors, one for a publicly known f, and one for a secret f encoded in the key-switching key.
In the following, we denote $\mathsf{PublicKS}(f, \mathsf{KS}, \mathfrak{c})$ and $\mathsf{PrivateKS}(\mathsf{KS}^{(f)}, \mathfrak{c})$ the output of Algorithm 2 and Algorithm 3, taking in input the functional key-switching keys KS and $\mathsf{KS}^{(f)}$ respectively and a TLWE ciphertext $\mathfrak{c}$.
As the inputs and the outputs are instantiated with different parameter sets and we want to keep the same name for the variables $n, N, \ell, B_g, \ldots$, we add an under bar to the output parameters to distinguish them from the input parameters.
From now on, we use the letter γ to indicate the standard deviation of the key-switching key, instead of the letter σ. The variable t represents the precision of the binary decomposition.

Algorithm 2 TLWE-to-T(R)LWE Public Functional Key Switching

Input: p TLWE ciphertexts $\mathfrak{c}^{(z)} = (\mathfrak{a}^{(z)}, \mathfrak{b}^{(z)}) \in \mathsf{TLWE}_{\mathfrak{K}}(\mu_z)$ for $z = 1, \ldots, p$, a public R-lipschitzian morphism $f : \mathbb{T}^p \to \mathbb{T}_{\underline{N}}[X]$, and $KS_{i,j} \in \mathsf{T(R)LWE}_{\underline{K}}(\frac{\mathfrak{K}_i}{2^j})$.
Output: A T(R)LWE sample $\mathbf{c} \in \mathsf{T(R)LWE}_{\underline{K}}(f(\mu_1, \ldots, \mu_p))$

1: **for** $i \in [\![1, n]\!]$ **do**
2: Let $a_i = f(\mathfrak{a}_i^{(1)}, \ldots, \mathfrak{a}_i^{(p)})$
3: Let $\tilde{a}_i$ be the closest multiple of $\frac{1}{2^t}$ to a_i, thus $\|\tilde{a}_i - a_i\|_\infty < 2^{-(t+1)}$
4: Binary decompose each $\tilde{a}_i = \sum_{j=1}^{t} \tilde{a}_{i,j} \cdot 2^{-j}$ where $\tilde{a}_{i,j} \in \mathbb{B}_{\underline{N}}[X]$
5: **end for**
6: **return** $(0, f(\mathfrak{b}^{(1)}, \ldots, \mathfrak{b}^{(p)})) - \sum_{i=1}^{n} \sum_{j=1}^{t} \tilde{a}_{i,j} \cdot \mathsf{KS}_{i,j}$

Theorem 4.1.1. *(Public Key Switching) Given p* TLWE *ciphertexts $\mathfrak{c}^{(z)} \in \mathsf{TLWE}_{\mathfrak{K}}(\mu_z)$, a public R-lipschitzian morphism $f : \mathbb{T}^p \to \mathbb{T}_{\underline{N}}[X]$ of $\mathbb{Z}$-modules, and*

$\mathsf{KS}_{i,j} \in \mathsf{T(R)LWE}_{\underline{K},\underline{\gamma}}(\frac{\mathfrak{K}_i}{2^j})$ *with standard deviation* $\underline{\gamma}$*, Algorithm 2 outputs a* $\mathsf{T(R)LWE}$ *sample* $\mathbf{c} \in \mathsf{T(R)LWE}_{\underline{K}}(f(\mu_1, \ldots, \mu_p))$ *such that:*

- $\|\mathsf{Err}(\mathbf{c})\|_\infty \leq R\|\mathsf{Err}(\mathfrak{c})\|_\infty + nt\underline{N}\underline{\mathcal{A}}_{\mathsf{KS}} + n2^{-(t+1)}$ *(worst case),*
- $\mathsf{Var}(\mathsf{Err}(\mathbf{c})) \leq R^2\mathsf{Var}(\mathsf{Err}(\mathfrak{c})) + nt\underline{N}\underline{\vartheta}_{\mathsf{KS}} + n2^{-2(t+1)}$ *(average case),*

where $\underline{\mathcal{A}}_{\mathsf{KS}}$ *and* $\underline{\vartheta}_{\mathsf{KS}} = \underline{\gamma}^2$ *are respectively the amplitude and the variance of the error of* KS*.*

Proof. Let $\mathbf{c}$ be the output of Algorithm 2 and $b = f(\mathfrak{b}^{(1)}, \ldots, \mathfrak{b}^{(p)})$ then

$$
\begin{aligned}
\varphi_{\underline{K}}(\mathbf{c}) &= b - \sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j} \cdot \varphi_{\underline{K}}(\mathsf{KS}_{i,j}) \\
&= b - \sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j}(\frac{\mathfrak{K}_i}{2^j} - \mathsf{Err}(\mathsf{KS}_{i,j})) \\
&= b - \sum_{i=1}^{n} \mathfrak{K}_i\tilde{a}_i - \sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j}\mathsf{Err}(\mathsf{KS}_{i,j}) \\
&= b - \sum_{i=1}^{n} \mathfrak{K}_i a_i - \sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j}\mathsf{Err}(\mathsf{KS}_{i,j}) + \sum_{i=1}^{n} \mathfrak{K}_i \cdot (a_i - \tilde{a}_i) \\
&= f(\mathfrak{b}^{(1)}, \ldots, \mathfrak{b}^{(p)}) - \sum_{i=1} f(\mathfrak{a}_i^{(1)}, \ldots, \mathfrak{a}_i^{(p)})\mathfrak{K}_i \\
&\quad - \sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j}\mathsf{Err}(\mathsf{KS}_{i,j}) + \sum_{i=1}^{n} \mathfrak{K}_i \cdot (a_i - \tilde{a}_i) \\
&= f\left((\mathfrak{b}^{(1)}, \ldots, \mathfrak{b}^{(p)}) - \sum_{i=1}^{n} \mathfrak{K}_i(\mathfrak{a}_i^{(1)}, \ldots, \mathfrak{a}_i^{(p)})\right) \\
&\quad - \sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j}\mathsf{Err}(\mathsf{KS}_{i,j}) + \sum_{i=1}^{n} \mathfrak{K}_i \cdot (a_i - \tilde{a}_i) \\
&= f(\varphi_{\mathfrak{K}}(\mathfrak{c}^{(1)}), \ldots, \varphi_{\mathfrak{K}}(\mathfrak{c}^{(p)})) - \sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j}\mathsf{Err}(\mathsf{KS}_{i,j}) + \sum_{i=1}^{n} \mathfrak{K}_i \cdot (a_i - \tilde{a}_i)
\end{aligned}
$$

Applying the expectation on each side, we obtain $\mathsf{msg}(\mathbf{c})$ on the left, and $f(\mu_1, \ldots, \mu_p)$ on the right, since all the error terms have expectation 0 and f is linear. For the worst-case bound, we obtain that:

$$
\begin{aligned}
\|\mathsf{Err}(\mathbf{c})\|_\infty &= \|\varphi_{\underline{K}}(\mathbf{c}) - \mathsf{msg}(\mathbf{c})\|_\infty \\
&\leq \|f(\mathsf{Err}(\mathfrak{c}^{(1)}), \ldots, \mathsf{Err}(\mathfrak{c}^{(p)}))\|_\infty + nt\underline{N}\underline{\mathcal{A}}_{\mathsf{KS}} + \underline{N}n2^{-(t+1)} \\
&\leq R\|\mathsf{Err}(\mathbf{c})\|_\infty + nt\underline{N}\underline{\mathcal{A}}_{\mathsf{KS}} + \underline{N}n2^{-(t+1)}
\end{aligned}
$$

since f is R-lipschitzian. For the average-case, we have a similar proof:

$$\begin{aligned}\mathsf{Var}(\mathsf{Err}(\mathbf{c})) &= \mathsf{Var}(\varphi_{\underline{K}}(\mathbf{c}) - \mathsf{msg}(\mathbf{c})) \\ &\leq \mathsf{Var}(f(\mathsf{Err}(\mathfrak{c}^{(1)}), \dots, \mathsf{Err}(\mathfrak{c}^{(p)}))) + nt\underline{N}\vartheta_{\mathsf{KS}} + \underline{N}n2^{-2(t+1)} \\ &\leq R^2\mathsf{Var}(\mathsf{Err}(\mathbf{c})) + nt\underline{N}\vartheta_{\mathsf{KS}} + \underline{N}n2^{-2(t+1)}.\end{aligned}$$

□

Remark 4.1.1. *The* TLWE*-to-*T(R)LWE *public key-switching procedure we described, allows to switch between the scalar message space* $\mathbb{T}$ *and the polynomial message space* $\mathbb{T}_N[X]$*. The same procedure can be used to perform a* TLWE*-to-*TLWE *public key switching and switch between scalar message spaces. Here is why we put parentheses around the* R *of* T(R)LWE*. In practice, the key-switching key is composed by* TLWE *encryptions of the old secret key, and the noise growth formulas remain the same, with the factor* N *equal to* 1.
Furthermore, observe that the public key-switching procedure can be used from TRLWE *to* TRLWE*: in this case the function* f *is just the identity function.*

We have a similar result when the function is private. In this algorithm, we extend the input secret key $\mathfrak{K}$ by adding a $(n+1)$-th coefficient equal to -1, so that $\varphi_{\mathfrak{K}}(\mathfrak{c}) = -\mathfrak{K} \cdot \mathfrak{c}$.

Algorithm 3 TLWE-to-T(R)LWE Private Functional Key Switching

Input: p TLWE ciphertexts $\mathfrak{c}^{(z)} \in \mathsf{TLWE}_{\mathfrak{K}}(\mu_z)$, a key-switching key $\mathsf{KS}^{(f)}_{z,i,j} \in \mathsf{T(R)LWE}_{\underline{K}}(f(0, \dots, 0, \frac{\mathfrak{K}_i}{2^j}, 0, \dots, 0))$ where $f : \mathbb{T}^p \to \mathbb{T}_{\underline{N}}[X]$ is a secret R-lipschitzian morphism and $\frac{\mathfrak{K}_i}{2^j}$ is at position z (also, $\mathfrak{K}_{n+1} = -1$ by convention).
Output: A T(R)LWE sample $\mathbf{c} \in \mathsf{T(R)LWE}_{\underline{K}}(f(\mu_1, \dots, \mu_p))$.

1: **for** $i \in [\![1, n+1]\!]$, $z \in [\![1, p]\!]$ **do**
2: Let $\tilde{c}_i^{(z)}$ be the closest multiple of $\frac{1}{2^t}$ to $\mathfrak{c}_i^{(z)}$, thus $|\tilde{c}_i^{(z)} - \mathfrak{c}_i^{(z)}| < 2^{-(t+1)}$
3: Binary decompose each $\tilde{c}_i^{(z)} = \sum_{j=1}^{t} \tilde{c}_{i,j}^{(z)} \cdot 2^{-j}$ where $\tilde{c}_{i,j}^{(z)} \in \{0, 1\}$
4: **end for**
5: **return** $-\sum_{z=1}^{p} \sum_{i=1}^{n+1} \sum_{j=1}^{t} \tilde{c}_{i,j}^{(z)} \cdot \mathsf{KS}^{(f)}_{z,i,j}$

Theorem 4.1.2. *(Private Key Switching) Given* p TLWE *ciphertexts* $\mathfrak{c}^{(z)} \in \mathsf{TLWE}_{\mathfrak{K}}(\mu_z)$*, and* $\mathsf{KS}^{(f)}_{i,j} \in \mathsf{T(R)LWE}_{\underline{K},\underline{\gamma}}(f(0, \dots, \frac{\mathfrak{K}_i}{2^j}, \dots, 0))$ *where* $f : \mathbb{T}^p \to \mathbb{T}_{\underline{N}}[X]$ *is a private R-lipschitzian morphism of* $\mathbb{Z}$*-modules, Algorithm 3 outputs a* T(R)LWE *sample* $\mathbf{c} \in \mathsf{T(R)LWE}_{\underline{K}}(f(\mu_1, \dots, \mu_p))$ *such that:*

- $\|\mathsf{Err}(\mathbf{c})\|_\infty \leq R\|\mathsf{Err}(\mathfrak{c})\|_\infty + (n+1)R2^{-(t+1)} + pt(n+1)\underline{\mathcal{A}}_{\mathsf{KS}}$ *(worst-case),*
- $\mathsf{Var}(\mathsf{Err}(\mathbf{c})) \leq R^2\mathsf{Var}(\mathsf{Err}(\mathfrak{c})) + (n+1)R^2 2^{-2(t+1)} + pt(n+1)\underline{\vartheta}_{\mathsf{KS}}$ *(average case),*

where $\underline{\mathcal{A}}_{\mathsf{KS}}$ *and* $\underline{\vartheta}_{\mathsf{KS}} = \underline{\gamma}^2$ *are respectively the amplitude and the variance of the error of* $\mathsf{KS}^{(f)}$.

Proof. Let $\mathbf{c}$ be the output of Algorithm 3 and $b = f(\mathfrak{b}^{(1)}, \ldots, \mathfrak{b}^{(p)})$ then:

$$\begin{aligned}
\varphi_{\underline{K}}(\mathbf{c}) &= -\sum_{z=1}^{p}\sum_{i=1}^{n+1}\sum_{j=1}^{t} \tilde{c}_{i,j}^{(z)} \cdot \varphi_{\underline{K}}(\mathsf{KS}_{z,i,j}^{(f)}) \\
&= -\sum_{z=1}^{p}\sum_{i=1}^{n+1}\sum_{j=1}^{t} \tilde{c}_{i,j}^{(z)} \left(f(0, \ldots, \frac{\mathfrak{K}_i}{2^j}, \ldots, 0) + \mathsf{Err}(\mathsf{KS}_{i,j}^{(f)}) \right) \\
&= -\sum_{z=1}^{p}\sum_{i=1}^{n+1}\sum_{j=1}^{t} \tilde{c}_{i,j}^{(z)} f(0, \ldots, \frac{\mathfrak{K}_i}{2^j}, \ldots, 0) - \sum_{z=1}^{p}\sum_{i=1}^{n+1}\sum_{j=1}^{t} \tilde{c}_{i,j}^{(z)} \mathsf{Err}(\mathsf{KS}_{z,i,j}^{(f)})
\end{aligned}$$

We set $\epsilon_{\mathsf{KS}} = \sum_{z=1}^{p}\sum_{i=1}^{n+1}\sum_{j=1}^{t} \tilde{c}_{i,j}^{(z)} \mathsf{Err}(\mathsf{KS}_{z,i,j}^{(f)})$. Then:

$$\begin{aligned}
&= -\sum_{i=1}^{n+1}\sum_{z=1}^{p} f(0, \ldots, \sum_{j=1}^{t} \tilde{c}_{i,j}^{(z)} \frac{\mathfrak{K}_i}{2^j}, \ldots, 0) - \epsilon_{\mathsf{KS}} \\
&= -\sum_{i=1}^{n+1}\sum_{z=1}^{p} f(0, \ldots, \mathfrak{K}_i \cdot \mathfrak{c}_i^{(z)}, \ldots, 0) \\
&\quad -\sum_{i=1}^{n+1}\sum_{z=1}^{p} f(0, \ldots, \mathfrak{K}_i \cdot (\tilde{c}_i^{(z)} - \mathfrak{c}_i^{(z)}), \ldots, 0) - \epsilon_{\mathsf{KS}} \\
&= -\sum_{i=1}^{n+1} \mathfrak{K}_i f(\mathfrak{c}_i^{(1)}, \ldots, \mathfrak{c}_i^{(z)}, \ldots, \mathfrak{c}_i^{(p)}) - \sum_{i=1}^{n+1} \mathfrak{K}_i f(\tilde{c}_i^{(1)} - \mathfrak{c}_i^{(1)}, \ldots, \tilde{c}_i^{(p)} - \mathfrak{c}_i^{(p)}) - \epsilon_{\mathsf{KS}} \\
&= f(-\sum_{i=1}^{n+1} \mathfrak{K}_i \mathfrak{c}_i^{(1)}, \ldots, -\sum_{i=1}^{n+1} \mathfrak{K}_i \mathfrak{c}_i^{(p)}) - \sum_{i=1}^{n+1} \mathfrak{K}_i f(\tilde{c}_i^{(1)} - \mathfrak{c}_i^{(1)}, \ldots, \tilde{c}_i^{(p)} - \mathfrak{c}_i^{(p)}) - \epsilon_{\mathsf{KS}} \\
&= f(\varphi_{\mathfrak{K}}(\mathfrak{c}^{(1)}), \ldots, \varphi_{\mathfrak{K}}(\mathfrak{c}^{(p)})) - \sum_{i=1}^{n+1} \mathfrak{K}_i f(\tilde{c}_i^{(1)} - \mathfrak{c}_i^{(1)}, \ldots, \tilde{c}_i^{(p)} - \mathfrak{c}_i^{(p)}) - \epsilon_{\mathsf{KS}} \\
&= f(\mu_1 + \mathsf{Err}(\mathfrak{c}^{(1)}), \ldots, \mu_p + \mathsf{Err}(\mathfrak{c}^{(p)})) \\
&\quad -\sum_{i=1}^{n+1} \mathfrak{K}_i f(\tilde{c}_i^{(1)} - \mathfrak{c}_i^{(1)}, \ldots, \tilde{c}_i^{(p)} - \mathfrak{c}_i^{(p)}) - \epsilon_{\mathsf{KS}}
\end{aligned}$$

By linearity of f and since the expectation of the error terms are 0, the message of the right side is equal to $f(\mu_1, \ldots, \mu_p)$. For the worst-case bound on the noise, as f is R-lipschitzian, we obtain:

$$\begin{aligned}
\|\mathsf{Err}(\mathbf{c})\|_\infty &= \|\varphi_{\underline{K}}(\mathbf{c}) - \mathsf{msg}(\mathbf{c})\|_\infty \\
&\leq R\|\mathsf{Err}(\mathbf{c})\|_\infty + (n+1)R2^{-(t+1)} + pt(n+1)\underline{\mathcal{A}}_{\mathsf{KS}}
\end{aligned}$$

The proof for the variance is similar. □

4.1.2 Sample Extraction.

It is known that a TRLWE message is a polynomial with N coefficients, which can be viewed as a container with N *slots* over $\mathbb{T}$, as shown in Figure 4.1.

$$\text{TRLWE message } \mu = \sum_{i=0}^{N-1} \mu_i \cdot X^i \in \mathbb{T}_N[X] \sim \mu = (\mu_0, \mu_1, \ldots, \mu_{N-1}) \in \mathbb{T}^N$$

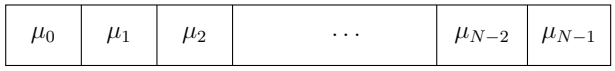

Figure 4.1: TRLWE container - *composed by N slots containing a value in $\mathbb{T}$.*

It is easy to homomorphically extract a coefficient as a scalar TLWE sample with the same key. We recall that a binary TLWE key $\mathfrak{K} \in \mathbb{B}^n$ can be interpreted as a TRLWE key $K \in \mathbb{B}_N[X]^k$ having the same sequence of coefficients, and vice-versa. Given a TRLWE sample $\mathbf{c} = (\mathbf{a}, b) \in \mathsf{TRLWE}_K(\mu)$ and a position $p \in [0, N-1]$, we call $\mathsf{SampleExtract}_p(\mathbf{c})$ the TLWE sample $(\mathfrak{a}, \mathfrak{b})$ where $\mathfrak{b} = b_p$ and $\mathfrak{a}_{N(i-1)+j+1}$ is the $(p-j)$-th coefficient of a_i (using the N-antiperiodic indexes $a_{N+i} = -a_i$). This extracted sample encodes the p-th coefficient μ_p with at most the same noise variance or amplitude as $\mathbf{c}$.

Algorithm 4 $\mathsf{SampleExtract}_p$

Input: A TRLWE ciphertexts $\mathbf{c} = (\mathbf{a}, b) \in \mathsf{TRLWE}_K(\mu)$, a position $p \in [\![0, N-1]\!]$.
Output: A TLWE sample $\mathfrak{c} = (\mathfrak{a}, \mathfrak{b}) \in \mathsf{TLWE}_{\mathfrak{K}}(\mu_p)$.

1: Set $\mathfrak{b} = b_p$
2: **for** $i \in [\![1, k]\!]$, $j \in [\![0, N-1]\!]$ **do**
3: $\quad \mathfrak{a}_{N(i-1)+j+1} = (a_i)_{p-j}$
4: **end for**
5: **return** $\mathfrak{c} = (\mathfrak{a}, \mathfrak{b})$

In the rest of the manuscript, we will simply write $\mathsf{SampleExtract}(\mathbf{c})$ when $p = 0$. In Section 4.2.1, we show how the KeySwitching and the SampleExtract procedures are used to efficiently pack data, unpack and move data across the slots, and how it differs from usual packing techniques.

4.1.3 Blind Rotate

The BlindRotate algorithm multiplies the polynomial encrypted in the input TRLWE ciphertext by an encrypted power of X. The effect produced is a rotation of the coefficients/slots inside the TRLWE container. The algorithm consists in two parts. The first one (line 1) is the rotation by a known power of X. The second one (loop

at line 2) is the rotation by a secret power of X, which is performed by using the `CMux` gate.

Algorithm 5 BlindRotate

Input: A TRLWE_K sample $\mathbf{c}$ of $v \in \mathbb{T}_N[X]$, $p+1$ integer coefficients $a_1, \ldots, a_p, b \in \mathbb{Z}/(2N\mathbb{Z})$, p TRGSW_K samples $C_1, \ldots, C_p$ of $s_1, \ldots, s_p \in \mathbb{B}$
Output: A TRLWE_K sample of $X^{-\rho} \cdot v$ where $\rho = b - \sum_{i=1}^{p} s_i.a_i \mod 2N$
1: $\mathsf{ACC} \leftarrow X^{-b} \cdot \mathbf{c}$
2: **for** $i = 1$ **to** p
3: $\quad \mathsf{ACC} \leftarrow \mathsf{CMux}(C_i, X^{a_i} \cdot \mathsf{ACC}, \mathsf{ACC})$
4: **return** ACC

Theorem 4.1.3. *Let $H \in \mathcal{M}_{(k+1)\ell,k+1}(\mathbb{T}_N[X])$ the gadget matrix and $Dec_{H,\beta,\epsilon}$ its efficient approximate gadget decomposition algorithm with quality β and precision ϵ. Let $\alpha \in \mathbb{R}_{\geq 0}$ be a standard deviation, $\mathfrak{K} \in \mathbb{B}^n$ be a* TLWE *secret key and $K \in \mathbb{B}_N[X]^k$ be its* TRLWE *interpretation. Given one sample $\mathbf{c} \in \mathsf{TRLWE}_K(\mathbf{v})$ with $\mathbf{v} \in \mathbb{T}_N[X]$, $p+1$ integers $a_1, \ldots, a_p$ and $b \in \mathbb{Z}/2N\mathbb{Z}$, and p* TRGSW *ciphertexts $C_1, \ldots, C_p$, where each $C_i \in \mathsf{TRGSW}_{K,\alpha}(s_i)$ for $s_i \in \mathbb{B}$. Algorithm 5 outputs a sample $\mathsf{ACC} \in \mathsf{TRLWE}_K(X^{-\rho} \cdot \mathbf{v})$ where $\rho = b - \sum_{i=1}^{p} s_i a_i$, such that:*

- $\|\mathsf{Err}(\mathsf{ACC})\|_\infty \leq \|\mathsf{Err}(\mathbf{c})\|_\infty + p(k+1)\ell N\beta\mathcal{A}_C + p(1+kN)\epsilon$ *(worst case),*
- $\mathsf{Var}(\mathsf{Err}(\mathsf{ACC})) \leq \mathsf{Var}(\mathsf{Err}(\mathbf{c})) + p(k+1)\ell N\beta^2\vartheta_C + p(1+kN)\epsilon^2$ *(average case),*

where $\vartheta_C = \alpha^2$ and $\mathcal{A}_C$ are the variance and amplitudes of $\mathsf{Err}(C_i)$.

Proof. Theorem 4.1.3 follows from the fact that algorithm 5 calls p times the `CMux` evaluation. □

We define $\mathsf{BlindRotate}(\mathbf{c}, (a_1, \ldots, a_p, b), (C_1, \ldots, C_p))$, the procedure described in Algorithm 5 that outputs the TRLWE sample ACC as in Theorem 4.1.3.

We can now update Figure 3.7 from last chapter with the building blocks. The updated Figure 4.2 has an additional link between the scalar and polynomial integer message space, that can be done via key switching, thanks to the fact that TRGSW ciphertexts are composed by TRLWE samples.

4.2 Leveled constructions in TFHE

Now that the basic operations and building blocks are explained, we can compose them to create more complicate constructions. Our first goal is to build efficient leveled homomorphic circuits, without any bootstrapping (the bootstrapped constructions are described in next chapter). We analyzed several techniques, and we

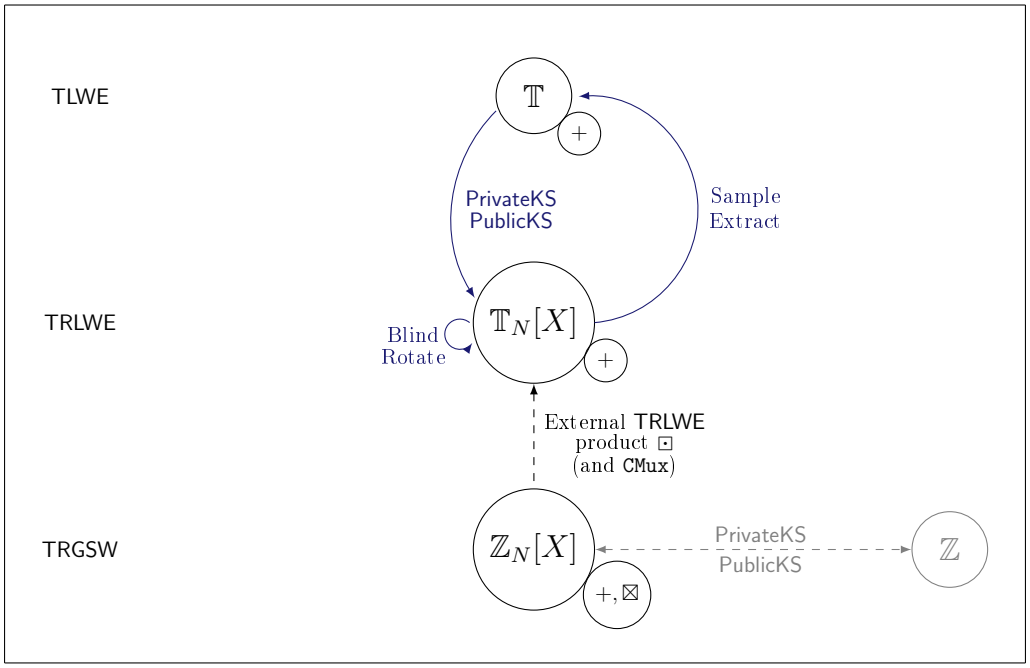

Figure 4.2: *Homomorphic operations: view from the message spaces. The new building blocks are highlighted.*

observed that there is not one "golden" technique to automatically evaluate any function. Rather, different methods exist and each one of them seems to be better than the others depending on the function to be evaluated.

In this section we describe three methods, while in Chapter 6 we explain how to use them in practice and we show our predictions on the execution timings.

The first leveled construction consists in the *evaluation of an arbitrary function via its look-up table*. In this case, we propose two packing techniques that can be used to accelerate the evaluation. Various packing techniques have already been proposed for homomorphic encryption: the Lagrange embedding in Helib [HS14, HS17], the diagonal matrices encoding in [HAO15] or the CRT encoding in [SV10, SV14, BBL17]. The message space is often a finite ring (e.g. $\mathbb{Z}/p\mathbb{Z}$), and the packing function is in general chosen as a ring isomorphism that preserves the structure of $(\mathbb{Z}/p\mathbb{Z})^N$. This way, elementary additions or products can be performed simultaneously on N independent slots, and thus, packing is in general associated to the concept of batching a single operation on multiple datasets. These techniques can have some limitations, especially if in the whole program, each function is only run on a single dataset, and most of the slots are unused. This is particularly true in the context of GSW evaluations, where functions are split into many branching algorithms or automata, that are each executed only once.

In the rest of the manuscript, packing refers to the coefficients embedding function, that maps N TLWE messages $\mu_0, \ldots, \mu_{N-1} \in \mathbb{T}$ into a single TRLWE message $\mu = \sum_{i=0}^{N-1} \mu_i X^i$, as shown in Section 4.1.2. This function is a $\mathbb{Z}$-module isomorphism. Messages can be homomorphically unpacked from any slot using the (noiseless) SampleExtract procedure, described in the same section. Reciprocally, we can repack, move data across the slots, or clear some slots by using our public functional key switching from Algorithm 2 to evaluate respectively the canonical coefficient embedding function (i.e. the identity), a permutation, or a projection. Since these functions are 1-lipschitzian, by Theorem 4.1.1, these key-switching operations only induce an additive noise overhead. It is arguably more straightforward than the permutation network technique used in Helib. But as in [BBL17, CCK+13, CLT14], our technique relies on a circular security assumption, even in the leveled mode since our key-switching key encrypts its own key bits[1].

The second leveled construction we propose is based on the *evaluation of functions via automata*. In [CGGI16a] we proposed an evaluation via deterministic finite automata (det-FA), but we discovered in [CGGI17a] that deterministic weighted finite automata (det-WFA) are more efficient. The main difference between the two techniques is, as the name announces, the presence of weights. The DFA are decisional, so each one of them outputs a single bit of information, while det-WFA are computational. The weights act like a memory that accumulates all the bits of the final result inside a TRLWE container all along the evaluation, so the result is computed in a single pass.

We give more details on all the automata logic behind those ideas and we show a few examples to better understand how to use all this in practice.

The third technique is a *new* TRLWE *homomorphic counter*, called TBSR. The T comes (again) from the torus and it is used to distinguish the homomorphic encrypted version of the counter. The BSR stands for Bit Sequence Representation and we use it to indicate the clear version of the counter, fundamental to understand the ideas before passing on the ciphertext space. The TBSR is able to perform efficiently 3 basic operations for the evaluation of arithmetic functions: extraction of a bit, increment and division by 2.

We start by explaining the plaintext construction of the BSR and then we translate the ideas in the ciphertext context by constructing and analyzing the $TBSR$.

Notice that the 3 techniques can be used independently, but they can also be combined to obtain better performances.

[1] Circular security assumption could still be avoided in leveled mode if we accept to work with many keys.

4.2.1 Arbitrary functions and Look-Up Tables

The first class of functions we analyze are arbitrary functions, very useful in multiple applications. The arbitrary functions we analyze have d-bit inputs and output a vector of s elements in the torus. The s output elements can be seen as the s independent outputs of the sub-functions $f_0, \ldots, f_{s-1} : \mathbb{B}^d \to \mathbb{T}$:

$$\begin{array}{rcl} f\colon \mathbb{B}^d & \longrightarrow & \mathbb{T}^s \\ \mathbf{x} = (x_0, \ldots, x_{d-1}) & \longmapsto & f(\mathbf{x}) = (f_0(\mathbf{x}), \ldots, f_{s-1}(\mathbf{x})). \end{array}$$

Such functions can be described with a Look-Up Table (LUT), containing the list of all the 2^d input values (each one composed by d bits) and corresponding LUT output values for the s sub-functions (1 element in $\mathbb{T}$ per sub-function f_j). We note the LUT values with $\sigma_{j,h} \in \mathbb{T}$, where $j \in [\![0, s-1]\!]$ is the sub-function index, and $h \in [\![0, 2^d - 1]\!]$ is the input index.
In order to compute the function $f(\mathbf{x}) = (f_0(\mathbf{x}), \ldots, f_{s-1}(\mathbf{x}))$, where $\mathbf{x} \in \mathbb{B}^d$, the classical way consists in evaluating the s sub-functions $f_0, \ldots, f_{s-1}$ separately, as proposed in [BV14b, CGGI16a]. Each of them consists in a binary decision tree composed by $2^d - 1$ `CMux` gates. The total complexity of the classical evaluation requires therefore to execute about $s \cdot 2^d$ `CMux` gates. Let's call $f_j(\mathbf{x}) = \sigma_{j,x} \in \mathbb{T}$ (where $x = \sum_{i=0}^{d-1} x_i 2^i$) the j-th output of $f(\mathbf{x})$, for $j = 0, \ldots, s-1$. Figure 4.3 summarizes the idea of the computation of $\sigma_{j,x}$.

x_0	$\cdots$	x_{d-1}	f_0	$\cdots$	f_{s-1}
0	$\cdots$	0	$\sigma_{0,0}$	$\cdots$	$\sigma_{s-1,0}$
1	$\cdots$	0	$\sigma_{0,1}$	$\cdots$	$\sigma_{s-1,1}$
0	$\cdots$	0	$\sigma_{0,2}$	$\cdots$	$\sigma_{s-1,2}$
1	$\cdots$	0	$\sigma_{0,3}$	$\cdots$	$\sigma_{s-1,3}$
$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
0	$\cdots$	1	$\sigma_{0,2^d-4}$	$\cdots$	$\sigma_{s-1,2^d-4}$
1	$\cdots$	1	$\sigma_{0,2^d-3}$	$\cdots$	$\sigma_{s-1,2^d-3}$
0	$\cdots$	1	$\sigma_{0,2^d-2}$	$\cdots$	$\sigma_{s-1,2^d-2}$
1	$\cdots$	1	$\sigma_{0,2^d-1}$	$\cdots$	$\sigma_{s-1,2^d-1}$

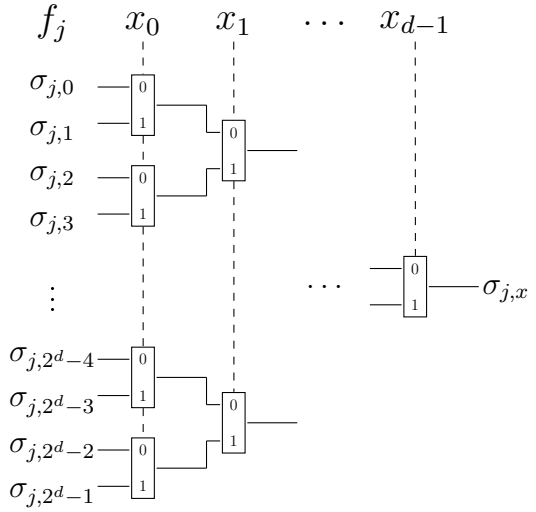

Figure 4.3: LUT with `CMux` tree - *The left part of the figure represents the LUT corresponding to the function f (and its sub-functions). The right part of the figure represents the evaluation of one sub-function f_j on $\mathbf{x} = (x_0, \ldots, x_{d-1})$ via a* `CMux` *binary decision tree.*

In the evaluation, the $\sigma_{j,h}$ are given as TRLWE samples, while the d-bits $x_0, \ldots, x_{d-1}$ are given as TRGSW ciphertexts. In this section we present two techniques, that we call horizontal and vertical packing, that can be used to improve the evaluation of a LUT. The packing technique is the same in both cases: the idea is to pack N torus values inside a single TRLWE ciphertext. The names horizontal and vertical refer to the two different ways to use such packing. Intuitively, they describe the way we pack and manipulate the LUT values in order to evaluate the function f.

Horizontal packing corresponds exactly to the classical batching. It exploits the fact that the s sub-functions evaluate the same `CMux` tree, with the same inputs but with different LUT values corresponding to the s separate truth tables. For each of the 2^d possible input values, we pack the corresponding LUT values of the s sub-functions in the first s slots (i.e. in the first s coefficients of the torus polynomial message) of a single TRLWE container (the remaining $N - s$ are unused). By using a single $(2^d - 1)$-size `CMux` tree to select the right ciphertext, we obtain the s slots containing the final result all at once, which is overall s times faster than the classical evaluation.

On the other hand, our *vertical packing* is very different from the batching techniques. The basic idea is to pack several LUT values of a single sub-function in the same ciphertext, and to use both `CMux` and blind rotations to select the desired value. Unlike batching, this can also speed up functions that have only a single bit of output.

In the following we detail these two techniques. They can be used both separately or combined, depending on the application.

Remark 4.2.1. *If the function f is public, trivial* TRLWE *samples of the LUT values $\sigma_{j,0}, \ldots, \sigma_{j,N-1}$ are used as inputs in the* `CMux` *gates. If f is private, the LUT values $\sigma_{j,0}, \ldots, \sigma_{j,N-1}$ are given encrypted. An analysis of the noise propagation in the binary decision* `CMux` *tree has already been given in [GINX14] and [CGGI16a], and can be easily retrieved by using Lemma 3.2.2.*

Horizontal Packing (or Batching)

The idea of the *horizontal packing* is to evaluate all the outputs of the function f together, instead of evaluating all the f_j separately. This is possible by using TRLWE samples, as the message space is $\mathbb{T}_N[X]$. In fact, we could encrypt up to N LUT values $\sigma_{j,h}$ (for a fixed $h \in [\![0, 2^d - 1]\!]$) per TRLWE sample and evaluate the binary decision tree as described before. Figure 4.4 illustrates this classical technique.
The number of `CMux` gates to evaluate is $\lceil \frac{s}{N} \rceil (2^d - 1)$. This technique is optimal if the size s of the output is a multiple of N. Unfortunately, s is in general $\leq N$ and the number of gates to evaluate remains $2^d - 1$. The evaluation of the function f is then only s times faster than the non-packed approach. As not all the slots are used,

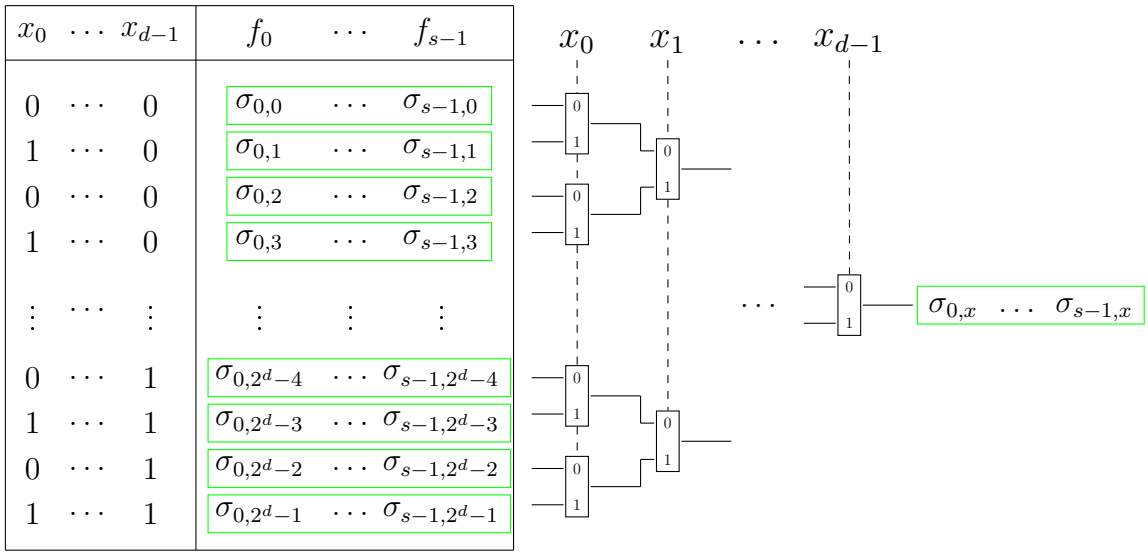

Figure 4.4: Horizontal packing (or batching) - *the LUT values corresponding to every input are packed together in a single* TRLWE *sample (green box). The d input bits $x_0, \ldots, x_{d-1}$ are encrypted separately in* TRGSW *ciphertexts. The evaluation of the* `CMux` *tree is done in a classical way but all the s output values $\sigma_{0,x}, \ldots, \sigma_{s-1,x}$ (with $x = \sum_{i=0}^{d-1} x_i 2^i$) are given in a single pass.*

this technique is not optimal if s is small. The elementary Lemma 4.2.1 specifies the noise propagation and it follows immediately from Lemma 3.2.2 and from the construction of the binary decision `CMux` tree, which has depth d.

Lemma 4.2.1 (Horizontal Packing - Batching)**.** *Let $\mathbf{d}_0, \ldots, \mathbf{d}_{2^d-1}$ be* TRLWE *samples*[2] *such that $\mathbf{d}_h \in \mathsf{TRLWE}_K(\sum_{j=0}^{s} \sigma_{j,h} X^j)$ for $h \in [\![0, 2^d - 1]\!]$. Here the $\sigma_{j,h}$ are the LUT values relative to an arbitrary function $f : \mathbb{B}^d \to \mathbb{T}^s$. Let $C_0, \ldots, C_{d-1}$ be* TRGSW *samples, such that $C_i \in \mathsf{TRGSW}_K(x_i)$ with $x_i \in \mathbb{B}$ (for $i \in [\![0, d-1]\!]$), and $\mathbf{x} = (x_0, \ldots, x_{d-1})$. Let $\mathbf{d}$ be the* TRLWE *sample output by the f evaluation of the binary decision* `CMux` *tree for the LUT (described in figure 4.4). Then, using the same notations as in lemma 3.2.2 and setting $\mathsf{msg}(\mathbf{d}) = f(\mathbf{x})$:*

- $\|\mathsf{Err}(\mathbf{d})\|_\infty \leq \mathcal{A}_{\mathsf{TRLWE}} + d \cdot ((k+1)\ell N \beta \mathcal{A}_{\mathsf{TRGSW}} + (kN+1)\epsilon)$ *(worst case),*
- $\mathsf{Var}(\mathsf{Err}(\mathbf{d})) \leq \vartheta_{\mathsf{TRLWE}} + d \cdot ((k+1)\ell N \beta^2 \vartheta_{\mathsf{TRGSW}} + (kN+1)\epsilon^2)$ *(average case),*

where $\mathcal{A}_{\mathsf{TRLWE}}$ and $\mathcal{A}_{\mathsf{TRGSW}}$ are upper bounds of the infinite norm of the errors of the TRLWE *samples ant the* TRGSW *samples respectively and $\vartheta_{\mathsf{TRLWE}}$ and $\vartheta_{\mathsf{TRGSW}}$ are upper bounds of their variances.*

[2]The TRLWE samples can be trivial samples, in the case where the function f and its LUT are public.

Vertical Packing

As explained in previous section, when s (number of output values of the function f to be evaluated) is too small compared to N, the horizontal packing leaves many unused slots on the TRLWE container. Evaluating the LUT with this improvement is more convenient than the trivial technique, but it does not exploit the full potential of the TRLWE container.
In order to improve the evaluation of the LUT, we propose another optimization called *Vertical Packing*. As for the horizontal packing we use the TRLWE encryption to encode N values at the same time. But now, instead of packing the LUT values $\sigma_{j,h}$ with respect to a fixed $h \in [\![0, 2^d - 1]\!]$, i.e. "horizontally", we pack N values $\sigma_{j,h}$ "vertically", with respect to a fixed $j \in [\![0, s-1]\!]$, as shown in Figure 4.5.

x_0	$\cdots$	x_{d-1}	f_0	$\cdots$	f_{s-1}
0	$\cdots$	0	$\sigma_{0,0}$	$\cdots$	$\sigma_{s-1,0}$
1	$\cdots$	0	$\sigma_{0,1}$	$\cdots$	$\sigma_{s-1,1}$
0	$\cdots$	0	$\sigma_{0,2}$	$\cdots$	$\sigma_{s-1,2}$
1	$\cdots$	0	$\sigma_{0,3}$	$\cdots$	$\sigma_{s-1,3}$
$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
0	$\cdots$	1	$\sigma_{0,2^d-4}$	$\cdots$	$\sigma_{s-1,2^d-4}$
1	$\cdots$	1	$\sigma_{0,2^d-3}$	$\cdots$	$\sigma_{s-1,2^d-3}$
0	$\cdots$	1	$\sigma_{0,2^d-2}$	$\cdots$	$\sigma_{s-1,2^d-2}$
1	$\cdots$	1	$\sigma_{0,2^d-1}$	$\cdots$	$\sigma_{s-1,2^d-1}$

Figure 4.5: *Vertical packing (vertical red box) and horizontal packing (or batching, horizontal green box).*

This packing technique can be used even if the function f has a single output value. Furthermore, to fill the TRLWE container is easier: generally $N = 2^{10}$, so it is sufficient to have a 10-bit input to fill all the slots.
This time, instead of just evaluating a full `CMux` tree, we use a different approach. If the LUT values are packed in "boxes", our technique first uses a `CMux` tree to select the right box, and then, a blind rotation (Algorithm 5) to find the right element inside the selected box. Finally, this element is extracted via SampleExtract (Algorithm 4) as a TLWE ciphertext. Figure 4.6 summarizes the schematized idea of the entire procedure.
To be more formal, suppose that one wants to evaluate the function f, or just one of its sub-functions f_j, on a fixed input $\mathbf{x} = (x_0, \ldots, x_{d-1})$. We assume we know the

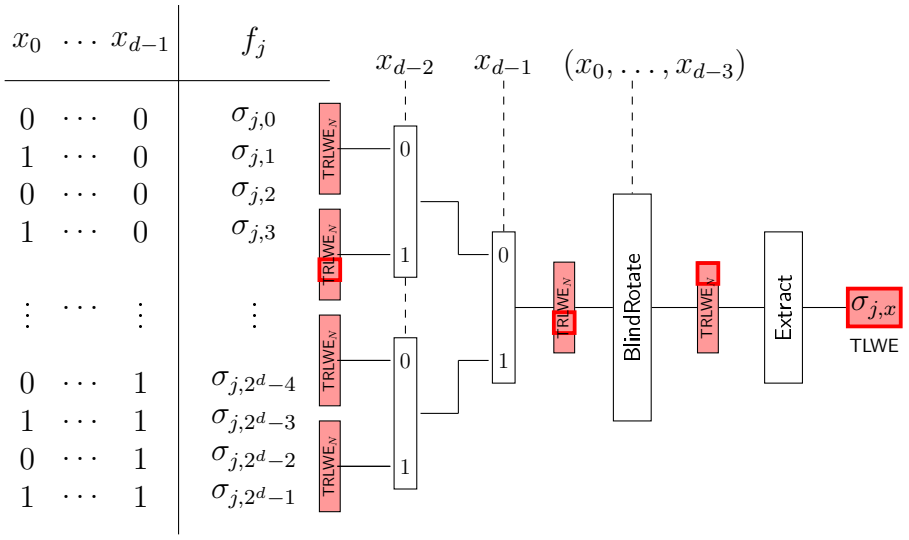

Figure 4.6: Vertical packing for the evaluation of the f_j LUT - *As described in Algorithm 6, the image represents the idea of evaluation of the sub-function f_j on $\mathbf{x} = (x_0, \ldots, x_{d-1})$ via vertical packing technique. After "vertically" packing the LUT values $\sigma_{j,h}$ (for $h \in [\![0, 2^d - 1]\!]$) in groups of size N, inside* TRLWE *samples, a* CMux *tree, a blind rotation and a sample extraction are evaluated. The* CMux *tree is initially used to select the* TRLWE *sample containing the output value (red box). Then the output value is moved in the first slot of the* TRLWE *container by using the blind rotation (Algorithm 5) and extracted by using the sample extraction (Algorithm 4). The bits of $\mathbf{x}$ are given as* TRGSW *samples and the final result $f_j(\mathbf{x}) = \sigma_{j,x}$ (with $x = \sum_{i=0}^{d-1} x_i 2^i$) is extracted as a* TLWE *sample. In our example, we used $2^d = 4N$.*

LUT associated to f_j as in figure 4.6. The output of $f_j(\mathbf{x})$ is just the LUT value $\sigma_{j,x}$, at position $x = \sum_{i=0}^{d-1} x_i 2^i$.

Let $\delta = \log_2(N)$. We analyze the general case where 2^d is a multiple of $N = 2^\delta$. The LUT of f_j, which is a column of 2^d values, is now packed as $2^d/N$ TRLWE ciphertexts $\mathbf{d_0}, \ldots, \mathbf{d}_{2^{d-\delta}-1}$, where each $\mathbf{d}_k$ encodes N consecutive LUT values $\sigma_{j,kN}, \ldots, \sigma_{j,(k+1)N-1}$.

To retrieve $f_j(\mathbf{x})$, we first need to select the block that contains $\sigma_{j,x}$. This block has index $p = \lfloor x/N \rfloor$, whose bits are the $d - \delta$ most significant bits of x. Since the TRGSW encryptions of these bits are among our inputs, one can use a CMux tree to select this block $\mathbf{d_p}$. Then, $\sigma_{j,x}$ is the ρ-th coefficient of the message of $\mathbf{d_p}$ where $\rho = x \mod N = \sum_{i=0}^{\delta-1} x_i 2^i$. The bits of ρ are the δ least significant bits of x, which are also available as TRGSW ciphertexts in our inputs. We can therefore use a blind rotation (Algorithm 5) to homomorphically multiply $\mathbf{d_p}$ by $X^{-\rho}$, which brings the coefficient $\sigma_{j,x}$ in the first slot of the TRLWE container, and finally, we extract it with a sample extraction (Algorithm 4). Algorithm 6 details the evaluation of $f_j(\mathbf{x})$.

Algorithm 6 Vertical Packing LUT of $f_j : \mathbb{B}^d \to \mathbb{T}$ (calling Algorithm 5 and 4)

Input: A list of $\frac{2^d}{N}$ TRLWE samples $\mathbf{d}_p \in \mathsf{TRLWE}_K(\sum_{i=0}^{N-1} \sigma_{j,pN+i} X^i)$ for $p \in [\![0, \frac{2^d}{N} - 1]\!]$, a list of d TRGSW samples $C_i \in \mathsf{TRGSW}_K(x_i)$, with $x_i \in \mathbb{B}$ and $i \in [\![0, d-1]\!]$.
Output: A TLWE sample $\mathfrak{c} \in \mathsf{TLWE}_{\mathfrak{K}}(\sigma_j = f_j(\mathbf{x}))$, with $\mathbf{x} = (x_0, \ldots, x_{d-1})$.

1: Evaluate the binary decision CMux tree of depth $d - \delta$, with TRLWE inputs $\mathbf{d}_0, \ldots, \mathbf{d}_{\frac{2^d}{N}-1}$ and TRGSW inputs $C_\delta, \ldots, C_{d-1}$, and output a TRLWE sample $\mathbf{d}$
2: $\mathbf{d} \leftarrow \mathsf{BlindRotate}(\mathbf{d}, (2^0, \ldots, 2^{\delta-1}, 0), (C_0, \ldots, C_{\delta-1}))$
3: Return $\mathfrak{c} = \mathsf{SampleExtract}(\mathbf{d})$

The entire cost of the evaluation of $f_j(\mathbf{x})$ with Algorithm 6 consists in $\frac{2^d}{N} - 1$ CMux gates and a single blind rotation, which corresponds to δ CMux gates. Overall, we get a speed-up by a factor N on the evaluation of each partial function, so a factor N in total.

Lemma 4.2.2 (Vertical Packing)**.** *Let $f_j : \mathbb{B}^d \to \mathbb{T}$ be a sub-function of the arbitrary function $f : \mathbb{B}^d \to \mathbb{T}^s$, with LUT values $\sigma_{j,0}, \ldots, \sigma_{j,2^d-1}$. Let $\mathbf{d}_0, \ldots, \mathbf{d}_{\frac{2^d}{N}-1}$ be* TRLWE *samples, such that $\mathbf{d}_p \in \mathsf{TRLWE}_K(\sum_{i=0}^{N-1} \sigma_{j,pN+i} X^i)$ for $p \in [\![0, \frac{2^d}{N} - 1]\!]$[3]. Let $C_0, \ldots, C_{d-1}$ be* TRGSW *samples, such that $C_i \in \mathsf{TRGSW}_K(x_i)$, with $x_i \in \mathbb{B}$ for $i \in [\![0, d-1]\!]$ and $\mathbf{x} = (x_0, \ldots, x_{d-1})$.*
Then algorithm 6 outputs a TLWE *sample $\mathfrak{c}$ such that $\mathsf{msg}(\mathfrak{c}) = f_j(\mathbf{x}) = \sigma_{j,x}$, with $x = \sum_{i=0}^{d-1} x_i 2^i$, and using the same notations as in Lemma 3.2.2 and Theorem 4.1.3, we have:*

- $\|\mathsf{Err}(\mathbf{d})\|_\infty \leq \mathcal{A}_{\mathsf{TRLWE}} + d \cdot ((k+1)\ell N \beta \mathcal{A}_{\mathsf{TRGSW}} + (1 + kN)\epsilon)$ *(worst case),*
- $\mathsf{Var}(\mathsf{Err}(\mathbf{d})) \leq \vartheta_{\mathsf{TRLWE}} + d \cdot ((k+1)\ell N \beta^2 \vartheta_{\mathsf{TRGSW}} + (1 + kN)\epsilon^2)$ *(average case),*

where $\mathcal{A}_{\mathsf{TRLWE}}$ and $\mathcal{A}_{\mathsf{TRGSW}}$ are upper bounds of the infinite norm of the errors in the TRLWE *samples ant the* TRGSW *samples respectively, while $\vartheta_{\mathsf{TRLWE}}$ and $\vartheta_{\mathsf{TRGSW}}$ are upper bounds of the variances.*

Proof. The proof follows immediately from the results of Lemma 3.2.2 and Theorem 4.1.3, and from the construction of the binary decision CMux tree. In particular, the first CMux tree has depth $(d - \delta)$ and the blind rotation evaluates δ CMux gates, which brings a total factor d in the depth. As the CMux depth is the same as in horizontal packing, the noise propagation matches too. □

Remark 4.2.2. *As previously mentioned, the horizontal and vertical packing techniques can be mixed together to improve the evaluation of f (Figure 4.7). This combination is optimal in the case where s and d are both small or if $2^d \cdot s > N$. In*

[3] If the sub-function f_j and its LUT are public, the LUT values $\sigma_{j,0}, \ldots, \sigma_{j,2^d-1}$ can be given in clear. This means that the TRLWE samples $\mathbf{d}_p$, for $p \in [\![0, \frac{2^d}{N} - 1]\!]$ are given as trivial TRLWE samples $\mathbf{d}_p \leftarrow (\mathbf{0}, \sum_{i=0}^{N-1} \sigma_{j,pN+i} X^i)$ in input to algorithm 6.

particular, if we pack $x = s$ coefficients horizontally and $y = N/x$ coefficients vertically, we need $\lceil 2^d/y \rceil - 1$ CMux gates plus one vertical packing LUT evaluation in order to evaluate f, which is equivalent to $\log_2(y)$ CMux evaluations. The result is composed of the first x TLWE samples extracted.

x_0	$\cdots$	x_{d-1}	f_0	$\cdots$	f_{s-1}
0	$\cdots$	0	$\sigma_{0,0}$	$\cdots$	$\sigma_{s-1,0}$
1	$\cdots$	0	$\sigma_{0,1}$	$\cdots$	$\sigma_{s-1,1}$
0	$\cdots$	0	$\sigma_{0,2}$	$\cdots$	$\sigma_{s-1,2}$
1	$\cdots$	0	$\sigma_{0,3}$	$\cdots$	$\sigma_{s-1,3}$
$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
0	$\cdots$	1	$\sigma_{0,2^d-4}$	$\cdots$	$\sigma_{s-1,2^d-4}$
1	$\cdots$	1	$\sigma_{0,2^d-3}$	$\cdots$	$\sigma_{s-1,2^d-3}$
0	$\cdots$	1	$\sigma_{0,2^d-2}$	$\cdots$	$\sigma_{s-1,2^d-2}$
1	$\cdots$	1	$\sigma_{0,2^d-1}$	$\cdots$	$\sigma_{s-1,2^d-1}$

Figure 4.7: *Intuitively, the horizontal green rectangle encircles the bits packed in the horizontal packing, while the vertical red rectangle encircles the bits packed in the vertical packing. The dashed blue square represents the packing in the case where the two techniques are mixed, as long as the number of coefficients in the area is $\leq N$.*

4.2.2 Deterministic automata

It is folklore that every deterministic program which reads its input bit-by-bit in a pre-determined order, uses fewer than B bits of memory, and produces a Boolean answer, is equivalent to a deterministic automata of at most 2^B states (independently of the time complexity). This is in particular the case for every Boolean function of p variables, that can be trivially executed with $p - 1$ bits of internal memory by reading and storing its input bit-by-bit before returning the final answer. It is of particular interest for most arithmetic functions, like addition, multiplication, or CRT operations, whose naive evaluation only requires $O(\log(p))$ bits of internal memory.

But when output space is not binary, and several bits are packed together as we show in previous sections, a more powerful tool is needed to manage the evaluations in an efficient way.

In this section we present deterministic Weighted Finite Automata (det-WFA), a generalization of deterministic Finite Automata (det-FA) obtained by adding a weight in each transition.
Very informally, a deterministic finite automaton is represented by a set of finite states linked by transitions. One of the states is an initial (or start) state, i.e. the entering state to the automaton, and some states are final (or accept) states. To go from a state to another, starting from the initial state, we read a word, which is the input to the automaton: every letter (generally bits) of the word activates one and only one transition (deterministic) from the current state to a following one. Once the entire word has been read, if the current state is an accept state the automaton accepts the word, otherwise it is rejected. We say that det-FA are decisional: they output a single bit of information, 1 if the word is accepted, 0 if the word is rejected. A toy example of deterministic finite automaton is given in Figure 4.8.

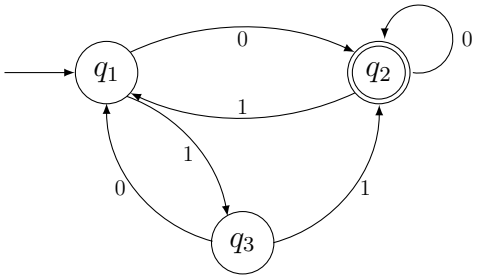

Figure 4.8: Toy example of det-FA - *The automaton has 3 states $Q = \{q_1, q_2, q_3\}$. The initial state i is equal to q_1, and there is a single finite state q_2, denoted by a double circle. The expected words are list of bits (so the alphabet Σ is $\mathbb{B}$). Two transitions (corresponding to letters 0 and 1) come out from every state.*

A deterministic weighted finite automaton, instead, looks exactly like a det-FA, but with weights added to every transition. An initial weight is set to 0 and it is incremented at every transition by the corresponding weight. Thus, the final result corresponds to the sum of the weights of the transitions that the automaton evaluated by reading the input word. A toy example is given in Figure 4.9.

If we want to evaluate a function via det-FA, we need to evaluate one automaton per output bit of the function. The det-FA approach is the first one we proposed in [CGGI16a]. The year after, in [CGGI17a], we proposed an evaluation via det-WFA. This generalized technique from the automata theory allows to evaluate a single automaton to obtain all the bits of the final result in a single pass. Det-WFA are in fact computational, instead of decisional: roughly speaking, the weights added in transitions act like a sort of memory that accumulates the bits of the final result all along the evaluation. Moreover, the transitions in det-FA and det-WFA have the same cost, which implies a considerable advantage in using the second ones. We

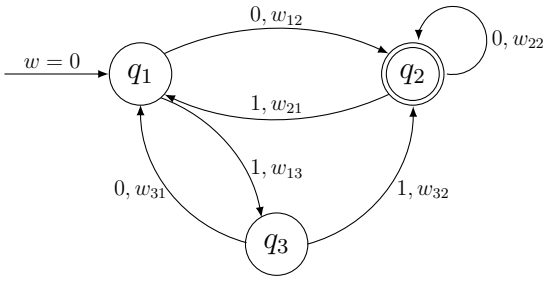

Figure 4.9: Toy example of det-WFA - *The automaton is the same as in Figure 4.8, but this time we added a weights in every transition. The weight w is initialized to 0: in every transition going from a state q_i to a state q_j, the weight w is incremented by w_{ij}. The final result is the weight at the final state.*

clarify all these ideas by giving some practical examples in the final part of this section.
We start by detailing the use of det-WFA to evaluate some arithmetic functions (largely used in applications, such as addition (and multi-addition), multiplication, squaring, comparison and max, etc.). We refer to [BGW00] and [DG09] for further details.
Definition 4.2.1 is a simplified version of the original definition, that we adapt in order to keep all this part as simple as possible.

Definition 4.2.1 (Deterministic weighted finite automata (det-WFA))**.** *A deterministic weighted finite automaton (det-WFA) over a group $(S, \oplus)$ is a tuple $\mathfrak{A} = (Q, i, \Sigma, \mathcal{T})$, where Q is a finite set of states, i is the initial state, Σ is the alphabet, $\mathcal{T} \subseteq Q \times \Sigma \times S \times Q$ is the set of transitions. Every transition itself is a tuple $t = q \xrightarrow{\sigma,\nu} q'$ from the state q to the state q' by reading the letter σ with weight $w(t)$ equal to ν, and there is at most one transition per every pair (q, σ).*

Let $P = (t_1, \ldots, t_d)$ be a path, with $t_j = q_{j-1} \xrightarrow{\sigma_j,\nu_j} q_j$. The word $\sigma = \sigma_1 \ldots \sigma_d \in \Sigma^d$ induced by P has weight $w(\sigma)$ equal to $\bigoplus_{j=1}^{d} w(t_j)$, where the $w(t_j)$ are all the weights of the transitions in P: σ is called the label of P. Because the automaton is deterministic, every label induces a single path (i.e. there is only one possible path per word).

Remark 4.2.3. *In our applications, we fix the alphabet $\Sigma = \mathbb{B}$. Definition 4.2.1 restraints the WFA to the deterministic complete accessible (the non-deterministic case is not supported), and universally accepting case (i.e all the words are accepted). In the general (non-deterministic) case, the additive group would be replaced by the second law of a semi-ring $(S, \bullet, \oplus)$, and we would sum the weights, using the first law, of all accepting paths. However, since non-determinism is not supported, we want to keep the definition as simple as possible. In the rest of the manuscript we set $(S, \oplus)$ as $(\mathbb{T}_N[X], +)$.*

Our goal is to use det-WFA to evaluate arithmetic functions. The evaluation of an automaton is done via a list of MUX gates, starting from the final states back to the initial one. Of course, we want to do this evaluation homomorphically. Next theorem gives the details of such evaluation and proposes the analysis of the noise propagation during the computations. A list of CMux gates is evaluated: the TRGSW ciphertexts contain the bits of the word to be evaluated and the TRLWE samples contain the partial weights. The final weight contains all the bits of the result.

Theorem 4.2.1 (Homomorphic evaluation of det-WFA)**.** *Let $\mathfrak{A} = (Q, i, \mathbb{B}, \mathcal{T})$ be a det-WFA with weights in $(\mathbb{T}_N[X], +)$, and let $|Q|$ denote the total number of states. Let $C_0, \dots, C_{d-1}$ be d valid TRGSW_K samples of the bits of a word $\sigma = \sigma_0 \dots \sigma_{d-1}$. By evaluating at most $d \cdot |Q|$ CMux gates, Algorithm 7 outputs a TRLWE_K sample $\mathbf{d}$ that encrypts the weight $w(\sigma)$, such that (using the same notations as in Lemma 3.2.2)*

- $\|\mathsf{Err}(\mathbf{d})\|_\infty \leq d \cdot ((k+1)\ell N \beta \mathcal{A}_{\mathsf{TRGSW}} + (kN+1)\epsilon)$ *(worst case),*
- $\mathsf{Var}(\mathsf{Err}(\mathbf{d})) \leq d \cdot ((k+1)\ell N \beta^2 \vartheta_{\mathsf{TRGSW}} + (kN+1)\epsilon^2)$ *(average case),*

where $\mathcal{A}_{\mathsf{TRGSW}}$ is an upper bound on the infinite norm of the error in the TRGSW *samples and $\vartheta_{\mathsf{TRGSW}}$ is an upper bound of their variance. Moreover, if all the words connecting the initial state to a fixed state $q \in Q$ have the same length, then the upper bound on the number of* CMux *evaluated by Algorithm 7 decreases to $|Q|$.*

Algorithm 7 Homomorphic evaluation of a det-WFA

Input: A det-WFA $\mathfrak{A} = (Q, i, \mathbb{B}, \mathcal{T})$. $\mathcal{T}_0(q)$ and $\mathcal{T}_1(q)$ denote the states that are reached when reading a binary letter from the state q, and $w_0(q)$ and $w_1(q)$ the weight of these transitions. d valid TRGSW_K samples $C_0, \dots, C_{d-1}$ of the bits of a word $\sigma = \sigma_0 \dots \sigma_{d-1}$.

Output: A TRLWE_K encryption of $w(\sigma)$

1: **for each** $q \in Q$ accessible at depth d **do**
2: set $c_{d,q} := 0$
3: **end for**
4: **for** $j = d-1$ down to 0 **do**
5: **for each** $q \in Q$ accessible at depth j **do**
6: set $c_{j,q} := \mathtt{CMux}\left(C_j, \mathbf{c}_{j+1,\mathcal{T}_1(q)} + (\mathbf{0}, w_1(q)), \mathbf{c}_{j+1,\mathcal{T}_0(q)} + (\mathbf{0}, w_0(q))\right)$.
7: **end for**
8: **end for**
9: **return** $c_{0,i}$

Proof. Let $q \in Q$ be a state, and σ a binary word, there exists a unique path starting from q and labelled by σ. We note $w(q, \sigma)$ the weight of this path. Algorithm 7

evaluates the weights backwards from the last letter σ_{d-1} of the word to the first one. The invariant of the the main loop is that for all j in $[0, d]$ and $q \in Q$, if q is accessible at depth j, $c_{j,q}$ is a TRLWE_K sample of $w(q, (\sigma_j, \dots, \sigma_{d-1}))$. Its error amplitude satisfies

$$\|\mathsf{Err}(c_{j,q})\|_\infty \leq (d-j)(k+1)\ell N\beta\mathcal{A}_{\mathsf{TRGSW}} + (d-j)(kN+1)\epsilon,$$

and its error variance satisfies

$$\mathsf{Var}(\mathsf{Err}(\mathbf{c}_{j,q})) \leq (d-j)(k+1)\ell N\beta^2\vartheta_{\mathsf{TRGSW}} + (d-j)(kN+1)\epsilon^2.$$

For $j = d$, the invariant is true, because step 2 initializes the TRLWE_K samples to zero. It is the weight of the empty word, and the error is null. Assuming by induction that the invariant holds at depth $j+1$, we analyze what happens at line 6 on iteration j on state q.

Consider the two transitions $q \xrightarrow{0,w_0(q)} \mathcal{T}_0(q)$ and $q \xrightarrow{1,w_1(q)} \mathcal{T}_1(q)$, where $\mathcal{T}_0(q)$ and $\mathcal{T}_1(q)$ denote the states that are reached when reading a binary letter from the state q, and $w_0(q)$ and $w_1(q)$ denote the weights of these transitions. If q is accessible at depth j, then $\mathcal{T}_0(q)$ and $\mathcal{T}_1(q)$ are both accessible at depth $j+1$, and encode respectively $w(\mathcal{T}_0(q), (\sigma_{j+1} \dots \sigma_{d-1}))$ and $w(\mathcal{T}_1(q), (\sigma_{j+1} \dots \sigma_{d-1}))$. Therefore, after applying the `CMux`, the message of $c_{j,q}$ is $w(q, (\sigma_j \dots \sigma_{d-1}))$. By applying the noise propagation inequalities of Lemma 3.2.2, we have that

$$\begin{aligned}
\|\mathsf{Err}(\mathbf{c}_{j,q})\|_\infty &\leq \|\mathsf{Err}(\texttt{CMux}(C_j, \mathbf{c}_{j+1,\mathcal{T}_1(q)} + (\mathbf{0}, w_1(q)), \mathbf{c}_{j+1,\mathcal{T}_0(q)} + (\mathbf{0}, w_0(q))))\|_\infty \\
&\leq \max(\|\mathsf{Err}(\mathbf{c}_{j+1,\mathcal{T}_1(q)})\|_\infty, \|\mathsf{Err}(\mathbf{c}_{j+1,\mathcal{T}_0(q)})\|_\infty) \\
&\quad + (k+1)\ell N\beta\|\mathsf{Err}(C_j)\|_\infty + (kN+1)\epsilon
\end{aligned}$$

in the worst case, and

$$\begin{aligned}
\mathsf{Var}(\mathsf{Err}(\mathbf{c}_{j,q})) &\leq \mathsf{Var}(\mathsf{Err}(\texttt{CMux}(C_j, \mathbf{c}_{j+1,\mathcal{T}_1(q)} + (\mathbf{0}, w_1(q)), \mathbf{c}_{j+1,\mathcal{T}_0(q)} + (\mathbf{0}, w_0(q)))))) \\
&\leq \max(\mathsf{Var}(\mathsf{Err}(\mathbf{c}_{j+1,\mathcal{T}_1(q)})), \mathsf{Var}(\mathsf{Err}(\mathbf{c}_{j+1,\mathcal{T}_0(q)}))) \\
&\quad + (k+1)\ell N\beta^2\mathsf{Var}(\mathsf{Err}(C_j)) + (kN+1)\epsilon^2
\end{aligned}$$

in the average case, which prove that the invariant holds at depth j, and thus, for all $j \in [0, d]$. Since the initial state i is accessible at depth $j = 0$, this proves that the final result is $\mathbf{c}_{0,i}$ encodes $w(i, \sigma) = w(\sigma)$, with the bounds announced in the theorem.

For the complexity, in the worst case, the main for-each loops over all states $q \in Q$ and all depths $j \in [0, d-1]$, which represents $d|Q|$ `CMux` evaluations. If, by the last condition of the theorem, each state is accessible only at a single depth, then all for-each ranges are disjoint subsets of Q, so the total number of `CMux` evaluated is $\leq |Q|$. □

Remark 4.2.4. *This algorithm can also evaluate regular deterministic Finite Automata (det-FA): in this case the weight of all transitions at depth $d-1$ that reach a final state is* 0.5*, and all other weights are* 0*.*

In the following sections we explain in detail how to use det-WFA to efficiently evaluate the maximal value and the multiplication between two d-bits integers.

Max

In this section we propose as an example the evaluation of the Max function of two d-bit integers $x = \sum_{i=0}^{d-1} x_i 2^i$ and $y = \sum_{i=0}^{d-1} y_i 2^i$, with $x_i, y_i \in \mathbb{B}$ for $i \in [\![0, d-1]\!]$. We take advantage of this example to show the difference between the evaluation via det-FA and det-WFA. The automata take in input all the bits of x and y and output the maximal value between them. This final result is a d-bit integer, that we note $m = \sum_{i=0}^{d-1} m_i 2^i$. This means that if we want to compute it with det-FA, we need to evaluate d separate automata, while with det-WFA a single automaton is sufficient.

We start by describing the evaluation via det-FA. In order to compute the max function, we exploit the automaton for the comparison, shown in Figure 4.10. The automaton reads in input the bits of x and y alternatively starting from the most significant ones. It starts from a state E, where x and y are equal and moves to the following states depending on the value of the input bits. If $x > y$, the automaton moves to the state A and stays there until the end of the evaluation and, in the same way, if $x < y$, the automaton moves to the state B and stays there until the end of the evaluation.
At this point, to compute all the bits of the max value, the idea is to deviate the automaton of the comparison and extract an intermediary value. For instance, as we show in Figure 4.11, to compute the third most significant bit m_{d-3} of the final result, we deviate the automaton and read just the 3 most significant bits of x and y. Every new bit of the result needs a different automaton, and previous computations cannot be re-used. In fact, as we explained before, the `MUX` evaluation of the automaton starts from the final state and goes back to the beginning. This implies that every time we deviate the automaton, the suffixes are ignored and the computations linked to those suffixes are useless.

Instead, by using det-WFA we evaluate a single automaton and we obtain the final result in a single pass. The construction of the Max det-WFA starts from the comparison automaton, as for det-FA. But this time, instead of deviating the automaton, we evaluate it entirely till the end and we add specific weights in some of the transitions, as shown in Figure 4.12. Here, we highlight in color the transitions on which we add the weights:

- If the most significant bit m_{d-1} of the final result is equal to 1, it implies that the automaton went through one of the light blue transitions;
- If the second most significant bit m_{d-2} of the final result is equal to 1, it implies that the automaton went through one of the orange transitions;

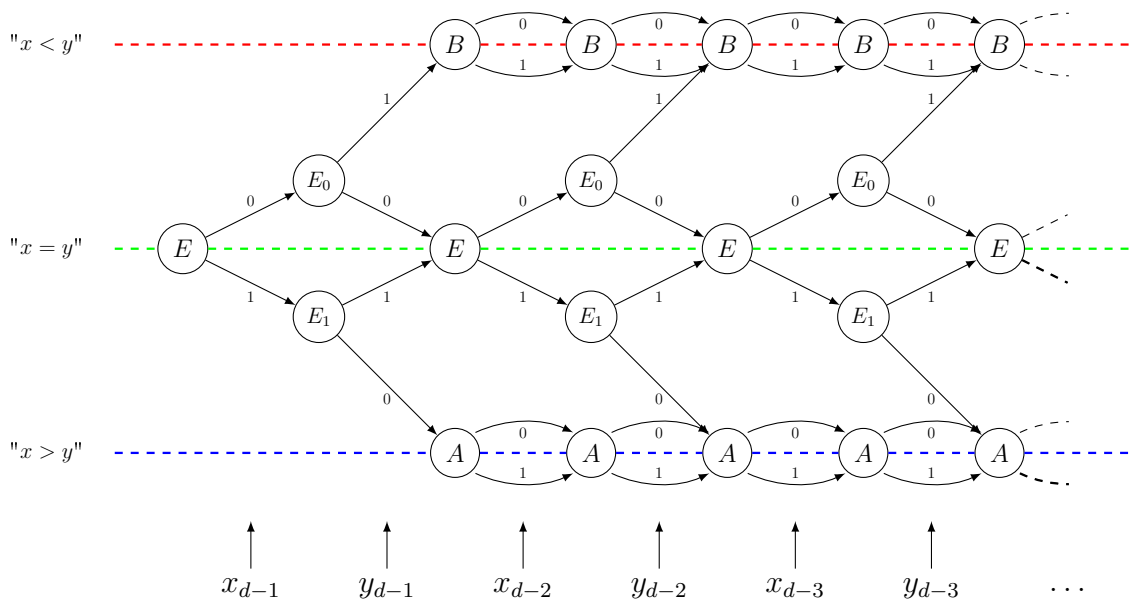

Figure 4.10: Comparison automaton - *The automaton reads in input the bits of x and y staring from the most significant ones. It moves between the main states E (indicating that $x = y$), A (indicating that $x > y$) and B (indicating that $x < y$), depending on the value of input bits.*

- If the third most significant bit m_{d-3} of the final result is equal to 1, it implies that the automaton went through one of the pink transitions;
- etc.

As the transitions in det-FA and det-WFA have the same cost (even if a weight is added), the cost of the evaluation of the max function via det-WFA is d times cheaper than the evaluation via det-FA.

To be more formal, we analyze in detail the evaluation of the max function via det-WFA. We can sum up the automaton presented in Figure 4.12 in a more compact way, as in Figure 4.13. As before, the automaton has 3 principal states (noted A, B, E) and 4 intermediary states (noted (A), (B), (E_1), (E_0)), that keep track of which number is the maximum, and in case of equality what is the last value of x_i. A weight $+\frac{1}{2}X^i$ is added on all the transitions that read the digit 1 from the maximum.

Remark 4.2.5. *In practice, to evaluate the MAX function, we convert the det-WFA in a circuit that counts* $5d$ `CMux` *gates. Roughly speaking, we have to read the automata (Figure 4.13) in the reverse. We initialize 3 states* A, B, E *as null* TRLWE

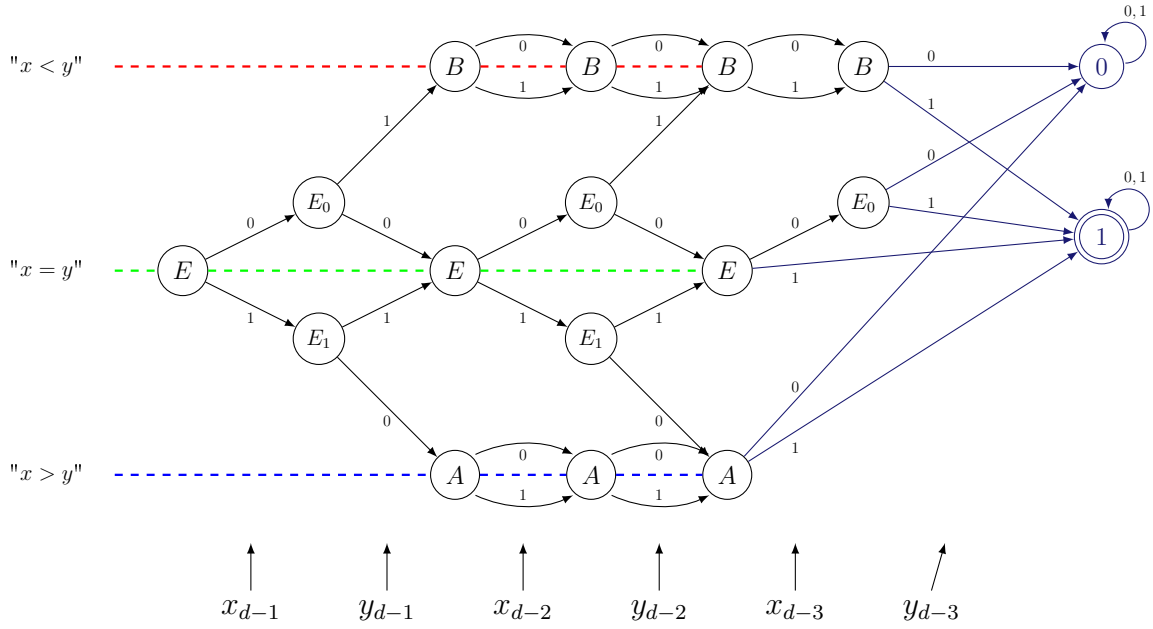

Figure 4.11: Idea - *Deviating the comparison automaton to compute the third most significant bit of the maximal value.*

samples. Then, for i from $d-1$ to 0, we update the states as follows:

$$\begin{cases} (A) := \texttt{CMux}(C_i^y, A + (\mathbf{0}, \frac{1}{2}X^i), A); \\ (E_0) := \texttt{CMux}(C_i^y, A + (\mathbf{0}, \frac{1}{2}X^i), E); \\ (E_1) := \texttt{CMux}(C_i^y, E, B); \\ (B) := \texttt{CMux}(C_i^y, B, B); \end{cases} \quad \begin{cases} A := \texttt{CMux}(C_i^x, (A), (A)); \\ E := \texttt{CMux}(C_i^x, (E_1) + (\mathbf{0}, \frac{1}{2}X^i), (E_0)); \\ B := \texttt{CMux}(C_i^x, (B) + (\mathbf{0}, \frac{1}{2}X^i), (B)). \end{cases}$$

Here the C_i^x and C_i^y are TRGSW *encryptions of the bits x_i and y_i respectively, and they are the inputs. We observe that two transitions are constant: the one computing (B) and the one updating A. So we can reduce the computation of the max value to the d-times repetition (for i from $d-1$ to 0) of the 5 following* CMux *homomorphic gates (the intermediary states (A) and (B) disappear):*

$$\begin{cases} A := \texttt{CMux}(C_i^y, A + (\mathbf{0}, \frac{1}{2}X^i), A); \\ E_0 := \texttt{CMux}(C_i^y, A + (\mathbf{0}, \frac{1}{2}X^i), E); \\ E_1 := \texttt{CMux}(C_i^y, E, B); \\ E := \texttt{CMux}(C_i^x, E_1 + (\mathbf{0}, \frac{1}{2}X^i), E_0); \\ B := \texttt{CMux}(C_i^x, B + (\mathbf{0}, \frac{1}{2}X^i), B). \end{cases}$$

The output of the evaluation is the TRLWE *sample E, which contains the maximal value.*

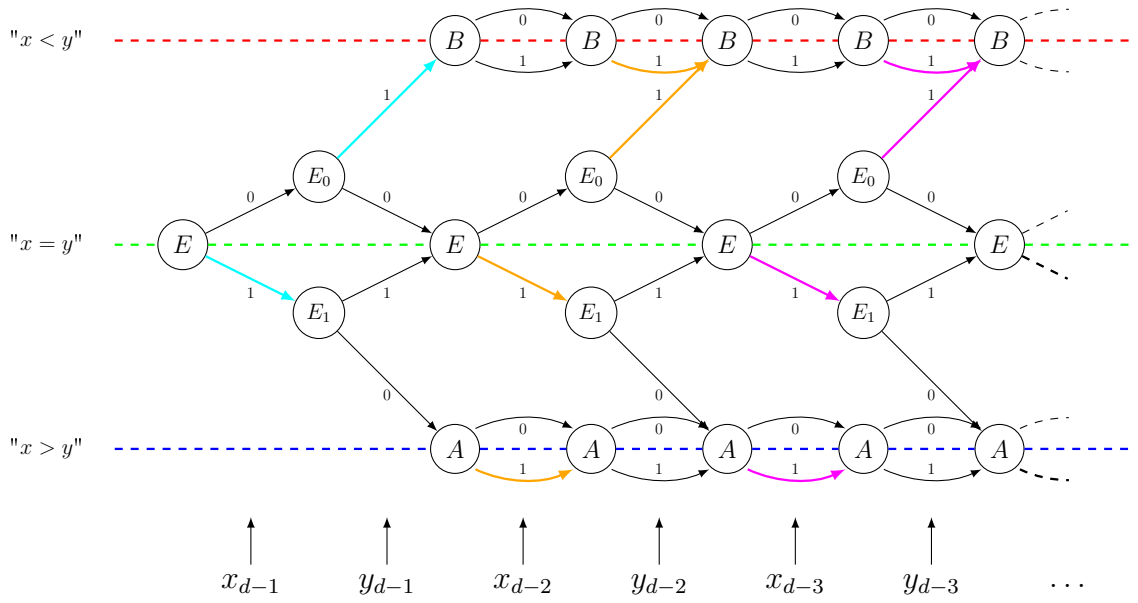

Figure 4.12: Det-WFA for the evaluation of the Max function - *In color we highlight the transitions on which a weight is added. Different colors indicate transitions that influence the value of the output bits in different positions.*

Overall, the next lemma, which is a direct consequence of Theorem 4.2.1, shows that the Max can be computed by evaluating only $5d$ `CMux` gates, instead of $\Theta(d^2)$ with classical deterministic automata.

Lemma 4.2.3 (Evaluation of Max det-WFA)**.** *Let $\mathfrak{A}$ be the det-WFA of the Max, described in Figure 4.13. Let $C_0^x, \ldots, C_{d-1}^x, C_0^y, \ldots, C_{d-1}^y$ be TRGSW_K samples of the bits of x and y respectively. By evaluating $5d$ `CMux` gates (depth $2d$), the Max det-WFA outputs a TRLWE sample $\mathbf{d}$ encrypting the maximal value between x and y and (with same notations as in Lemma 3.2.2)*

- $\|\mathsf{Err}(\mathbf{d})\|_\infty \leq 2d \cdot ((k+1)\ell N \beta \mathcal{A}_{\mathsf{TRGSW}} + (kN+1)\epsilon)$ *(worst case),*
- $\mathsf{Var}(\mathsf{Err}(\mathbf{d})) \leq 2d \cdot ((k+1)\ell N \beta^2 \vartheta_{\mathsf{TRGSW}} + (kN+1)\epsilon^2)$ *(average case).*

Here $\mathcal{A}_{\mathsf{TRGSW}}$ and $\vartheta_{\mathsf{TRGSW}}$ are upper bounds of the amplitude and of the variance of the errors in the TRGSW samples.

Multiplication

For the multiplication we use the same approach and we construct a det-WFA which maps the schoolbook multiplication. We illustrate the construction on the example of the multiplication between two 2-bits integers $x = x_1x_0$ and $y = y_1y_0$. As shown in the top part of Figure 4.14, after an initial step of bit by bit multiplication, a

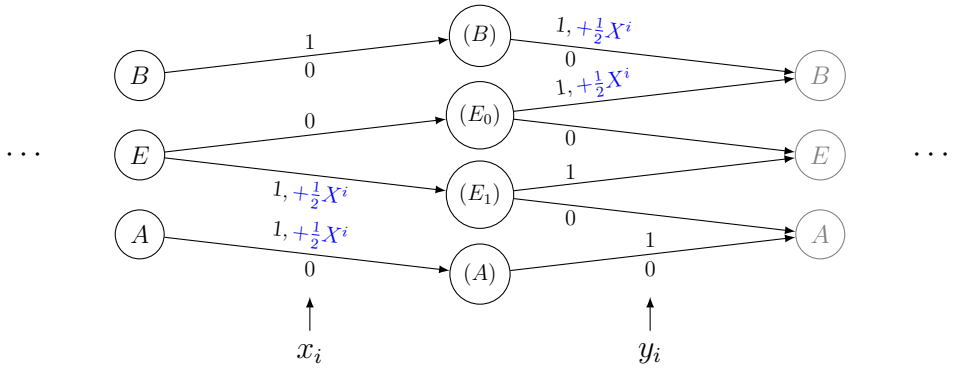

Figure 4.13: Max det-WFA - *The states A and (A) mean that x is the maximal value, the states B and (B) mean that y is the maximal value, and finally, the states E, (E_1) and (E_0) mean that x and y are equals on the most significant bits. If the current state is A or B, the following state remains the same. The initial state is E. If the current state is E, after reading x_i there are two possible intermediate states: (E_1) if $x_i = 1$ and (E_0) if $x_i = 0$. After reading the value of y_i, the 3 possible states A, B and E are possible. The det-WFA is repeated as many times as the bit length of the integers evaluated and the weights are given in clear.*

multi-addition (shifted of one place on the left for every line) is performed. The bits of the final result $m = m_3m_2m_1m_0$ are computed as the sum of each column with carry.
The det-WFA computes the multiplication by keeping track of the partial sum of each column in the states, and by using the transitions to update these sums. For the multiplication of 2-bits integers, the automaton (described in the bottom part of Figure 4.14) has 6 main states (i, c_0, c_{10}, c_{11}, c_{20}, c_{21}), plus 14 intermediary states (noted with capital letters and parenthesis) that store the last bit read. The value of the j-th (for $j \in [\![0, 3]\!]$) output bit is put in a weight on the last transition of each column. The final weight is the result of the multiplication.
For the generic multiplication of two d-bits integers, we can upper bound the number of states by $4d^3$, instead of $\Theta(d^4)$ with one classical automata per output bit. For a more precise number of states we wrote a C++ program to eliminate unreachable states and refine the leading coefficient. The depth is $2d^2$ and the noise evaluation can be easily deducted by previous results. The same principle can be used to construct the *multi-addition*, and its det-WFA is slightly simpler.

4.2.3 Bit Sequence Representation

We now present another design which is specific to the multi-addition (or its derivatives), but which is faster than the generic construction with det-WFA. The idea is to build an homomorphic scheme that can represent small integers, say between 0

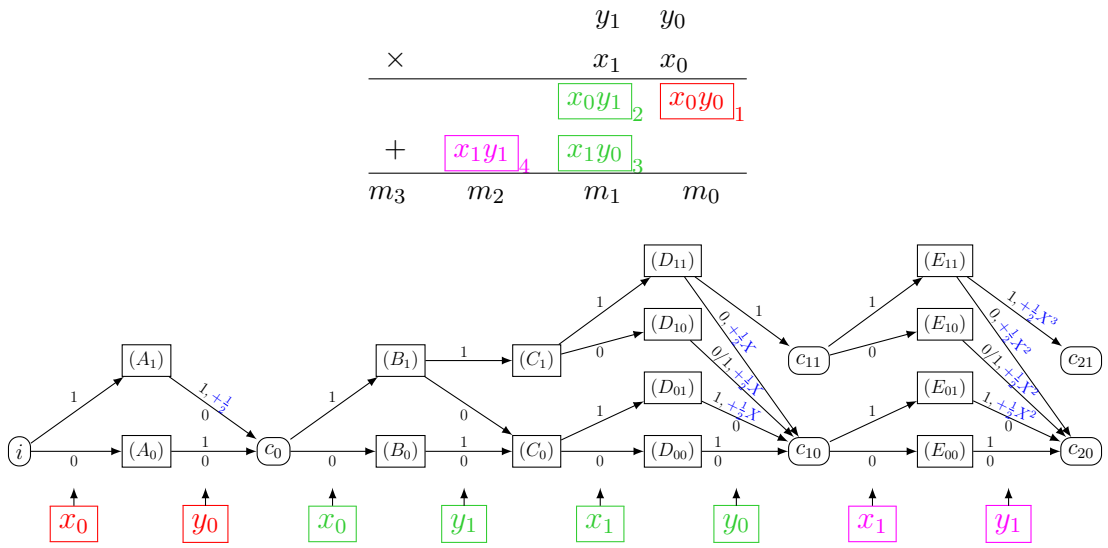

Figure 4.14: Schoolbook 2-bits multiplication and corresponding det-WFA

and $N = 2^p$, and which is dedicated to only the three elementary operations used in the multi-addition algorithm, namely:

1. Extract any of the bits of the value as a TLWE sample;
2. Increment the value by 1 (or by a small integer);
3. Integer division of the value by 2.

We now explain the basic idea, and then, we show how to implement it efficiently on TRLWE ciphertexts.

We represent integers modulo $2N$, with $N = 2^p$, by using their *Bit Sequence Representation* (BSR). For $j \in [\![0, p]\!]$ and $k, l \in \mathbb{Z}$, we call $B_{j,k}^{(l)}$ the j-th bit of $l + k$ in the little endian signed binary representation.
For each integer $k \in \mathbb{Z}$, $[B_{0,k}^{(l)}, B_{1,k}^{(l)}, ..., B_{p,k}^{(l)}]$ is the (little endian signed) binary representation of $l + k \mod 2N$. When k is not specified, $B_j^{(l)}$ represents the binary sequence of all the j-th bits of integers $l, l + 1, l + 2, \ldots$.
Let $l = 0$. Observe that $B_0^{(0)} = (0, 1, 0, 1, ...)$ is 2-periodic, $B_1^{(0)} = (0, 0, 1, 1, 0, 0, 1, 1...)$ is 4-periodic and, more generally, for all $j \in [\![0, p]\!]$ and $l \in \mathbb{Z}$, $B_j^{(l)}$ is 2^j-antiperiodic and it is the left shift of $B_j^{(0)}$ by l positions. Therefore, it suffices to have $2^j \leq N$ consecutive values of the sequence to (blindly) deduce all the remaining bits.
We now suppose that an integer $l \in [\![0, N - 1]\!]$ is represented by its BSR, defined as $BSR(l) = [B_0^{(l)}, \ldots, B_p^{(l)}]$.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	$\leftarrow k$
0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	$\leftarrow B_0^{(l)}$
1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	$\leftarrow B_1^{(l)}$
2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	$\leftarrow B_2^{(l)}$
3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	$\leftarrow B_3^{(l)}$
$\uparrow j$	l																

Figure 4.15: BSR - *The figure represents the $BSR(l)$. Here $l = 0$, $N = 2^3$. The first column corresponding to $k = 0$ contains the bits of l in the little endian representation.*

We now explain how to compute $BSR(l+1)$ (increment) and $BSR(\lfloor l/2 \rfloor)$ (divide by 2) using only copy and negations operations on bits at a fixed position which does not depend on l (blind computation). Then, we show how to represent these operations homomorphically on TRLWE ciphertexts.

Increment. Let $BSR(l) = [B_0^{(l)}, \dots, B_p^{(l)}]$ be the BSR of some unknown number $l \in [\![0, N-1]\!]$. Our goal is to compute the BSR of $l+1$, $BSR(l+1) = [B_0^{(l+1)}, \dots, B_p^{(l+1)}]$. If we know at lest N consecutive values of $B_j^{(l)}$ (for $j \in [\![0, p]\!]$), it suffices to define the sequence $B_j^{(l+1)}$ on N consecutive values, the rest is deduced by periodicity. To map the increment operation, all we need to shift the sequences by 1 position

$$B_{j,k}^{(l+1)} = B_{j,k+1}^{(l)} \text{ for all } k \in \mathbb{Z}.$$

More generally, we can increment the BSR by any integer in $[\![0, N-1]\!]$, as in Figure 4.16.

r_0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
r_1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
r_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
r_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

$$\downarrow \cdot X^{-5}$$

r'_0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
r'_1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
r'_2	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1
r'_3	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0

Figure 4.16: (T)BSR - *example of addition +5.*

Integer division by two. Let $BSR(l) = [B_0^{(l)}, \dots, B_p^{(l)}]$ be the BSR of some unknown number $l \in [\![0, N-1]\!]$. Our goal is to compute the BSR of $\lfloor \frac{l}{2} \rfloor$, $BSR(\lfloor \frac{l}{2} \rfloor) = [B_0^{(\lfloor \frac{l}{2} \rfloor)}, \dots, B_p^{(\lfloor \frac{l}{2} \rfloor)}]$.
As the BSR is the representation of integers in base 2, when we want to perform the division by 2, it corresponds to eliminate the first line and keep just the odd columns of the BSR (see Figure 4.15). Thus we can set

$$B_{j,k}^{(\lfloor \frac{l}{2} \rfloor)} = B_{j+1,2k}^{(l)} \text{ for } j \in [\![0, p-1]\!] \text{ and } \forall k \in \mathbb{Z}.$$

The only exception is the last line of the BSR, that has to be regenerated. Indeed, $B_{j+1,2k}^{(l)}$ is the $(j+1)$-th bit of $l+2k$ and it is the j-th bit of its half $\lfloor l/2 \rfloor + k$, which is our desired $B_{j,k}^{(\lfloor l/2 \rfloor)}$. This is unfortunately not enough to reconstruct the last sequence $B_p^{(\lfloor l/2 \rfloor)}$, since we have no information on the $(p+1)$-th bits in $BSR(l)$. However, in our case, we can reconstruct this last sequence directly. First, the numbers $\lfloor \frac{l}{2} \rfloor + k$ for $k \in [0, N/2-1]$ are all $< N$, so we can blindly set the corresponding $B_{p,k}^{(\lfloor \frac{l}{2} \rfloor)} = 0$. Then, we just need to note that $[B_{p,0}^{(l)}, \dots, B_{p,N-1}^{(l)}]$ is $N-l$ times 0 followed by l times 1, and our target $[B_{p,N/2}^{(\lfloor \frac{l}{2} \rfloor)}, \dots, B_{p,N-1}^{(\lfloor \frac{l}{2} \rfloor)}]$ must consist $N/2 - l$ times 0 followed by $\lfloor l/2 \rfloor$ times 1. Therefore, our target can be filled with the even positions $[B_{p,0}^{(l)}, B_{p,2}^{(l)}, \dots, B_{p,N-2}^{(l)}]$. To summarize, the last line of the BSR in the division by 2 corresponds to the following blind transformation:

$$\begin{cases} B_{p,k}^{(\lfloor \frac{l}{2} \rfloor)} & = & 0 \text{ for } k \in [\![0, \frac{N}{2} - 1]\!] \\ B_{p,N/2+k}^{(\lfloor \frac{l}{2} \rfloor)} & = & B_{p,2k}^{(l)} \text{ for } k \in [\![0, \frac{N}{2} - 1]\!] \end{cases}$$

The idea of the division by 2 is schematized in Figure 4.17.

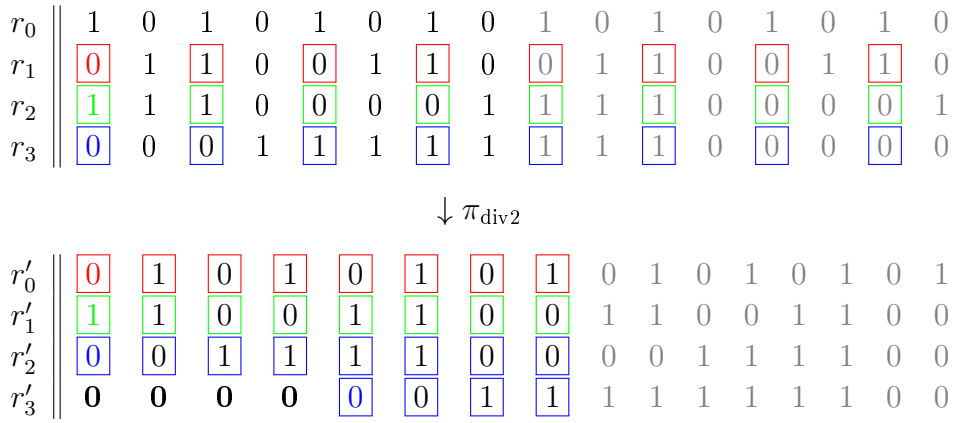

Figure 4.17: (T)BSR - *example of division by* 2.

TBSR

We now explain how we can encode these BSR sequences on TRLWE ciphertexts, considering that all the coefficients need to be in the torus rather than in $\mathbb{B}$, and that we need to encode sequences that are either N-periodic or N-antiperiodic. We note the encoded BSR by TBSR: this latter is simply an homomorphic counter.
Therefore, this is our basic encoding of the BSR sequences: let $BSR(l) = [B_0^{(l)}, \dots, B_p^{(l)}]$ be the BSR of some unknown number $l \in [\![0, N-1]\!]$. For $j \in [\![0, p-1]\!]$, we represent $B_j^{(l)}$ with the polynomial

$$\mu_j = \sum_{k=0}^{N-1} \frac{1}{2} B_{j,k}^{(l)} \cdot X^k$$

and we represent the last $B_p^{(l)}$ with the polynomial

$$\mu_p = \sum_{k=0}^{N-1} \left(\frac{1}{2} B_{p,k}^{(l)} - \frac{1}{4} \right) \cdot X^k.$$

This simple rescaling between the bit representation $BSR(l)$ and the torus representation $M = [\mu_0, \dots, \mu_p]$ is bijective.

The *increment* operation described in previous section for the BSR consists in a cyclic shift of coefficients. It correspond to the multiplication by X (or to a power of X) in the TBSR encoding, which has a similar behaviour on coefficients of torus polynomials.

The *integer division by two* immediately rewrites into an affine function thanks to the TBSR encoding. It transforms the coefficients $(\mu_{j,k})_{j \in [\![1,p]\!], k \in \{0,2,\dots,2N-2\}} \in \mathbb{T}^{pN}$ into $(\mu'_0, \dots, \mu'_p)$ as follow:

$$\pi_{\text{div2}} : \begin{cases} \mu'_{j,k} &= \mu_{j+1,2k} & \text{for } j \in [0, p-2], k \in [0, N-1] \\ \mu'_{p-1,k} &= \mu_{p,2k} + \frac{1}{4} & \text{for } k \in [0, N-1] \\ \mu'_{p,k} &= -\frac{1}{4} & \text{for } k \in [0, \frac{N}{2} - 1] \\ \mu'_{p,N/2+k} &= \mu_{p,2k} & \text{for } k \in [0, \frac{N}{2} - 1] \end{cases}$$

The output of the division by two is still an integer in $[\![0, N-1]\!]$, as the input. Finally, we call TBSR ciphertext of an unknown integer $l \in [\![0, N-1]\!]$ a vector $C = [c_0, ..., c_p]$ of TRLWE ciphertexts of message $[\mu_0, \dots, \mu_p]$.

Definition 4.2.2 (TBSR encryption)**.** *We define the TBSR encryption as follows.*

- *Parameters and keys:* TRLWE *parameter N with secret key $K \in \mathbb{B}_N[X]$, and a circular-secure (public) key-switching key $\mathsf{KS}_{K \to K, \gamma}$ from K to itself, noted just* KS*.*

- $\mathsf{TBSRSet}(l)$*: return a vector of trivial* TRLWE *ciphertexts encoding the torus representation of* $[B_0^{(l)}, \ldots, B_p^{(l)}]$.
- $\mathsf{TBSREncrypt}(l)$*: return a vector of non-trivial* TRLWE *ciphertexts encoding the torus representation of* $[B_0^{(l)}, \ldots, B_p^{(l)}]$.
- $\mathsf{TBSRBitExtract}_j(C)$*: Return* $\mathsf{SampleExtract}(c_j)$ *(Algorithm 4) when* $j < p$. [4]
- $\mathsf{TBSRIncrement}(C)$*: Return* $X^{-1} \cdot C$.
- $\mathsf{TBSRDiv2}(C)$*: Use* KS *to evaluate* π_{div2} *homomorphically on* C*. Since it is a 1-lipschitzian affine function, this consists in applying the public functional* $\mathsf{KeySwitch}$ *to* KS*, the linear part of* π_{div2} *and* C*, and then, translate the result by the constant part of* π_{div2}.

The following theorem is a direct consequence of Theorem 4.1.1 (with $n = N$). The correctness of the result has already been discussed.

Theorem 4.2.2 (TBSR operations)**.** *Let* N*,* K *and* KS *be TBSR parameters and keys as defined before, and let* C *be a TBSR ciphertext of* $l \in [\![0, N-1]\!]$ *with noise amplitude* η *(and noise variance* ϑ*). Then for* $j \in [\![0, p-1]\!]$*,* $\mathsf{TBSRBitExtract}_j(C)$ *is a* $\mathsf{TLWE}_{\mathfrak{K}}$ *ciphertext of the* j*-th bit of* l*, over the message space* $\{0, \frac{1}{2}\}$*, with noise amplitude (resp. variance)* $\leq \eta$ *(resp.* $\leq \vartheta$*). If* $l \leq N-2$ *(if* $l = N-1$ *the result is not determined),* $\mathsf{TBSRIncrement}(C)$ *is a TBSR ciphertext of* $l+1$ *with noise amplitude (resp. variance)* $\leq \eta$ *(resp.* $\leq \vartheta$*).* $C' = \mathsf{TBSRDiv2}(C)$ *is a TBSR ciphertext of* $\lfloor l/2 \rfloor$ *such that:*

- $\|\mathsf{Err}(C')\|_\infty \leq \|\mathsf{Err}(C)\|_\infty + N^2 t \mathcal{A}_{\mathsf{KS}} + N 2^{-(t+1)}$ *(worst-case),*
- $\mathsf{Var}(\mathsf{Err}(C')) \leq \mathsf{Var}(\mathsf{Err}(C)) + N^2 t \vartheta_{\mathsf{KS}} + N 2^{-2(t+1)}$ *(average case),*

where $\mathcal{A}_{\mathsf{KS}}$ *and* ϑ_{KS} *are the amplitude and variance of the key-switching key* KS*, respectively.*

Using the TBSR counter for a multi-addition or a multiplication.

The TBSR counter allows to perform a multi-addition or multiplication using the school-book elementary algorithms (see Algorithm 8 and Algorithm 9). This leads to a leveled multiplication circuit (with $\mathsf{KeySwitching}$) which is quadratic instead of cubic with weighted automata.

The following lemma analyzes the case of the multiplication and the result described is a consequence of Theorem 4.1.1. The correctness of the result can be deducted from the construction, presented in Algorithm 9 (with $N > 2d$).
The formulas for the multi-addition can be easily found, and the correctness comes from the construction (Algorithm 8, with $N > 2m$).

[4] For the p-th bit, one would return $\mathsf{SampleExtract}(c_p) + (\mathbf{0}, \frac{1}{4})$, but it is always 0 if $l \in [0, N-1]$.

Lemma 4.2.4. *Let N,B_g,ℓ and* KS *be TBSR and* TRGSW *parameters with the same key K. We suppose that each TBSR ciphertext has $p \leq 1 + \log(N)$* TRLWE *ciphertexts. Let (A_i) and (B_i) for $i \in [0, d-1]$ be* TRGSW*-encryptions of the bits of two d-bits integers (little endian), with the same noise amplitude $\mathcal{A}$ (resp. variance ϑ).*
Then, Algorithm 9 computes all the bits of the product within $2d^2p$ `CMux` *and $(2d-2)p$ public key switching, and the output ciphertexts satisfy:*

- $\|\mathsf{Err}(Out)\|_\infty \leq 2d^2((k+1)\ell N\beta\mathcal{A} + (kN+1)\epsilon) + (2d-2)(N^2t\mathcal{A}_{\mathsf{KS}} + N2^{-(t+1)})$,
- $\mathsf{Var}(\mathsf{Err}(Out)) \leq 2d^2((k+1)\ell N\beta^2\vartheta + (kN+1)\epsilon^2) + (2d-2)(N^2t\vartheta_{\mathsf{KS}} + N2^{-2(t+1)})$,

where $\mathcal{A}_{\mathsf{KS}}$ and ϑ_{KS} are the amplitude and variance of the key-switching key KS*, respectively.*

Algorithm 8 TBSR multi-addition

Input: The bitwise TRGSW-encryption $(A_{i,j})_{i\in[0,m-1],j\in[0,d-1]}$ of m integers $a_0, \ldots, a_{m-1}$ of d-bits each
Output: bitwise LWE encryptions of the sum $\mathrm{Out}_0, \ldots, \mathrm{Out}_{d+log_2(m)}$.

$C \leftarrow \mathsf{TBSRSet}(0)$ ▷ The counter is initialized at 0
for $j = 0$ to $d-2$ **do**
  for $i = 0$ to $m-1$ **do** ▷ Sum the j-th column
    $C \leftarrow \mathsf{CMux}(A_{j,i}, \mathsf{TBSRIncr}(C), C)$;
  end for
  $\mathrm{Out}_j \leftarrow \mathsf{TBSRBitExtract}_0(C)$; ▷ Extract Out_j = the lsb
  $C \leftarrow \mathsf{TBSRDiv2}(C)$; ▷ and compute the carry
end for
for $i = 0$ to $m-1$ **do** ▷ Sum the last column
  $C \leftarrow \mathsf{CMux}(A_{d,i}, \mathsf{TBSRIncr}(C), C)$;
end for
$\mathrm{Out}_{d-1+k} \leftarrow \mathsf{TBSRBitExtract}_k(C)$ **for each** $k \in [\![0, \log_2(m)]\!]$;
return Out ▷ and output all the bits

Improving TBSR with horizontal packing.

Recall that our TRLWE ciphertext encrypt torus polynomials of degree $\leq N$, with $N = 1024$. So if the TBSR encrypts numbers in the interval $[\![0, 1027]\!]$, we need $p = \log_2(N)+1 = 11$ TRLWE ciphertexts, one per line $B_0^{(l)}, \ldots, B_p^{(l)}$ of the BSR.

In this section we propose two improvements for the TBSR encryption. The first one concerns the case when the TBSR encrypts numbers in a smaller interval (always a power of 2). The second improvement takes advantage of the fact that some lines of the TBSR are periodic/antiperiodic to reduce the number of TRLWE ciphertexts.

Algorithm 9 TBSR multiplication

Input: The bitwise TRGSW-encryptions $(A_j)_{j\in[0,d-1]}$,$(B_j)_{j\in[0,d-1]}$ of two integers a, b of d-bits each

Output: bitwise LWE encryptions of the product $\text{Out}_0, \ldots, \text{Out}_{2d-1}$.

$C \leftarrow \mathsf{TBSRSet}(0)$ ▷ The counter is initialized at 0
for $j = 0$ to $d-1$ **do** ▷ Column of the multi-addition
  for $i = 0$ to j **do** ▷ Sum the j-th column
    $C \leftarrow \mathsf{CMux}(A_i, \mathsf{CMux}(B_{j-i}, \mathsf{TBSRIncr}(C), C), C)$;
  end for
  $\text{Out}_j \leftarrow \mathsf{TBSRBitExtract}_0(C)$; ▷ Extract Out_j = the lsb
  $C \leftarrow \mathsf{TBSRDiv2}(C)$; ▷ and compute the carry
end for
for $j = d$ to $2d-3$ **do** ▷ Column of the multi-addition
  for $i = j-d+1$ to $d-1$ **do** ▷ Sum the j-th column
    $C \leftarrow \mathsf{CMux}(A_i, \mathsf{CMux}(B_{j-i}, \mathsf{TBSRIncr}(C), C), C)$;
  end for
  $\text{Out}_j \leftarrow \mathsf{TBSRBitExtract}_0(C)$; ▷ Extract Out_j = the lsb
  $C \leftarrow \mathsf{TBSRDiv2}(C)$; ▷ and compute the carry
end for
$C \leftarrow \mathsf{CMux}(A_{d-1}, \mathsf{CMux}(B_{d-1}, \mathsf{TBSRIncr}(C), C), C)$;
$\text{Out}_{2d-2} \leftarrow \mathsf{TBSRBitExtract}_0(C)$;
$\text{Out}_{2d-1} \leftarrow \mathsf{TBSRBitExtract}_1(C)$;
return *Out* ▷ Output all the bits

First improvement. If the domain of the integers is $[\![0, y-1]\!]$ where $xy = N$, we can use horizontal packing to pack x different polynomials mod $X^y + 1$ in a single TRLWE ciphertext mod $X^N + 1$. This allows to store the $p = \log_2(y) + 1$ rows of the BSR in only $\lceil p/x \rceil$ TRLWE ciphertexts.
Suppose we have to pack the BSR of a numbers $l \in [\![0, y-1]\!]$. Then, every TRLWE container packs the x rows of the BSR of l. the first container, as instance, packs $B_0^{(l)}, \ldots, B_{x-1}^{(l)}$ (each one of them contains y coefficients). The coefficients are grouped by column position:

TRLWE =

$B_{0,0}^{(l)}$	$\ldots$	$B_{x-1,0}^{(l)}$	$B_{0,1}^{(l)}$	$\ldots$	$B_{x-1,1}^{(l)}$	$\ldots$	$B_{0,y-1}^{(l)}$	$\ldots$	$B_{x-1,y-1}^{(l)}$

Then, in the increment operation we just replace the shift by X^{-1} by a multiplication by X^{-x}. This provides a factor x speedup compared to the basic scheme. For instance, if $N = 1024$ and the domain of the BSR integers is $[0, 127]$, which is enough to perform a multiplication with a 64 bit number, the 8 sequences can be packed in a single TRLWE ciphertext ($x = 8$,$y = 128$).

The same improvement can be used to pack the BSR of x different numbers $l_0 \ldots, l_{x-1} \in [\![0, y-1]\!]$, and perform the same operation to all of them (batching). In this case, every TRLWE container packs the j-th line of the BSR of each number, i.e. $B_j^{(l_0)}, \ldots, B_j^{(l_{x-1})}$: each line contains y coefficients. In the TRLWE container, the coefficients are grouped by column position, for a total of x groups:

$$\mathsf{TRLWE}_0 = \left| B_{0,0}^{(l_0)} \right| B_{0,0}^{(l_1)} \left| \ldots \right| B_{0,0}^{(l_{x-1})} \left\| B_{0,1}^{(l_0)} \right| B_{0,1}^{(l_1)} \left| \ldots \right| B_{0,1}^{(l_{x-1})} \left\| \ldots \right.$$

$$\vdots$$

$$\mathsf{TRLWE}_j = \left| B_{j,0}^{(l_0)} \right| B_{j,0}^{(l_1)} \left| \ldots \right| B_{j,0}^{(l_{x-1})} \left\| B_{j,1}^{(l_0)} \right| B_{j,1}^{(l_1)} \left| \ldots \right| B_{j,1}^{(l_{x-1})} \left\| \ldots \right.$$

$$\vdots$$

$$\mathsf{TRLWE}_p = \left| B_{p,0}^{(l_0)} \right| B_{p,0}^{(l_1)} \left| \ldots \right| B_{p,0}^{(l_{x-1})} \left\| B_{p,1}^{(l_0)} \right| B_{p,1}^{(l_1)} \left| \ldots \right| B_{p,1}^{(l_{x-1})} \left\| \ldots \right.$$

As well as before, in the increment operation we just replace the shift by X^{-1} by a multiplication by X^{-x}.

Second improvement. Even if the domain is as large as $[0, N-1]$, the first bit sequences have a small period. It is therefore possible to use the previous improvement to encode many of the first sequences as a single ciphertext, and leave the last N-antiperiodic one alone on its ciphertext.
For instance, if $N = 1024$, we observe that:

- The first 8 sequences (rows of the BSR) $B_0^{(l)}, \ldots, B_7^{(l)}$ are all 128-periodic or 128-antiperiodic (the last one), so they can be packed on a single TRLWE ciphertext.
- The next two sequences $B_8^{(l)}, B_9^{(l)}$ are 512-periodic/antiperiodic, and they can be packed on a single TRLWE ciphertext.
- The last sequence $B_{10}^{(l)}$ is 1024-antiperiodic, and stays alone in a TRLWE container.

As long as periodic sequences use the $\{0, \frac{1}{2}\}$ message space, and anti-periodic sequences use $\{-\frac{1}{4}, \frac{1}{4}\}$ (and the constant terms of f_{div2} are updated accordingly), all TBSR computations over $[0, 1023]$ can be done in only 3 TRLWE ciphertexts instead of 11, which gives a time-speedup of a factor $11/3 = 3.66$ compared to the basic scheme.

Chapter 5

Bootstrapped TFHE

The schemes and the techniques described in previous chapter can be used in a leveled context, where the depth of the circuit to be evaluated is known in advance. But this is not always the case. Several applications require computations with indefinite depth: in this case, we need to control and reduce periodically the noise growth. The bootstrapping technique, introduced by Gentry and largely improved later, is a solution to this problem.
Bootstrapping reduces the noise produced during the homomorphic computations and brings it back to a fixed level. Unfortunately, this procedure is the most expensive in the entire set of homomorphic computations. Moreover, its cost increases the more the noise needs to be reduced. At this point, the real question about bootstrapping concerns how often it has to be used to make the computations less expensive. Explicitly, it is better to perform a relatively fast bootstrapping after every elementary operation or a slow bootstrapping after the evaluation of an entire leveled circuit? Again, it depends on the application, the type of functions to be evaluated and the number of inputs and outputs.

The first bootstrapping solution we propose in this chapter is the *gate bootstrapping*, called this way because it is performed after every homomorphic binary gate (`AND`, `NAND`, `OR`, `XOR`, etc.).
In the GSW context, the bootstrapping had polynomial complexity in [GSW13] and [AP14]. The fastest bootstrapping was proposed in 2015 by Ducas and Micciancio [DM15]. They used ring operations and construct an "external" bootstrapping for LWE via GSW, while before the bootstrapping proposed worked only with GSW ciphertexts. Ducas and Micciancio build the "FHE brick": a bootstrapped `NAND` gate. It is known that by composing `NAND` gates it is possible to construct any Boolean circuit. If the homomorphic version of this gate is able to keep the noise growth always at the same level, it can be used to construct any kind of fully homomorphic circuit. Their method is incredibly fast, compared to all previous solutions. They perform a fast bootstrapping in less than a second. Of course, as the bootstrapping needs to be performed after every `NAND` gate, the evaluation of a complex circuit is

still slow, compared to the plaintext evaluation. But their result marks a turning point.
We improved the result of [DM15] in [CGGI16a] and [CGGI17a]. We made their bootstrapping faster (and called it gate bootstrapping for the first time) and we build all the bootstrapped gates (not only `NAND`). We also show that this technique has even more potential: it allows to change the message space and evaluate more complex functions.

The second bootstrapping solution we propose is the *circuit bootstrapping*, called this way because it can be used after the evaluation of a larger leveled circuit. We introduced this technique in [CGGI17a].
Circuit bootstrapping is able to convert a TLWE sample in a TRGSW ciphertext. The main reason why we needed this technique is the non-composability of some of the techniques previously presented. The `CMux` (Section 3.2.4), for instance, takes as input two TRLWE and one TRGSW ciphertexts, and outputs a TRLWE ciphertext. This implies that the TRGSW ciphertexts are always given as fresh ciphertexts. Full composability is not allowed in this case[1]. As TRGSW ciphertexts in the `CMux` evaluation encrypt a single bit, it is sufficient to efficiently convert a TLWE back to a TRGSW. This allows to produce new TRGSW ciphertexts anytime we need them and to finally make all the blocks composable. This technique, composed with the ones we described before, closes the loop, because it allows to go trough all the message spaces, potentially without limitations.
Circuit bootstrapping can be used both in the fully and in the leveled contexts. In particular, in the leveled mode, it can be seen as a support to improve leveled evaluations.

In the rest of the chapter, we detail these two techniques. Practical applications and comparisons between them are presented in Chapter 6.

5.1 Gate bootstrapping (TLWE-to-TLWE)

Given a TLWE sample $\mathsf{TLWE}_{\mathfrak{K}}(\mu) = (\mathfrak{a}, \mathfrak{b})$, the gate bootstrapping procedure constructs another TLWE encryption of μ (or of a multiple of μ, or even of a different message in $\mathbb{T}$ depending on the value of the original phase) under the same key $\mathfrak{K}$ but with a fixed amount of noise. As in [DM15], we use TRLWE as an intermediate encryption scheme to homomorphically evaluate the phase, but we use the external product from Theorem 3.2.1 (instead of the internal TRGSW product used in [DM15]) with a TRGSW encryption of the bits of the secret key.
We start by describing the bootstrapping from TLWE to TLWE, as the composition

[1] If instead the `CMux` is constructed by using the internal product, the composability is still possible.

between a blind rotation and an extraction. Then, the gate bootstrapping can be obtained by adding a final key switching.

Definition 5.1.1 (Bootstrapping key)**.** *Let $\mathfrak{K} \in \mathbb{B}^n$, $\mathfrak{K}' \in \mathbb{B}_N[X]^{n'}$ and α be a standard deviation. We define the bootstrapping key $\mathsf{BK}_{\mathfrak{K}\to\mathfrak{K}',\alpha}$ as the sequence of n TGSW samples where $\mathsf{BK}_i \in \mathsf{TRGSW}_{\mathfrak{K}',\alpha}(\mathfrak{K}_i)$.*

Algorithm 10 Bootstrapping TLWE-to-TLWE (calling algorithm 5)

Input: A constant $\mu_1 \in \mathbb{T}$, a TLWE sample $\underline{\mathfrak{c}} = (\underline{\mathfrak{a}}, \underline{\mathfrak{b}}) \in \mathsf{TLWE}_{\underline{\mathfrak{K}},\underline{\eta}}(x \cdot \frac{1}{2})$, with $x \in \mathbb{B}$ a bootstrapping key $\mathsf{BK}_{\underline{\mathfrak{K}}\to\bar{\mathfrak{K}},\bar{\alpha}} = (\mathsf{BK}_i)_{i\in[\![1,\underline{n}]\!]}$,
Output: A TLWE sample $\bar{\mathfrak{c}} = (\bar{\mathfrak{a}}, \bar{\mathfrak{b}}) = \mathsf{Bootstrap}_{\mu_1,\mathsf{BK}}(\underline{\mathfrak{c}}) \in \mathsf{TLWE}_{\bar{\mathfrak{K}},\bar{\eta}}(x \cdot \mu_1)$
1: Let $\mu_2 = \frac{1}{2}\mu_1 \in \mathbb{T}$ (Pick one of the two possible values)
2: Let $\tilde{b} = \lfloor 2\bar{N}\underline{\mathfrak{b}} \rceil$ and $\tilde{a}_i = \lfloor 2\bar{N}\underline{\mathfrak{a}}_i \rceil \in \mathbb{Z}$ **for each** $i \in [\![1, \underline{n}]\!]$
3: Let $v := (1 + X + \ldots + X^{\bar{N}-1}) \cdot X^{\frac{\bar{N}}{2}} \cdot \mu_2 \in \mathbb{T}_{\bar{N}}[X]$
4: $\mathsf{ACC} \leftarrow \mathsf{BlindRotate}((\mathbf{0}, v), (\tilde{a}_1, \ldots, \tilde{a}_n, \tilde{b}), (\mathsf{BK}_1, \ldots, \mathsf{BK}_n))$
5: Return $(\mathbf{0}, \mu_2) + \mathsf{SampleExtract}(\mathsf{ACC})$

Theorem 5.1.1 (Bootstrapping TLWE-to-TLWE)**.** *Let $\bar{H}$ be the gadget matrix in $\mathcal{M}_{(\bar{k}+1)\bar{\ell},\bar{k}+1}(\mathbb{T}_{\bar{N}}[X])$ and $Dec_{\bar{H},\bar{\beta},\bar{\epsilon}}$ its efficient approximate gadget decomposition algorithm, with quality $\bar{\beta}$ and precision $\bar{\epsilon}$ defining TRLWE and TRGSW parameters. Let $\underline{\mathfrak{K}} \in \mathbb{B}^{\underline{n}}$ and $\bar{\mathfrak{K}} \in \mathbb{B}^{\bar{n}}$ be two TLWE secret keys, and $\bar{K} \in \mathbb{B}_{\bar{N}}[X]^{\bar{k}}$ be the TRLWE interpretation of the key $\bar{\mathfrak{K}}$, and let $\bar{\alpha} \in \mathbb{R}_{\geq 0}$ be a standard deviation. Let $\mathsf{BK}_{\underline{\mathfrak{K}}\to\bar{\mathfrak{K}},\bar{\alpha}}$ be a bootstrapping key, composed by the $\underline{n}$ TRGSW encryptions $\mathsf{BK}_i \in \mathsf{TRGSW}_{\bar{K},\bar{\alpha}}(\underline{\mathfrak{K}}_i)$ for $i \in [\![1, \underline{n}]\!]$. Given one constant $\mu_1 \in \mathbb{T}$, and one sample $\underline{\mathfrak{c}} \in \mathbb{T}^{\underline{n}+1}$ whose coefficients are all multiples of $\frac{1}{2\bar{N}}$, Algorithm 10 outputs a TLWE sample $\bar{\mathfrak{c}} = \mathsf{Bootstrap}_{\mu_1,\mathsf{BK}}(\underline{\mathfrak{c}}) \in \mathsf{TLWE}_{\bar{\mathfrak{K}}}(\mu)$ where $\mu = 0$ iff. $|\varphi_{\underline{\mathfrak{K}}}(\underline{\mathfrak{c}})| < \frac{1}{4}$, $\mu = \mu_1$ otherwise and such that:*

- $\|\mathsf{Err}(\bar{\mathfrak{c}})\|_\infty \leq \underline{n}(\bar{k}+1)\bar{\ell}\bar{N}\bar{\beta}\bar{\mathcal{A}}_{\mathsf{BK}} + \underline{n}(1+\bar{k}\bar{N})\bar{\epsilon}$ *(worst case),*
- $\mathsf{Var}(\mathsf{Err}(\bar{\mathfrak{c}})) \leq \underline{n}(\bar{k}+1)\bar{\ell}\bar{N}\bar{\beta}^2\bar{\vartheta}_{\mathsf{BK}} + \underline{n}(1+\bar{k}\bar{N})\bar{\epsilon}^2$ *(average case),*

where $\bar{\mathcal{A}}_{\mathsf{BK}}$ is the amplitude of BK and $\bar{\vartheta}_{\mathsf{BK}}$ its variance s.t. $\mathsf{Var}(\mathsf{Err}(\mathsf{BK}_{\underline{\mathfrak{K}}\to\bar{K},\bar{\alpha}})) = \bar{\alpha}^2$.

Proof. By using the definitions of $\tilde{b}$ and $\tilde{\mathbf{a}}$ given at line 2 of the Algorithm 10, we define $\tilde{\varphi} := \tilde{b} - \sum_{i=1}^{\underline{n}} \tilde{a}_i s_i \mod 2\bar{N}$. We have

$$\left|\underline{\varphi} - \frac{\tilde{\varphi}}{2\bar{N}}\right| = \underline{\mathfrak{b}} - \frac{\lfloor 2\bar{N}\underline{\mathfrak{b}} \rceil}{2\bar{N}} + \sum_{i=1}^{\underline{n}} \left(\underline{\mathfrak{a}}_i - \frac{\lfloor 2\bar{N}\underline{\mathfrak{a}}_i \rceil}{2\bar{N}}\right)\underline{\mathfrak{K}}_i \leq \frac{1}{4\bar{N}} + \sum_{i=1}^{\underline{n}} \frac{1}{4\bar{N}} \leq \frac{\underline{n}+1}{4\bar{N}} = \delta. \quad (5.1)$$

And if the coefficients $\tilde{a}_1, \ldots, \tilde{a}_{\underline{n}}, \tilde{b} \in \frac{1}{2\bar{N}}\mathbb{Z}/\mathbb{Z}$, then $\underline{\varphi} = \frac{\tilde{\varphi}}{2\bar{N}}$. In all cases, $|\underline{\varphi} - \frac{\tilde{\varphi}}{2\bar{N}}| < \delta$.

At line 3, the test vector $v := (1+X+\ldots+X^{\bar{N}-1}) \cdot X^{\frac{\bar{N}}{2}} \cdot \mu_2$ is defined such that for all $p \in [0, 2N]$, the constant term of $X^{-p} \cdot v$ is either μ_2 if $p \in]\!]\frac{\bar{N}}{2}, \frac{3\bar{N}}{2}]\!]$, and $-\mu_2$ otherwise.
At line 4, a blind rotation (Algorithm 5) is applied to the test vector. The result is $\mathsf{msg}(ACC) = X^{-\tilde{\varphi}} \cdot v$ and the error (from the results shown in Theorem 4.1.3 and as the TRLWE encryption of the test vector is noiseless trivial) is:

- $\|\mathsf{Err}(\mathsf{ACC})\|_\infty \leq \underline{n}(\bar{k}+1)\bar{\ell}\bar{N}\bar{\beta}\bar{\mathcal{A}}_{\mathsf{BK}} + \underline{n}(1+\bar{k}\bar{N})\bar{\epsilon}$, in the worst case,
- $\mathsf{Var}(\mathsf{Err}(\mathsf{ACC})) \leq \underline{n}(\bar{k}+1)\bar{\ell}\bar{N}\bar{\beta}^2\bar{\vartheta}_{\mathsf{BK}} + \underline{n}(1+\bar{k}\bar{N})\bar{\epsilon}^2$, in the average case.

The SampleExtract at line 5 doesn't add any noise, so the error after bootstrapping remains the same as after the blind rotation.
After the blind rotation, the message in the accumulator is $\mathsf{msg}(ACC) = X^{-\tilde{\varphi}} \cdot v$. After SampleExtract, the message is equal to the constant term of $\mathsf{msg}(ACC)$, i.e. μ_2 if $\tilde{\varphi} \in]\!]\frac{\bar{N}}{2}, \frac{3\bar{N}}{2}]\!]$, and $-\mu_2$ otherwise. The addition with $(\mathbf{0}, \mu_2)$ makes the message equal to μ_1 if $\tilde{\varphi} \in]\!]\frac{\bar{N}}{2}, \frac{3\bar{N}}{2}]\!]$, and equal to 0 otherwise.
In other words, $|\varphi_{\mathfrak{K}}(\mathfrak{a}, \mathfrak{b})| < 1/4 - \delta$, then $-1/4 + \delta \leq \varphi_{\mathfrak{K}}(\mathfrak{a}, \mathfrak{b}) < 1/4 - \delta$, and thus using Equation (5.1), we obtain that $\tilde{\varphi} \in]\!] - \frac{N}{2}, \frac{N}{2}[\!]$ and thus, the message is equal to 0. And if $|\varphi_{\mathfrak{K}}(\mathfrak{a}, \mathfrak{b})| > 1/4 + \delta$ then $\varphi_{\mathfrak{K}}(\mathfrak{a}, \mathfrak{b}) > 1/4 + \delta$ or $\varphi_{\mathfrak{K}}(\mathfrak{a}, \mathfrak{b}) < -1/4 - \delta$ and using Equation (5.1), we obtain that the message is equal to μ_1. □

Algorithm 10 is the latest version of the bootstrapping from TLWE to TLWE, it is proposed in [CGGI17a], and it has some common points with Algorithms 1, 2 in [DM15] and Algorithm 3 in [CGGI16a].

- Like [DM15] and [CGGI16a], we rescale the computation of the phase of the input TLWE sample so that it is modulo $2N$ (line 2) and we map all the corresponding operations in the multiplicative cyclic group $\{1, X, \ldots, X^{2N-1}\}$. Since our TLWE samples are described over the real torus, the rescaling is done explicitly. This rescaling may induce an accumulated rounding error of amplitude at most $\delta \approx \sqrt{n}/4N$ in the average case and $\delta \leq (n+1)/4N$ in the worst case. In the best case, this amplitude can even be zero ($\delta = 0$) if in the actual representation of TLWE samples, all the coefficients are restricted to multiples of $\frac{1}{2N}$.
- As in [DM15] and [CGGI16a], messages are encoded as roots of unity in $\mathbb{Z}_N[X]$. Our accumulator is a TRLWE sample (as in [CGGI16a]) instead of a TRGSW sample (as in [DM15]). Also accumulator operations use the external product from Theorem 3.2.1 instead of the slower classical internal product. The test vector $(1+X+\ldots+X^{N-1})$ is embedded in the accumulator from the very start, when the accumulator is still noiseless while in [DM15], it is added at the very end. This removes a factor $\sqrt{N}$ to the final noise overhead.

- Instead of the explicit loop proposed in [DM15] and in [CGGI16a], we directly use the blind rotation Algorithm 5. As in [CGGI16a], all the TRGSW ciphertexts of $X^{-\tilde{a}_i \mathfrak{K}_i}$ required to update the accumulator internal value are computed dynamically as a very small polynomial combination of BK_i in the for loop of the Algorithm 5. This completely removes the need to decompose each $\tilde{a}_i$ on an additional base B_r, and to precompute all possibilities in the bootstrapping key. In practice, this makes our bootstrapping key 46 times smaller than in [DM15], for the exact same noise overhead. Besides, due to this squashing technique, two accumulator operations were performed per iteration instead of one in our case. This gives us an additional 2× speed up. Also, a small difference in the way we associate `CMux` operations in Algorithm 5 removes a factor 2 in the noise compared to the previous gate bootstrapping procedure in [CGGI16a], and it is also faster. The speed ups have been obtained on a 64-bit (single core) Intel Core i7-4910MQ at 2.90GHz laptop, for 159-bits of security: more details are given in Chapter 6.

The bootstrapping from Algorithm 10 takes in input a TLWE ciphertext, and depending on its phase, it outputs either a ciphertext of 0 or of μ_1 with a noise amplitude that is independent on the input. However, the input and output ciphertexts are not encrypted with the same key, since $\underline{\mathfrak{K}}$ and $\overline{\mathfrak{K}}$ have not the same parameters. The next elementary theorem fixes this by applying a key switching (Theorem 4.1.1) at the end of the bootstrapping.

Algorithm 11 Gate Bootstrapping (calling Algorithms 10 and 2)

Input: A constant $\mu_1 \in \mathbb{T}$, a TLWE sample $\underline{\mathfrak{c}} = (\underline{\mathfrak{a}}, \underline{\mathfrak{b}}) \in \mathsf{TLWE}_{\underline{\mathfrak{K}},\underline{\eta}}(x \cdot \frac{1}{2})$, with $x \in \mathbb{B}$ a bootstrapping key $\mathsf{BK}_{\underline{\mathfrak{K}} \to \bar{\mathfrak{K}},\bar{\alpha}} = (\mathsf{BK}_i)_{i \in [\![1,\underline{n}]\!]}$, a key-switching key $\mathsf{KS}_{\bar{\mathfrak{K}} \to \underline{\mathfrak{K}},\underline{\gamma},t} = \left(\mathsf{KS}_{i,j}^{(id)}\right)_{i \in [\![1,\bar{n}]\!], j \in [\![1,t]\!]}$

Output: A TLWE sample $\underline{\mathfrak{c}}' = (\underline{\mathfrak{a}}', \underline{\mathfrak{b}}') = \mathsf{GateBootstrap}_{\mu_1,\mathsf{BK},\mathsf{KS}}(\underline{\mathfrak{c}}) \in \mathsf{TLWE}_{\underline{\mathfrak{K}},\underline{\eta}}(x \cdot \mu_1)$

1: $\overline{\mathfrak{c}} \leftarrow \mathsf{BootstrappingTLWEtoTLWE}(\mu_1, \underline{\mathfrak{c}}, \mathsf{BK}_{\underline{\mathfrak{K}} \to \bar{\mathfrak{K}},\bar{\alpha}})$

2: Return $\underline{\mathfrak{c}}' \leftarrow \mathsf{PublicKeySwitchingTLWEtoTLWE}(\overline{\mathfrak{c}}, \mathrm{Identity}, \mathsf{KS}_{\bar{\mathfrak{K}} \to \underline{\mathfrak{K}},\underline{\gamma},t})$

Theorem 5.1.2 (Gate Bootstrapping TLWE-to-TLWE)**.** *Let $\bar{H}$ be the gadget matrix in $\mathcal{M}_{(\bar{k}+1)\bar{\ell},\bar{k}+1}(\mathbb{T}_{\bar{N}}[X])$ and $Dec_{\bar{H},\bar{\beta},\bar{\epsilon}}$ its efficient approximate gadget decomposition algorithm, with quality $\bar{\beta}$ and precision $\bar{\epsilon}$ defining* TRLWE *and* TRGSW *parameters. Let $\underline{\mathfrak{K}} \in \mathbb{B}^{\underline{n}}$ and $\bar{\mathfrak{K}} \in \mathbb{B}^{\bar{n}}$ be two* TLWE *secret keys, and $\bar{K} \in \mathbb{B}_{\bar{N}}[X]^{\bar{k}}$ be the* TRLWE *interpretation of the key $\bar{\mathfrak{K}}$, and let $\bar{\alpha} \in \mathbb{R}_{\geq 0}$ be a standard deviation. Let $\mathsf{BK}_{\underline{\mathfrak{K}} \to \bar{\mathfrak{K}},\bar{\alpha}}$ be a bootstrapping key, composed by the $\underline{n}$* TRGSW *encryptions $\mathsf{BK}_i \in \mathsf{TRGSW}_{\bar{K},\bar{\alpha}}(\underline{\mathfrak{K}}_i)$ for $i \in [\![1, \underline{n}]\!]$. Let $\mathsf{KS} = \mathsf{KS}_{\bar{\mathfrak{K}} \to \underline{\mathfrak{K}},\underline{\gamma},t} = \left(\mathsf{KS}_{i,j}^{(id)}\right)_{i,j}$, with $\mathsf{KS}_{i,j}^{(id)} \in \mathsf{TRLWE}_{\mathfrak{K}',\gamma}(\frac{\mathfrak{K}_i}{2^j})$ be a key-switching key defined as in Theorem 4.1.2 (with the function f equal to the identity function $id : \mathbb{T} \to \mathbb{T}$). Given one constant $\mu_1 \in \mathbb{T}$, and one sample $\underline{\mathfrak{c}} \in \mathbb{T}^{\underline{n}+1}$*

whose coefficients are all multiples of $\frac{1}{2N}$*, Algorithm 11 outputs a* TLWE *sample* $\underline{\mathfrak{c}}' = \mathsf{GateBootstrap}_{\mu_1,\mathsf{BK},\mathsf{KS}}(\underline{\mathfrak{c}}) \in \mathsf{TLWE}_{\underline{\mathfrak{K}}}(\mu)$ *where* $\mu = 0$ *iff.* $|\varphi_{\underline{\mathfrak{K}}}(\underline{\mathfrak{c}})| < \frac{1}{4}$*,* $\mu = \mu_1$ *otherwise and such that:*

- $\|\mathsf{Err}(\underline{\mathfrak{c}}')\|_\infty \leq \underline{n}(\bar{k}+1)\bar{\ell}\bar{N}\bar{\beta}\bar{\mathcal{A}}_{\mathsf{BK}} + \underline{n}(1+\bar{k}\bar{N})\bar{\epsilon} + \bar{n}2^{-(t+1)} + t\bar{n}\underline{\mathcal{A}}_{\mathsf{KS}}$ *(worst case),*
- $\mathsf{Var}(\mathsf{Err}(\underline{\mathfrak{c}}')) \leq \underline{n}(\bar{k}+1)\bar{\ell}\bar{N}\bar{\beta}^2\bar{\vartheta}_{\mathsf{BK}} + \underline{n}(1+\bar{k}\bar{N})\bar{\epsilon}^2 + \bar{n}2^{-2(t+1)} + t\bar{n}\underline{\vartheta}_{\mathsf{KS}}$ *(average case),*

where $\bar{\mathcal{A}}_{\mathsf{BK}}$ *and* $\bar{\vartheta}_{\mathsf{BK}} = \bar{\alpha}^2$ *are respectively the amplitude and the variance of the error of* BK*, and* $\underline{\mathcal{A}}_{\mathsf{KS}}$ *and* $\underline{\vartheta}_{\mathsf{KS}} = \underline{\gamma}^2$ *are respectively the amplitude and the variance of the error of* KS*.*

The idea of bootstrapping can be summarized in Figure 5.1. The torus is divided in $2N$ slices and an output value, encoded in the anticyclic test vector v ($v_{N+i} = -v_i$), is assigned to every slice. In the bootstrapping we just described, only 2 values (μ_2 and its opposite) are assigned to the coefficients of the test vector. But the technique can be generalized to multiple values and it can be used to evaluate different functions.

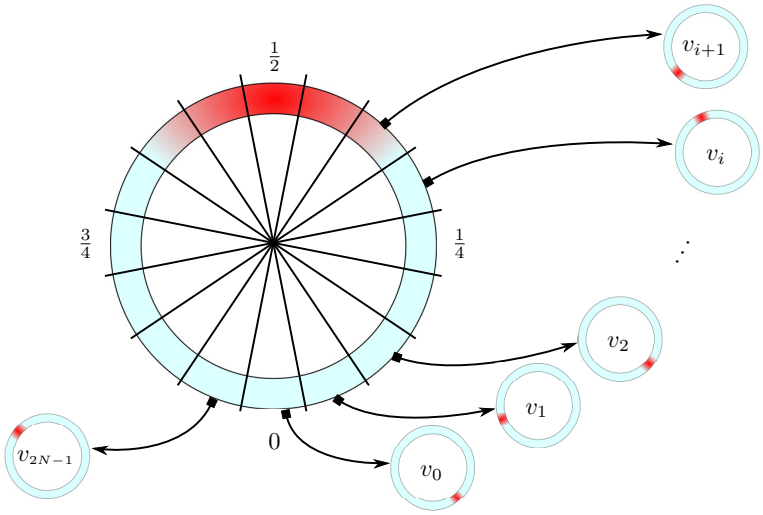

Figure 5.1: Bootstrapping - *depending on where it is the phase of the input* TLWE *sample (big torus), the bootstrapping selects a new output* TLWE *sample (i.e. one of the small torus).*

Fully Homomorphic Boolean Gates.

As in [DM15], the homomorphic evaluation of a `NAND` gate between TLWE samples is achieved with 2 additions (one with a noiseless trivial sample) and a gate bootstrapping (Algorithm 11).
We chose the parameters such that $\mathsf{Var}(\mathsf{Err}(\underline{\mathfrak{c}}')) < \frac{1}{16}$ and we denote as $\underline{\mathfrak{c}}' = \mathsf{GateBootstrap}_{\frac{1}{4},\mathsf{BK},\mathsf{KS}}(\underline{\mathfrak{c}})$ the output of the gate bootstrapping (Algorithm 11,

with $\mu_1 = \frac{1}{4}$).

Let us consider two TLWE samples $\underline{\mathfrak{c}}_1$ and $\underline{\mathfrak{c}}_2$, with message space $\{0, 1/4\}$ and $\|\mathsf{Err}(\underline{\mathfrak{c}}_1)\|_\infty, \|\mathsf{Err}(\underline{\mathfrak{c}}_2)\|_\infty \leq \frac{1}{16}$. The result of the bootstrapped NAND gate is obtained by computing $\underline{\mathfrak{c}} = (\mathbf{0}, \frac{5}{8}) - \underline{\mathfrak{c}}_1 - \underline{\mathfrak{c}}_2$, plus a gate bootstrapping. Indeed the possible values for the messages of $\underline{\mathfrak{c}}$ are $\frac{5}{8}, \frac{3}{8}$ (upper part of the torus, encoding 1/2) if either $\underline{\mathfrak{c}}_1$ or $\underline{\mathfrak{c}}_2$ encode 0, and $\frac{1}{8}$ (lower part of the torus, encoding 0) if both encode $\frac{1}{4}$. Since the noise amplitude $\|\mathsf{Err}(\underline{\mathfrak{c}})\|_\infty$ is $< \frac{1}{8}$, then $|\varphi_{\mathfrak{K}}(\underline{\mathfrak{c}})| > \frac{1}{4}$ if and only if $\mathrm{NAND}(\mathsf{msg}(\underline{\mathfrak{c}}_1), \mathsf{msg}(\underline{\mathfrak{c}}_2)) = 1$. This explains why it suffices to bootstrap $\underline{\mathfrak{c}}$ with parameters $\mu_1 = \frac{1}{4}$ to get the answer.
By using a similar approach, it is possible to directly evaluate with a single bootstrapping all the basic gates:

- $\mathrm{HomNOT}(\underline{\mathfrak{c}}) = (\mathbf{0}, \frac{1}{4}) - \underline{\mathfrak{c}}$ (no bootstrapping is needed);
- $\mathrm{HomAND}(\underline{\mathfrak{c}}_1, \underline{\mathfrak{c}}_2) = \mathsf{GateBootstrap}_{\frac{1}{4},\mathsf{BK},\mathsf{KS}}\left((\mathbf{0}, -\frac{1}{8}) + \underline{\mathfrak{c}}_1 + \underline{\mathfrak{c}}_2\right)$;
- $\mathrm{HomNAND}(\underline{\mathfrak{c}}_1, \underline{\mathfrak{c}}_2) = \mathsf{GateBootstrap}_{\frac{1}{4},\mathsf{BK},\mathsf{KS}}\left((\mathbf{0}, \frac{5}{8}) - \underline{\mathfrak{c}}_1 - \underline{\mathfrak{c}}_2\right)$;
- $\mathrm{HomOR}(\underline{\mathfrak{c}}_1, \underline{\mathfrak{c}}_2) = \mathsf{GateBootstrap}_{\frac{1}{4},\mathsf{BK},\mathsf{KS}}\left((\mathbf{0}, \frac{1}{8}) + \underline{\mathfrak{c}}_1 + \underline{\mathfrak{c}}_2\right)$;
- $\mathrm{HomXOR}(\underline{\mathfrak{c}}_1, \underline{\mathfrak{c}}_2) = \mathsf{GateBootstrap}_{\frac{1}{4},\mathsf{BK},\mathsf{KS}}\left(2 \cdot (\underline{\mathfrak{c}}_1 - \underline{\mathfrak{c}}_2)\right)$.

Remark 5.1.1. *The* $\mathrm{HomXOR}(\underline{\mathfrak{c}}_1, \underline{\mathfrak{c}}_2)$ *gate can be achieved also by performing* $\mathsf{GateBootstrap}_{\frac{1}{4},\mathsf{BK},\mathsf{KS}}\left(2 \cdot (\underline{\mathfrak{c}}_1 + \underline{\mathfrak{c}}_2)\right)$. *Other binary homomorphic gates can be constructed by following the same footprint.*

Remark 5.1.2. *The term gate bootstrapping refers to the fact that this fast bootstrapping is performed after every gate evaluation, but it can be used even if we do not need to evaluate a specific gate and we just want to refresh noisy ciphertexts.*

The ternary MUX gate ($\mathtt{MUX}(c, d_0, d_1) = c?d_1{:}d_0 = (c \wedge d_1) \oplus ((1 - c) \wedge d_0)$, for $c, d_0, d_1 \in \mathbb{B}$) is generally expressed as a combination of 3 binary gates. As already mentioned in [DM15], we can improve the MUX evaluation by performing the middle $\oplus$ as a regular addition before the final $\mathsf{KeySwitching}$. Indeed, this xor has at most one operand which is true, and at this location, it only affects a negligible amount of the final noise. Overall, the ternary MUX gate can be evaluated in FHE mode by evaluating only two bootstrappings and one public key switching. We call this procedure *native* MUX, and we note it HomMUX as the other bootstrapped homomorphic gates, which computes:

- $c \wedge d_1$ via a bootstrapping (Algorithm 10):

$$\bar{\mathfrak{c}}_1 = \mathsf{Bootstrap}_{\frac{1}{4},\mathsf{BK}}\left(\left(\mathbf{0}, -\frac{1}{8}\right) + \underline{\mathfrak{c}} + \underline{\mathfrak{d}}_1\right);$$

- $(1-c) \wedge d_0$ via a bootstrapping (Algorithm 10):

$$\bar{\mathfrak{c}}_2 = \mathsf{Bootstrap}_{\frac{1}{4},\mathsf{BK}}\left(\left(\mathbf{0}, \frac{1}{8}\right) - \underline{\mathfrak{c}} + \underline{\mathfrak{d}}_0\right);$$

- A final public key switching (Algorithm 2) of the sum $\bar{\mathfrak{c}}_1 + \bar{\mathfrak{c}}_2$, which dominates the noise:

$$\underline{\mathfrak{d}} = \mathsf{PublicKS}_{\mathsf{KS}}^{(Id)}(\bar{\mathfrak{c}}_1 + \bar{\mathfrak{c}}_2).$$

Figure 5.2 summarizes the general idea.

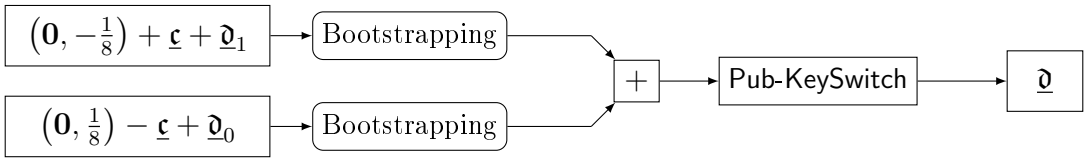

Figure 5.2: *Native* $MUX(\mathbf{c}, \mathbf{d_0}, \mathbf{d_1})$

This HomMUX is therefore bootstrappable with the same parameters for the other binary gates. The following lemma, which is a direct consequence of of Theorem 5.1.1 and Theorem 4.1.1, gives the explicit noise analysis.

Lemma 5.1.1 (Bootstrapped homomorphic native MUX gate)**.** *Under the same hypothesis of Theorem 5.1.1 and 4.1.1, assuming* $\underline{\mathfrak{c}}, \underline{\mathfrak{d}}_0, \underline{\mathfrak{d}}_1 \in \mathbb{T}^{\underline{n}+1}$ *are three* TLWE *samples whose coefficients are all multiples of* $\frac{1}{2\bar{N}}$*, Then, the result of the evluation of* $\mathtt{MUX}(\underline{\mathfrak{c}}, \underline{\mathfrak{d}}_1, \underline{\mathfrak{d}}_0)$ *is a* TLWE *sample* $\underline{\mathfrak{d}} \in \mathsf{TLWE}_{\underline{\mathfrak{K}}}((x?x_1 : x_0) \cdot \frac{1}{4})$ *such that*

- $\|\mathsf{Err}(\bar{\mathfrak{c}})\|_\infty \leq 2 \cdot (\underline{n}(\bar{k}+1)\bar{\ell}\bar{N}\bar{\beta}\bar{\mathcal{A}}_{\mathsf{BK}} + \underline{n}(1+\bar{k}\bar{N})\bar{\epsilon}) + \bar{n}2^{-(t+1)} + t\bar{n}\underline{\mathcal{A}}_{\mathsf{KS}}$ *(worst case),*

- $\mathsf{Var}(\mathsf{Err}(\bar{\mathfrak{c}})) \leq 2 \cdot (\underline{n}(\bar{k}+1)\bar{\ell}\bar{N}\bar{\beta}^2\bar{\vartheta}_{\mathsf{BK}} + \underline{n}(1+\bar{k}\bar{N})\bar{\epsilon}^2) + \bar{n}2^{-2(t+1)} + t\bar{n}\underline{\vartheta}_{\mathsf{KS}}$ *(average case),*

where $\bar{\mathcal{A}}_{\mathsf{BK}}$ *and* $\bar{\vartheta}_{\mathsf{BK}} = \bar{\alpha}^2$ *are respectively the amplitude and the variance of the error of the bootstrapping key* BK*, and* $\underline{\mathcal{A}}_{\mathsf{KS}}$ *and* $\underline{\vartheta}_{\mathsf{KS}}$ *are respectively the amplitude and the variance of the error of the key-switching key* KS*.*

In the rest of the paper, when we compare different homomorphic techniques, we refer to the gate bootstrapping mode as the technique consisting in evaluating small circuits expressed by using these bootstrapped gates.

5.2 Circuit bootstrapping (TLWE-to-TRGSW)

In the previous chapter, we presented efficient leveled algorithms to evaluate some arithmetic operations. These techniques are more efficient than before because they use the external product 3.2.10 from [CGGI16a] and [BP16] instead of the classical internal TRGSW product, but the inputs (TRLWE and TRGSW) and output (only TRLWE) have different types. As a consequence, we can not always compose these operations, like in a usual algorithm. A possible solution to solve this problem is to have an efficient bootstrapping between TLWE and TRGSW ciphertexts: TLWE is sufficient, as in our applications the TRGSW ciphertexts encrypt only a bit of information.
Fast bootstrapping procedures have been proposed in [DM15] and [CGGI16a]. Unfortunately, these bootstrappings cannot output GSW ciphertexts. On the other hand, previous solutions proposed in [GSW13], [AP14] and [GINX14] based on the internal product are not practical. In this section, we propose an efficient technique to convert back TLWE ciphertexts to TRGSW (running in just 137ms, on a 64-bit (single core) Intel Core i7-4910MQ at 2.90GHz laptop, for 152-bits of security, see Chapter 6) that we call *circuit bootstrapping*.

Our goal is to convert a TLWE sample with large noise amplitude over some binary message space (e.g amplitude $\frac{1}{4}$ over $\{0, \frac{1}{2}\}$), into a TRGSW sample with a low noise amplitude over the integer message space $\{0, 1\}$.
In all previous constructions, the TLWE decryption consists in a circuit, which is then evaluated using the internal addition and multiplication laws over TRGSW ciphertexts. The target TRGSW ciphertext is thus the result of an arithmetic expression over TRGSW ciphertexts. Instead, we propose a more efficient technique, which reconstructs the target directly from its very sparse internal structure. Namely, a TRGSW ciphertext C of a message $\mu \in \{0, 1\}$ is a vector of $(k+1)\ell$ TRLWE ciphertexts.

$$C = Z + \mu H = \left(\begin{array}{c} \mathsf{TRLWE}_K(0) \\ \vdots \\ \mathsf{TRLWE}_K(0) \\ \hline \vdots \\ \hline \mathsf{TRLWE}_K(0) \\ \vdots \\ \mathsf{TRLWE}_K(0) \end{array}\right) + \mu \cdot \left(\begin{array}{c|c|c} 1/B_g & \dots & 0 \\ \vdots & \ddots & \vdots \\ 1/B_g^\ell & \dots & 0 \\ \hline \vdots & \ddots & \vdots \\ \hline 0 & \dots & 1/B_g \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1/B_g^\ell \end{array}\right)$$

Each of these TRLWE ciphertexts encrypts the same message as $\mu \cdot \mathbf{h}_i$, where $\mathbf{h}_i$ is the corresponding line of the gadget matrix H. We denote the rows of the TRGSW ciphertext as $c^{(u,w)}$, where u is the bloc index ($u = 1, \dots, k+1$) and w is the line index ($w = 1, \dots, \ell$) in the u-th bloc of C. Then, the line $c^{(u,w)}$ encrypts the message

is $-\mu \cdot \frac{K_u}{B_g^w}$, where K_u is the u-th polynomial of the secret TRLWE/TRGSW key K and $K_{k+1} = -1$.
In order to do the reconstruction (line by line), we can use ℓ times the TLWE-to-TLWE bootstrapping (Algorithm 10) to obtain a TLWE sample of each message in $\{\mu B_g^{-1}, \ldots, \mu B_g^{-\ell}\}$. Then we use the private key switching (Algorithm 3) technique to "multiply" these ciphertexts by the secret $-K_u$, to reconstruct the correct message.

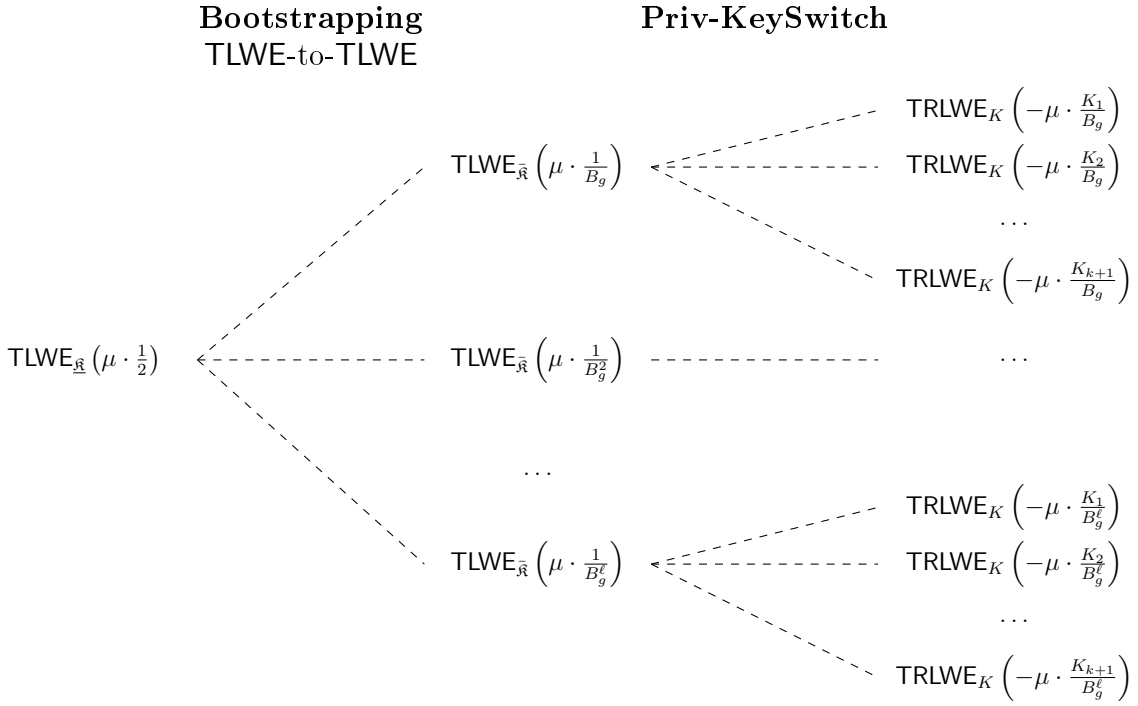

Figure 5.3: Circuit bootstrapping

Once each line has been constructed, we just compose them in the right order to obtain the TRGSW encryption of the message $\mu \in \mathbb{B}$.

Our circuit bootstrapping, detailed in Algorithm 12, crosses 3 levels of noise and encryption. Each level has its own key and parameters set. In order to distinguish the different levels, we use an intuitive notation with bars. The upper bar will be used for level 2 variables, the under bar for the level 0 variables and level 1 variables will remain without any bar. The main difference between the three levels of encryption is the amount of noise supported. Indeed, the higher the level is, the smaller is the noise. Level 0 corresponds to ciphertexts with very large noise (about $\underline{\alpha} \approx 2^{-15}$). Level 0 parameters are very small, computations are almost instantaneous, but only a very limited amount of linear operations are tolerated. Level 1 corresponds to medium noise (about $\alpha \approx 2^{-30}$). Ciphertexts in level 1 have medium size parameters, which allows to execute relatively fast operations, and for instance a leveled

homomorphic evaluation of a relatively large automata. Level 2 corresponds to ciphertexts with small noise (about $\bar{\alpha} \approx 2^{-45}$). This level corresponds to the limit of what can be mapped over native 64-bit operations. Figure 5.4 summarizes our levels (not corresponding to the classical multiplicative depth) and the operations we perform in each one of them. Practical values and further details are given in Chapter 6.

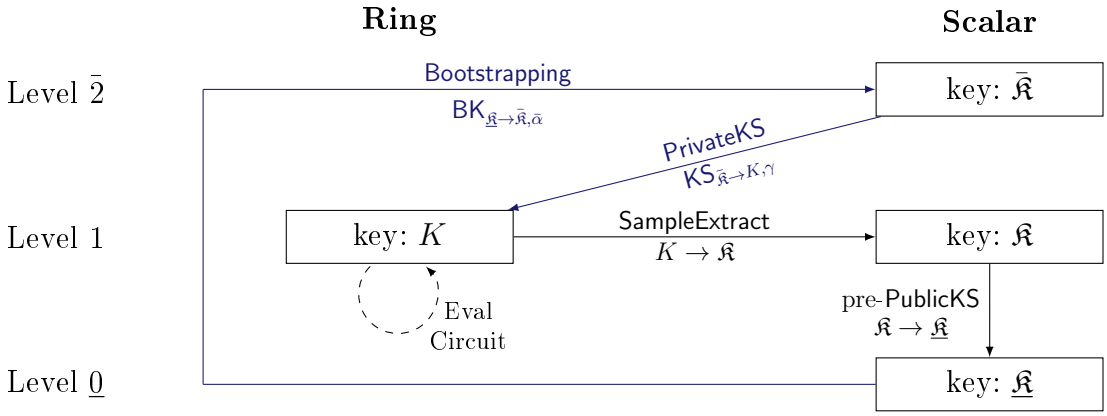

Figure 5.4: *The figure represents the three levels of encryption on which our construction operates. The arrows show the operations that can be performed inside each level or how to move from a level to another. In order to distinguish the objects with respect to their level, we adopted the intuitive notations "superior bar" for level 2, "no bar" for level 1 and "under bar" for level 0. We highlight in blue the different stages of the circuit bootstrapping (whose detailed description is given below). The leveled homomorphic circuits are evaluated at level 1. Once we want to perform a circuit bootstrapping, we perform a sample extract (from* TRLWE *to* TLWE*) and a pre-public key switching to bring the extracted* TLWE *at level 0.*

As we mentioned before (and as we illustrate in Figure 5.3), our circuit bootstrapping consists in two main parts: bootstrapping and private key switching. An additional pre-public key switching have to be performed before the circuit bootstrapping to bring the TLWE sample to level 0. In fact, the main leveled circuit evaluations are done at level 1.

1. TLWE**-to-**TLWE **pre-public key switching**: The input of the algorithm is a TLWE sample at level 1 with a large noise amplitude over the message space $\{0, \frac{1}{2}\}$. Without loss of generality, it can be key switched to a level 0 TLWE ciphertext $\underline{\mathfrak{c}} = (\underline{\mathfrak{a}}, \underline{\mathfrak{b}}) \in \mathsf{TLWE}_{\underline{\mathfrak{K}}, \underline{\eta}}(\mu \cdot \frac{1}{2})$, of a message $\mu \in \mathbb{B}$ with respect to the small secret key $\underline{\mathfrak{K}} \in \mathbb{B}^{\underline{n}}$ and a large standard deviation $\underline{\eta} \in \mathbb{R}$ (typically, $\underline{\eta} \approx 2^{-5}$ to guaranty correct decryption with overwhelming probability). This step is standard.

2. TLWE-to-TLWE **bootstrapping**: Given a level 2 bootstrapping key $\mathsf{BK}_{\underline{\bar{\mathfrak{K}}}\to\bar{\mathfrak{K}},\bar{\alpha}} = (\mathsf{BK}_i)_{i\in[\![1,\underline{n}]\!]}$ where $\mathsf{BK}_i \in \mathsf{TRGSW}_{\bar{K},\bar{\alpha}}(\underline{\mathfrak{K}}_i)$, we use ℓ times the TLWE-to-TLWE bootstrapping algorithm (Algorithm 10) on $\underline{\mathfrak{c}}$, to obtain ℓ TLWE ciphertexts $\bar{\mathfrak{c}}^{(1)}, \dots, \bar{\mathfrak{c}}^{(\ell)}$ where $\bar{\mathfrak{c}}^{(w)} \in \mathsf{TLWE}_{\bar{\mathfrak{K}},\bar{\eta}}(\mu \cdot \frac{1}{B_g{}^w})$, with respect to the level 2 secret key $\bar{\mathfrak{K}} \in \mathbb{B}^{\bar{n}}$, and with a fixed noise parameter $\bar{\eta} \in \mathbb{R}$ which does not depend on the input noise. If the bootstrapping key has a level 2 noise $\bar{\alpha}$, we expect the output noise $\bar{\eta}$ to remain smaller than level 1 value.

3. TLWE-to-TRLWE **private key switching**: Finally, to reconstruct the final TRGSW ciphertext of μ, we simply need to craft a TRLWE ciphertext which has the same phase as $\mu \cdot \mathbf{h_i}$, for each row of the gadget matrix H. Since $\mathbf{h_i}$ contains only a single non-zero constant polynomial in position $u \in [\![1, k+1]\!]$ whose value is $\frac{1}{B_g^w}$ where $w \in [\![1, \ell]\!]$, the phase of $\mu \cdot \mathbf{h_i}$ is $-\mu \cdot \frac{K_u}{B_g^w}$ where K_u is the u-th term of the key K. If we call f_u the (secret) morphism from $\mathbb{T}$ to $\mathbb{T}_N[X]$ defined by $f_u(x) = -K_u \cdot x$, we just need to apply f_u homomorphically to the TLWE sample $\bar{\mathfrak{c}}^{(w)}$ to get the desired TRLWE sample. Since f_u is 1-lipschitzian (for the infinity norm), this operation can be done with additive noise overhead via the private functional key switching (Algorithm 3).

Algorithm 12 Circuit Bootstrapping (calling algorithms 10 and 3)

Input: A level 0 TLWE sample $\underline{\mathfrak{c}} = (\underline{\mathfrak{a}}, \underline{\mathfrak{b}}) \in \mathsf{TLWE}_{\underline{\mathfrak{K}},\underline{\eta}}(\mu \cdot \frac{1}{2})$, with $\mu \in \mathbb{B}$, a bootstrapping key $\mathsf{BK}_{\underline{\mathfrak{K}}\to\bar{\mathfrak{K}},\bar{\alpha}} = (\mathsf{BK}_i \in \mathsf{TRGSW}_{\bar{K},\bar{\alpha}}(\underline{\mathfrak{K}}_i))_{i\in[\![1,\underline{n}]\!]}$, $k+1$ private key-switching keys $\mathsf{KS}^{(f_u)}_{\bar{\mathfrak{K}}\to K,\gamma}$ corresponding to the functions $f_u(x) = -K_u \cdot x$ when $u \le k$, and $f_{k+1}(x) = 1 \cdot x$.
Output: A level 1 TRGSW sample $C \in \mathsf{TRGSW}_{K,\eta}(\mu)$

```
for w = 1 to ℓ
    c̄^(w) ← Bootstrapping_{BK, 1/B_g^w}(c)
    for u = 1 to k + 1
        c^(u,w) = PrivKS(KS^(f_u), c̄^(w))
Return  C = (c^(u,w))_{1≤u≤k+1, 1≤w≤ℓ}
```

Theorem 5.2.1 (Circuit Bootstrapping Theorem)**.** *Let $n, \alpha, N, k, B_g, \ell, H, \epsilon$ denote* TRLWE/TRGSW *level 1 parameters, and the same variables names with underbars/upper-bars for level 0 and 2 parameters. Let $\underline{\mathfrak{K}} \in \mathbb{B}^{\underline{n}}$, $\mathfrak{K} \in \mathbb{B}^n$ and $\bar{\mathfrak{K}} \in \mathbb{B}^{\bar{n}}$, be a level 0, 1 and 2* TLWE *secret keys, and $\underline{K}, K, \bar{K}$ their respective* TRLWE *interpretation. Let $\mathsf{BK}_{\underline{\mathfrak{K}}\to\bar{\mathfrak{K}},\bar{\alpha}}$ be a bootstrapping key, composed by the $\underline{n}$* TRGSW *encryptions $\mathsf{BK}_i \in \mathsf{TRGSW}_{\bar{K},\bar{\alpha}}(\underline{\mathfrak{K}}_i)$ for $i \in [\![1, \underline{n}]\!]$. For each $u \in [\![1, k+1]\!]$, let f_u be the morphism from $\mathbb{T}$ to $\mathbb{T}_N[X]$ defined by $f_u(x) = -K_u \cdot x$, and $\mathsf{KS}^{f_u}_{\bar{\mathfrak{K}}\to K,\gamma} = (\mathsf{KS}^{(u)}_{i,j} \in \mathsf{TRLWE}_{K,\gamma}((\bar{\mathfrak{K}}_i K_u \cdot 2^{-j})))_{i\in[\![1,\bar{n}]\!],j\in[\![1,t]\!]}$ be the corresponding private key-switching key.*

Given a level 0 TLWE *sample* $\underline{\mathfrak{c}} = (\underline{\mathfrak{a}}, \underline{\mathfrak{b}}) \in \mathsf{TLWE}_{\underline{\mathfrak{K}}}(\mu \cdot \frac{1}{2})$*, with* $\mu \in \mathbb{B}$*, the algorithm 12 outputs a level 1* TRGSW *sample* $C \in \mathsf{TRGSW}_K(\mu)$ *such that*

- $\|\mathsf{Err}(C)\|_\infty \leq \underline{n}(\bar{k}+1)\bar{\ell}\bar{N}\bar{\beta}\mathcal{A}_{\mathsf{BK}} + \underline{n}(1+\bar{k}\bar{N})\bar{\epsilon} + \bar{n}2^{-(t+1)} + \bar{n}t\mathcal{A}_{\mathsf{KS}}$ *(worst);*
- $\mathsf{Var}(\mathsf{Err}(C)) \leq \underline{n}(\bar{k}+1)\bar{\ell}\bar{N}\bar{\beta}^2\bar{\vartheta}_{\mathsf{BK}} + \underline{n}(1+\bar{k}\bar{N})\bar{\epsilon}^2 + \bar{n}2^{-2(t+1)} + \bar{n}t\vartheta_{\mathsf{KS}}$ *(average).*

Here $\bar{\vartheta}_{\mathsf{BK}} = \bar{\alpha}^2$ *and* $\mathcal{A}_{\mathsf{BK}}$ *is the variance and amplitude of* $\mathsf{Err}(\mathsf{BK}_{\underline{\mathfrak{K}}\to\bar{K},\bar{\alpha}})$*, and* $\vartheta_{\mathsf{KS}} = \gamma^2$ *and* $\mathcal{A}_{KS}$ *are the variance and amplitude of* $\mathsf{Err}(\mathsf{KS}_{\bar{\mathfrak{K}}\to K,\gamma})$*.*

Proof. The output TRGSW ciphertext is correct, because by construction, the i-th TRLWE component $\mathbf{c}^{(u,w)}$ has the correct message $\mathsf{msg}(\mu \cdot \mathbf{h}_i) = -\mu \cdot \frac{K_u}{B_g^w}$. The row $\mathbf{c}^{(u,w)}$ is obtained by chaining one TLWE-to-TLWE bootstrapping (Algorithm 10) with one private key switching (Algorithm 3). The values of maximal amplitude and variance of $\mathsf{Err}(C)$ are directly obtained from the partial results of Theorem 5.1.1 and Theorem 4.1.2. In total, Algorithm 12 performs exactly ℓ bootstrappings (Algorithm 10), and $\ell(k+1)$ private key switchings (Algorithm 3). □

Circuit bootstrapping closes the loop that was left open in [CGGI16a]. At this point, all the message spaces can be linked one to each other by using one of the techniques previously described. Figure 5.5 summarizes all these connections and operations.

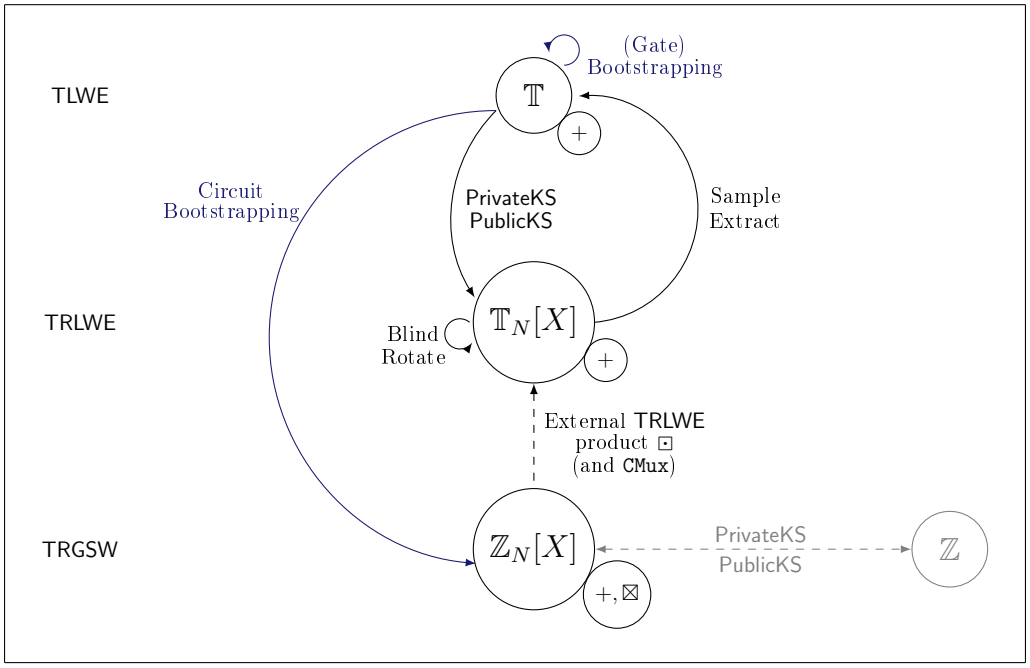

Figure 5.5: *Homomorphic operations: view from the message spaces. Gate and circuit bootstrappings are highlighted.*

Chapter 6

Security estimates, practical results and implementation

This chapter is dedicated to the practical implementation of our TFHE scheme.
The first part concerns the semantic security analysis of TFHE: in this section we express the security parameter λ only as a function depending on the entropy of the secret key and on the noise level (and not on all other parameters). Our security analysis is completely based on previous results from the state of the art and it is fundamental in order to choose the practical parameters.
In the second part we describe in detail our TFHE implementation, which is now a C/C++ library available in open-source, while in the third part we explicit the parameters we used in the implementation and the execution timings for several practical use cases.

6.1 Semantic security

We start this section by shortly recalling the basics on lattices, then we continue with the semantic security analysis of TFHE.

Lattices in short.

A lattice is a discrete additive subgroup of $\mathbb{R}^n$. Let $\mathbf{b_1}, \ldots, \mathbf{b_n} \in \mathbb{R}^n$ be linearly independent vectors. Then a lattice is a set of points $L = \{a_1\mathbf{b_1} + \ldots + a_n\mathbf{b_n} \mid a_i \in \mathbb{Z}\}$. The set $B = \{\mathbf{b_1}, \ldots, \mathbf{b_n}\}$ is a (not unique) basis of the lattice L. A lattice can be also denoted by $L(B)$, where

$$B = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \cdots & b_{n,n} \end{pmatrix} \in \mathcal{M}_{n\times n}(\mathbb{R})$$

and the rows are the entries of the vectors $\mathbf{b_1}, \ldots, \mathbf{b_n}$ of the basis. So $L(B) = \{\mathbf{a}B \mid \mathbf{a} \in \mathbb{Z}^n\}$. As already said, a basis is not unique.
Two bases B and B' are equivalent, i.e. they generate the same lattice, if and only if $B' = U \cdot B$, where U is an integer unimodular matrix, i.e. integer matrix with determinant equal to ± 1. The multiplication of B with an unimodular matrix is equivalent to row operations like permutations, negations or additive integer combinations.

A lattice can be visualized as a grid of points repeated with constant period. This constant period is actually a parallelepiped, called *fundamental parallelepiped*: it is a fundamental region of space (with points of the lattice as vertex) which contains only one representant of each point. Each fundamental parallelepiped has the same volume, equal to the determinant of $L(\mathbf{B})$:

$$det(L) := |det(\mathbf{B})| = \text{ volume of the fundamental parallelepiped.}$$

Next to the determinant, it is also important to define the successive minima $\lambda_k(L)$, equal to the smallest radius of a ball containing k linearly independent vectors. In the particular case of $k = 1$ we have $\lambda_1(L)$, which is equal to the (euclidean) length of a shortest non-zero vector of L. The problem of finding a shortest vector in a lattice is considered hard to solve, as well as the problem of finding the closest vector to some target. We better define these hard problems, by giving their approximate version (parametrized by γ):

- *Shortest Vector Problem (SVP_γ)*: given a basis B, find a vector $\mathbf{v} \in L(B)$ such that $length(\mathbf{v}) \leq \gamma \cdot \lambda_1(L)$ (i.e. find the shortest vector in a lattice).
- *Shortest Independent Vector Problem ($SIVP_\gamma$)*: given a basis B, find k linearly independent vectors in $L(B)$ of length $\leq \gamma \cdot \lambda_k(L)$.
- *Closest Vector Problem (CVP_γ)*: given a basis B and a point V (not necessarily in the lattice), find a lattice vector whose distance to V is at most γ.

A lattice basis is not unique. We distinguish two kind of bases: good (or reduced) bases and bad bases, where good bases are the ones composed by short and almost orthogonal vectors. Having a good basis makes the lattice problems easier to solve. It is for this reason that when a bad basis is given, we try to reduce it to a good basis.
The most known lattice basis reduction algorithms are LLL, a 1982 polynomial time algorithm by Lenstra, Lenstra and Lovász [LLL82], and BKZ (Blockwise-Korkine-Zolotarev) by Schnorr and Euchner [SE94]. LLL uses Gram-Schmidt orthogonalization and swaps vectors until the Lovász condition is satisfied. As a result, it produces a new basis for a lattice that is size reduced (i.e. has Gram-Schmidt coefficients $|\mu_{i,j}| \leq \frac{1}{2}$) and satisfies Lovász condition (i.e. $\delta\|\mathbf{b}_i\|^2 \leq \|\mathbf{b}_{i+1} + \mu_{i+1,i}\mathbf{b}_i\|^2$,

for a certain $\delta \in (\frac{1}{4}, 1]$). BKZ reduces the basis by calling successively LLL and a SVP oracle on blocks of basis vectors[1]. The algorithm has been subjected to many improvements: extreme pruning [GNR10], early termination [HPS11], enumeration subroutine [Kan83], etc. All these improvements have been collected and implemented in BKZ 2.0.
The quality of an output basis by lattice reduction is measured by the root-Hermite factor, noted δ_0: the shortest non-zero vector $\mathbf{b}$ of the reduced basis is such that $\|\mathbf{b}\| \leq \delta_0^n \cdot Vol(L)^{\frac{1}{n}}$. This factor is estimated $\delta_0 \in (1, 1.1]$, depending on the blocksize [APS15].

6.1.1 Security analysis

The TFHE scheme is a GSW-based construction, whose security relies on the hardness of the LWE problem.
In fact, on the asymptotic side, TLWE samples can be equivalently rescaled and rounded to their closest binary-LWE representative, which in turn can be reduced to standard LWE with full secret using the modulus-dimension reduction from [BLP+13], or the group-switching techniques from [GINX14]. Therefore, the semantic security of TFHE is asymptotically equivalent to worst-case lattice problems.
At the time of writing, there is still no reduction between the RingLWE and the binary-RingLWE instances. There is instead an asymptotic equivalence between ring [SSTX09, LPR10] and module [LS15] LWE problems for large modulus [AD17].

In this section, we rather focus on the practical hardness of TFHE, and express its effective security parameter λ directly as a function of the entropy of the secret (noted n) and the error standard deviation (noted α).
Our analysis is based on the methodology proposed in [APS15] and [Alb17]. In their work, they review many classes of attacks against LWE:

- A direct BDD[2] approach with standard lattice reduction. Given many homogeneous LWE samples, it is possible to construct a lattice by using the masks of the samples. Then, the right terms are close points to the lattice. The goal is to find the nearest lattice point: once this point has been found, obtaining the secret demands to solve a linear system.

- Sieving algorithms, used to solve the SVP.

- A variant of the BKW methods [BKW03]. Indeed, the original method is used

[1] BKZ-2.0 uses running time and quality predictions to optimize the placement and block sizes and pruning in the SVP oracles during the reduction.

[2] *Bounded Distance Decoding* problem: given a lattice basis and point that is already pretty close to the lattice, find the closest lattice point (similar to the Closest Vector Problem).

to solve the Learning Parity with Noise problem, but it can be adapted to LWE.

- Meet in the middle attacks, proposing a better time-memory trade-off than naive brute force attacks.

In general, they found out that there is no single-best attack against all possible parameters. However, according to their results table [APS15, Section 8, Tables 7,8], for the range of dimensions and noise used for FHE, it appears that the SIS-distinguisher attack is often the most efficient attack. Since the modulus q is not a parameter in our definition of TLWE, we need to adapt their results.

We rapidly recall the *SIS* (Short Integer Solution) problem, presented in 1996 by Ajtai [Ajt96]: given random vectors $\mathbf{a_1}, \ldots, \mathbf{a_m} \in \mathbb{Z}_q^n$, find a non-trivial solution $z_1, \ldots, z_m \in \{-1, 0, 1\}$ such that $z_1\mathbf{a_1} + \ldots + z_m\mathbf{a_m} = \mathbf{0} \in \mathbb{Z}_q^n$. This problem is intimately related with the Shortest Vector Problem.

Heuristics. This section relies on the following heuristics concerning the experimental behaviour of lattice reduction algorithms. They have been extensively verified and used in practice.

1. The fastest lattice reduction algorithms in practice are blockwise lattice algorithms, like BKZ-2.0[CN11], D-BKZ [MW16], or the slide reduction with large blocksize [GN08b, MW16]: their quality is measured by using the root-Hermite factor δ_0 defined in previous section.

2. Different approaches to evaluate the running time of BKZ-2.0 have been proposed in the literature [ACD+17, CN11, APS15, ADPS16, ACF+15]. In this section, we use the following estimation in seconds:

$$\log_2(t_{\mathrm{BKZ}})(\delta_0) = \frac{0.009}{\log_2(\delta_0)^2} - 27$$

   according to the extrapolation by Albrecht et al [ACF+15] of Liu-Nguyen datasets [LN13].

3. The coordinates of vectors produced by lattice reduction algorithms are balanced. Namely, if the algorithm produces vectors of norm $\|v\|_2$, each coefficient has a marginal Gaussian distribution of standard deviation $\|v\|_2/\sqrt{n}$. Provided that the geometry of the lattice is not too skewed in particular directions, this fact can sometimes be proved, especially if the reduction algorithm samples vectors with Gaussian distribution over the input lattice.

4. For mid-range dimensions and polynomially small noise, the SIS-distinguisher plus lattice reduction algorithms combined with the search-to-decision is the

best attack against LWE (but this point is less clear, according to the analysis of [ACF+15], at least, this attack model tends to over-estimate the power of the attacker, so it should produce more conservative parameters).

5. Except for small polynomial speedups in the dimension, we do not know better algorithms to find short vectors in random anti-circulant lattices than generic algorithms. This folklore assumption seems still up to date at the time of writing. The most recent asymptotic attacks [CDW17] against ideal lattices (that appear in our construction) do not reach polynomial noise rates, and they are not practical.

The SIS-based distinguisher attack against the LWE problem consists in finding a small integer combination that cancels the left hand side (i.e. the mask $\mathfrak{a}$) of homogeneous LWE samples. If applying the same combination to the right hand side (i.e. $\mathfrak{b}$) of the samples does not make it small, we can deduce that our inputs are not LWE samples, but rather uniformly random samples. Such SIS-distinguisher has in general a small advantage ε. To recover the full key, we use the well known decision-to-search reduction, which is particularly tight for TLWE: guess that the first key bit is zero, randomize the first coordinates of each sample, and use the distinguisher about $1/\varepsilon^2$ times to amplify its advantage to $\Theta(1)$, and to confirm whether the result are still TLWE samples, i.e. if our guess of the first key bit is correct. Once a key bit is found, getting the other bits involves solving lower-dimensional TLWE problems, that are significantly easier. Therefore we consider that the complexity of the attack is the time needed to find the first key bit. We also extend the analysis of [APS15] to handle the continuous torus.

Security Estimation. Let $(\mathfrak{a}_\mathbf{1}, \mathfrak{b}_1), \ldots, (\mathfrak{a}_\mathbf{m}, \mathfrak{b}_m)$ be either m TLWE samples of error standard deviation α or m uniformly random samples in $\mathbb{T}^{n+1}$. We need to find a small tuple $(v_1, \ldots, v_m) \in \mathbb{Z}^m$ such that the combination of samples $\sum v_i \mathfrak{a}_\mathbf{i}$ is small (SIS-distinguisher). Most previous models working on a discrete group would require that this term is exactly zero. By allowing approximations, we may find valid solutions in smaller dimension m than the usual bound $n \log n$. In particular, even $m < n$ would make sense.
Now, consider the $(m+n)$-dimensional lattice L, generated by the rows of the following basis $B \in \mathcal{M}_{n+m,n+m}(\mathbb{R})$:

$$B = \left(\begin{array}{ccc|ccc} 1 & & 0 & & & \\ & \ddots & & & 0 & \\ 0 & & 1 & & & \\ \hline a_{1,1} & \cdots & a_{1,n} & 1 & & 0 \\ \vdots & \ddots & \vdots & & \ddots & \\ a_{m,1} & \cdots & a_{m,n} & 0 & & 1 \end{array}\right).$$

Our goal is to find a short vector $\mathbf{w} = (\mathbf{x}|\mathbf{v}) = (x_1, \ldots, x_n, v_1, \ldots, v_m)$ in the lattice $L(B)$, whose first n coordinates $(x_1, \ldots, x_n) = \sum_{i=1}^{m} v_i \mathbf{a_i} \mod 1$ are shorter than the second part $(v_1, \ldots, v_m)$.
To do that, we choose a real parameter $q > 1$ (that will be optimized later), and apply the unitary transformation f_q to the lattice: f_q multiplies the first n coordinates of each line by q and the last m coordinates by $1/q^{n/m}$. Although this new basis now looks like a classical LWE matrix, the variable q is a real parameter, rather than an integer. It then suffices to find a regular short vector with balanced coordinates in the transformed lattice, defined by this basis:

$$f_q(B) = \left(\begin{array}{ccc|ccc} q & & 0 & & & \\ & \ddots & & & 0 & \\ 0 & & q & & & \\ \hline qa_{1,1} & \cdots & qa_{1,n} & \frac{1}{q^{n/m}} & & 0 \\ \vdots & \ddots & \vdots & & \ddots & \\ qa_{m,1} & \cdots & qa_{m,n} & 0 & & \frac{1}{q^{n/m}} \end{array}\right), \text{ with } q \in \mathbb{R} > 1.$$

To that end, we apply the fastest lattice basis reduction algorithm (BKZ-2.0 or slide reduction [GN08a]) directly to $f_q(B)$, which outputs a vector $f_q(\mathbf{w})$ of standard deviation $\frac{\delta_0^{n+m}}{\sqrt{n+m}}$.

Once we obtain such vector $\mathbf{w}$, all we need is to analyze the term

$$\sum_{i=1}^{m} v_i \mathfrak{b}_i = \sum_{i=1}^{m} v_i(\mathfrak{a}_\mathbf{i}\mathfrak{K} + e_i) = \mathfrak{K} \cdot \sum_{i=1}^{m} v_i \mathfrak{a}_\mathbf{i} + \sum_{i=1}^{m} v_i e_i = \mathfrak{K} \cdot \mathbf{x} + \mathbf{v} \cdot \mathbf{e},$$

which has Gaussian distribution of variance

$$\begin{aligned} \sigma^2 &= nS^2 \cdot \frac{\delta_0^{2(m+n)}}{q^2(m+n)} + \frac{q^{2\frac{n}{m}}\delta_0^{2(m+n)}}{m+n} \cdot \alpha^2 m \\ &= \delta_0^{2(m+n)} \left(\frac{S^2}{q^2} \cdot \frac{n}{m+n} + q^{2\frac{n}{m}} \alpha^2 \frac{m}{m+n} \right). \end{aligned} \tag{6.1}$$

Here $S = \frac{\|\mathbf{s}\|}{\sqrt{n}} \approx \frac{1}{\sqrt{2}}$. This distribution may be distinguished from the uniform distribution with advantage ε when σ^2 is equal to the smoothing variance $\frac{1}{2\pi}\eta_\varepsilon^2(\mathbb{Z})$. To summarize, the security parameter of LWE is bounded by the solution of the following system of equations

$$\lambda(n, \alpha) = \log_2(t_{\text{attack}}) = \min_{0<\varepsilon<1} \log_2 \left(\frac{1}{\varepsilon^2} t_{\text{BKZ}}(n, \alpha, \varepsilon) \right) \tag{6.2}$$

$$\log_2(t_{\text{BKZ}})(n, \alpha, \varepsilon) = \frac{0.009}{\log_2(\delta_0)^2} - 27 \tag{6.3}$$

$$\ln(\delta_0)(n,\alpha,\varepsilon) = \max_{\substack{m>1\\q>1}} \frac{1}{2(m+n)}\left(\ln\left(\frac{1}{2\pi}\eta_\varepsilon^2(\mathbb{Z})\right) - \ln\left(\frac{S^2}{q^2}\frac{n}{m+n} + q^{\frac{2n}{m}}\alpha^2\frac{m}{m+n}\right)\right) \tag{6.4}$$

$$\frac{1}{2\pi}\eta_\varepsilon^2(\mathbb{Z}) \approx \frac{1}{2\pi^2}\ln\left(\frac{2}{\varepsilon}\right). \tag{6.5}$$

Here, Equation (6.2) means that we need to run the distinguisher $\frac{1}{\varepsilon^2}$ times (by Chernoff's bound), and we need to optimize the advantage ε accordingly. [3] Equation (6.3) is the heuristic prediction of the running time of BKZ-2.0 reduction. Equation (6.4) follows from Equation (6.1): q and m need to be chosen in order to maximize the targeted approximation factor of the lattice reduction step.
Differentiating Equation (6.4) in q, we find that its maximal value is

$$q_{\text{best}} = \left(\frac{S^2}{\alpha^2}\right)^{\frac{m}{2(m+n)}}.$$

Replacing this value and setting $t = \frac{n}{m+n}$, Equation (6.4) becomes:

$$\ln(\delta_0)(n,\alpha,\varepsilon) = \max_{t>0}\frac{1}{2n}\left(t^2\ell_2 + t(1-t)\ell_1\right) \text{ where } \begin{cases}\ell_1 = \ln\left(\frac{\eta_\varepsilon^2(\mathbb{Z})}{2\pi\alpha^2}\right)\\ \ell_2 = \ln\left(\frac{\eta_\varepsilon^2(\mathbb{Z})}{2\pi S^2}\right).\end{cases}$$

Finally, by differentiating this new expression in t, the maximum of δ_0 is reached for $t_{\text{best}} = \frac{\ell_1}{2(\ell_1-\ell_2)}$, because $\ell_1 > \ell_2$, which gives the best choices of m and q and δ_0. Finally, we optimize ε numerically in Equation (6.5).

All previous results are summarized in Figure 6.1, which displays the security parameter λ as a function of n and $\log_2(\alpha)$. The figure is useful for the practical choice of the parameters, in particular from an user point of view. The idea is that when it comes to implementation, we know what is the security level we want to achieve, i.e. λ, and we know the limits of our implementation tools (memory, computational power and native operations supported). This can be translated in the amount of noise we can deal with and quantified by α. With the curve in Figure 6.1, the knowledge of λ and α let us easily chose the value of n.

6.2 TFHE: Fast Fully Homomorphic Encryption over the Torus

In order to have a proof of concept for the techniques described in the main chapters of this manuscript, we decided to implement the construction in 2016, at the same

[3] Amplifying a distinguishing advantage from ε to $\Omega(1)$ requires at least $O(1/\varepsilon)$ and at most $O(1/\varepsilon^2)$ trials, depending on the shape of the symmetric difference between the two distributions. Here, the difference between a modular Gaussian with large parameter and the uniform distribution is uniformly small, so we have to apply the upper-bound.

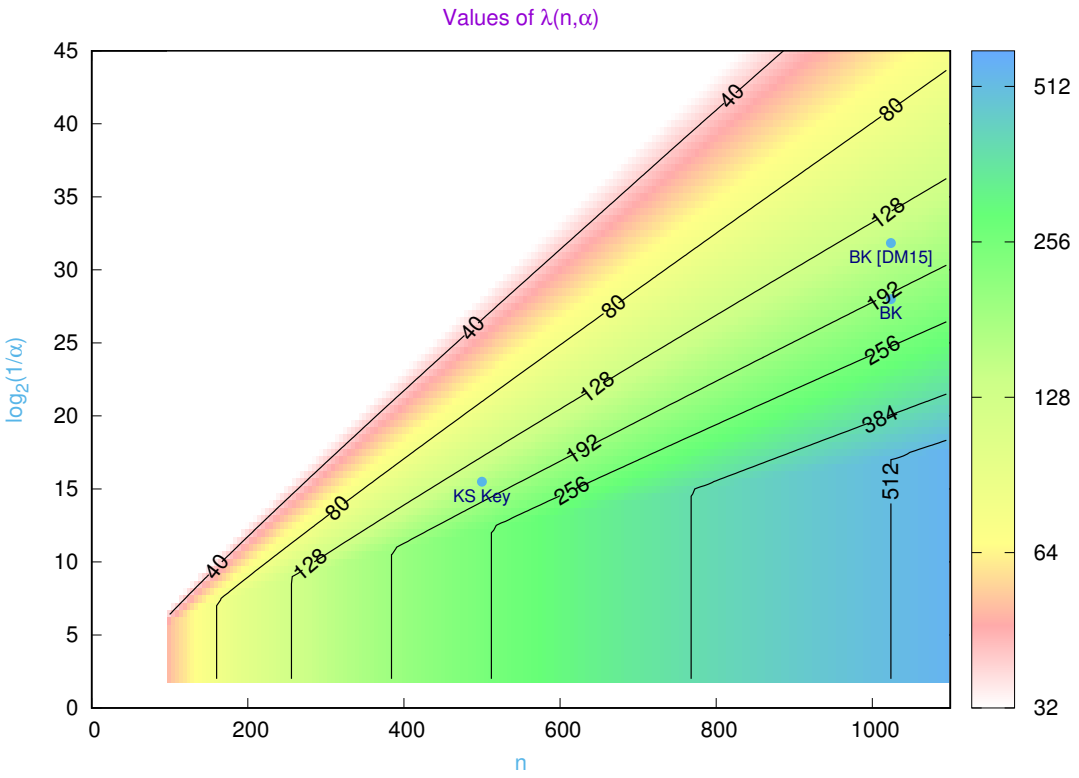

Figure 6.1: Security parameter λ as a function of n, entropy of the secret, and α, error standard deviation, for LWE samples - *This curve shows the security parameter levels λ (black levels) as a function of $n = kN$ (along the x-axis) and $\log_2(1/\alpha)$ (along the y-axis) for* TLWE *(also holds for binary-*LWE*), considering both the attack of this section and the collision attack in time $2^{n/2}$.*

time as our work on the gate bootstrapping [CGGI16a].
The implementation is a C/C++ library: the first official release has been done on may 2017 on the TFHE website [CGGI16d]. Two GitHub repositories exist for TFHE:

1. The library repository [CGGI16c]: containing the code of the library.

2. The experimental repository [CGGI17b]: containing the experimental features not already included in an official release.

The TFHE library. TFHE is an open source library distributed under the terms of the Apache 2.0 license, working on Linux and MacOS platforms (not yet on Windows). The library contains the implementation of the gate bootstrapping for the evaluation of 10 different binary gates and for the native `MUX` gate. The

gate bootstrapping corresponds to the one described in [CGGI16a] with the improvements on the blind rotation proposed in [CGGI17a].

Our primary goal was to simplify as much as possible the use of the library. In the website, the installation and the usage of TFHE are detailed, and a short tutorial on a simple use case is given.
The functions implemented for the gate bootstrapping API can be divided in five main families:

- **Generation of parameters -** including all the functions needed to generate, delete and import/export in a file the default parameters. The only value needed to generate the parameters is the security level: for now a single security level is supported[4].
- **Generation of the keys -** including all the functions needed to generate, delete and import/export in a file the secret keys, used to encrypt/decrypt the TLWE samples, and the bootstrapping keys, used during the evaluation.
- **Manipulation of ciphertexts -** including all the functions needed to initialize, delete and import/export in a file a ciphertext or an array of TLWE ciphertexts.
- **Encryption and decryption -** including the encryption and decryption functions, implemented for the symmetric version of TFHE.
- **Homomorphic gates -** including all the bootstrapped homomorphic gates implemented: a constant gate, producing a trivial sample of 0 or 1, the unary `NOT` gate and the copy gate, the binary gates `NAND`, `OR` (and two composition with the `NOT` gate), `AND` (and two composition with the `NOT` gate), `XOR`, `XNOR`, `NOR`, and the ternary `MUX` native gate.

The execution of unary and trivial gates is immediate. All binary gates are executed in $13ms$ and the ternary `MUX` takes $26ms$, on a 64-bit (single core) Intel Core i7-4910MQ at 2.90GHz laptop.

Gate bootstrapping mode

Time per gate bootstrapping (1 bit)	$t_{\texttt{GB}} = 13ms$
Time per any binary bootstrapped gate (`NAND`, `OR`, `AND`, etc.)	$t_{\texttt{GB}} = 13ms$
Time per bootstrapped `MUX`	$2t_{\texttt{GB}} = 26ms$

In many previous implementations, as multiplication was more expensive than addition, it was almost required to use a dedicated compiler reducing the function

[4] In the library webpage, the security leveled announced is about 110-bits, but it is actually $\sim$ 159-bits. More details are given in next section.

to its "more FHE friendly form" (see as instance [CDS15]). Instead, as in the gate bootstrapping mode of TFHE there is no difference between the cost of binary gates, the construction of homomorphic circuits is easier: it suffices to know the naive circuit of the function to be evaluated. Every gate in clear is then translated in its homomorphic bootstrapped version. A circuit composed of a few gates, or even naive circuits with large depth are evaluated efficiently with TFHE.

From a theoretical point of view, our scale invariant scheme TLWE is defined over the real torus $\mathbb{T}$, where all the operations are modulo 1. In practice, since we can work with approximations, we chose to rescale the elements over $\mathbb{T}$ by a factor 2^{32}, and to map them to 32-bit integers. Thus, we take advantage of the native mod 2^{32} operations, including for the external multiplication with integers.
The internal functions, on which the API functions (just described) are based, work with 32-bit and also with 64-bit integers (taking advantage of the same native operations), and perform polynomial multiplications by using the Fast Fourier Transform (FFT). Except for some FFT operations, using 32 and 64-bit integers seems more stable and efficient than working with floating point numbers and reducing modulo 1 regularly.

FFT. The library supports three FFT processors and at least one of them is needed to run the computations: the users can chose which one they want to use and the resulting performances variate depending on the choice.

1. The Project Nayuki FFT library [Nay], usable in the portable C or with AVX assembly instructions. We added to their code the reverse FFT functions. The library is published under the MIT license.

2. Our dedicated *Spqlios* FFT processor, implemented with AVX and FMA assembly instructions. The implementation is written following the Nayuki Project footstep, and is dedicated to the FFT operations over the ring $\mathbb{R}[X]/(X^N+1)$, with N a power of 2.

3. The FFTW3 library (Fastest Fourier Transform in the West) [FJ05], which is between 2 and 3 times faster than the Nayuki FFT. The library is published under the GPL license and it is also used in the FHEW implementation by [DM15].

AVX and FMA are SIMD floating point instructions, available on recent processors. In our TFHE implementation, polynomials mod X^N+1 are either represented as the classical list of the N coefficients, or using the Lagrange half-complex representation, which consists in the complex (2 times 64-bits) evaluations of the polynomial over the roots of unity $\exp(i(2j+1)\pi/N)$ for $j \in [\![0, \frac{N}{2}[\![$. Indeed, the $\frac{N}{2}$ other evaluations are the conjugates of the first ones, and do not need to be stored.

The conversion between both representations is done using our dedicated Spqlios implementation of the FFT. Note that the direct FFT transform is $\sqrt{2N}$ lipschitzian, so the Lagrange half-complex representation tolerates approximations, and 53-bits of precision is indeed more than enough, provided that the real representative remains small. However, the modulo 1 reduction, as well as the gadget decomposition in base H, are not compatible with the Lagrange representation: we therefore need to regularly transform the polynomials to and from their classical representation.

Profiling the execution shows that the FFTs and complex multiplications are taking 66% of the total time. The remaining time is mostly taken by the key-switching operation and the decomposition in base H.

Experimental TFHE. The leveled version of the TFHE scheme, with all the improvements presented in Chapter 4 and the circuit bootstrapping, is not already included in the official release of TFHE.
The circuit bootstrapping is instead implemented in the experimental repository of the project [CGGI17b]. In order to execute the circuit bootstrapping, as we work with different noise levels, we use both 32-bit and 64-bit integer operations.
The execution timing of a circuit bootstrapping is $137ms$, on a 64-bit (single core) Intel Core i7-4910MQ at 2.90GHz laptop.

Future releases. We are planning on release new versions of the TFHE library, including more features than before.
The first features we are planning to add are the *asymmetric encryption* version of the scheme, which is practical for real world applications, the *compatibility with Windows* operative systems, and the *multi-threading*, as the library works only with a single core. In particular, this last improvement could considerably accelerate the execution timings.
Later, we would also include in the library the *leveled mode* of the scheme, by implementing all the improvements presented in Chapter 4, the *multi-bit arithmetic* and the *circuit bootstrapping*, already implemented in the experimental branch.
Furthermore, it could be interesting to allow *multiple choices for the security levels* and for the corresponding parameters, as for now the implementation provide a single parameter set.

6.3 Concrete Parameters

One of the most complicated questions in homomorphic encryption is the choice of parameters for the scheme. This choice have to take into account many factors: the security level of the construction, the memory consumption, the execution timings, etc.

In Section 6.1 we expressed the security parameter with respect to the entropy of the secret key and the noise level of the ciphertexts. The curve we obtained allows us to choose properly the basic parameters.
But the noise growth in the leveled evaluation and in bootstrapping presents many additional parameters that have to be chosen in the more convenient way. The choice can be done by using the noise growth formulas and by balancing them until we reach a satisfying result.

In this section we give the concrete parameters we used in both gate and circuit bootstrapping implementations. We often compare our results on gate bootstrapping with the results obtained in [DM15], and recently improved in the implementation FHEW [DM17]. We also present some practical results we obtained by testing the library and the experimental features for several use cases.

6.3.1 Gate bootstrapping Parameters.

In the TFHE library, the gate bootstrapping uses the following parameters.

- TLWE samples use $n = 500$ and standard deviation $\sigma = 2^{-7}$, so their amplitude is $< \frac{1}{16}$. A TLWE sample has $32 \cdot (n+1)$ bits ≈ 2 KBytes.
- TRLWE samples use $N = 1024$ and $k = 1$. This corresponds to $(k+1) \cdot N \cdot 32$ bits ≈ 8 KBytes.
- TRGSW samples use $\ell = 2$ and $B_g = 1024$. This defines the gadget H and its decomposition $\mathrm{Dec}_{H,\beta,\epsilon}$ where $\beta = 512$ and $\epsilon = 2^{-21}$. A TRGSW sample is composed by $(k+1) \cdot \ell$ TRLWE samples ≈ 32 KBytes.
- The bootstrapping key has n TRGSW samples ≈ 15.6 MBytes. Its noise standard deviation is $\alpha = 3.73 \cdot 10^{-9}$ ($\sim 2^{-28}$). Section 6.1 results predict about 198 bits of security for our bootstrapping key. In [DM15], the size of the bootstrapping key is of 1 GByte, and its standard deviation is $\approx 2.59 \cdot 10^{-10}$ ($\sim 2^{-32}$).
- We chose $t = 16$ bits for the key switching: the decomposition has precision 2^{-17}, and the key-switching key is composed by $k \cdot N \cdot t$ TLWE samples ≈ 32 MBytes. We set the noise standard deviation to $\gamma = 2.16 \cdot 10^{-5}$ ($\sim 2^{-15.5}$), which is estimated to about 159-bits of security following the results from Section 6.1.

Correctness. The final error variance after bootstrapping is $1.63 \cdot 10^{-5}$, by Theorem 5.1.2. It corresponds to a standard deviation of $\sigma = 4.04 \cdot 10^{-3}$. The noise amplitude after our bootstrapping is $< \frac{1}{16}$ with very high probability $\mathsf{erf}(1/16\sqrt{2\pi}\sigma) \geq 1 - 2^{-58}$, which is better than $\geq 1 - 2^{-31}$ in [DM15].

Note that the size of the key-switching key can be reduced by a factor $n + 1 = 501$ if all the masks are the output of a pseudo random function: we may for instance just give the seed. The same technique can be applied to the bootstrapping key, on which the size is only reduced by a factor $k + 1 = 2$.
As we announced in previous section, we measured a running time of $13ms$ per binary gate bootstrapping, $26ms$ per bootstrapped `MUX` and $34\mu s$ per external product, all using the our Spqlios FFT for multiplications. The timings are obtained by running the library on a 64-bit (single core) Intel Core i7-4910MQ at 2.90GHz laptop. This seems to correspond to the machine used in [DM15].

	n	α	λ	$\varepsilon_{\text{best}}$	m_{best}	q_{best}	δ_{best}
KS key	500	$2.43 \cdot 10^{-5}$	159	2^{-14}	497	178.7	1.0052
BK	1024	$3.73 \cdot 10^{-9}$	198	2^{-10}	1037	14696.	1.0046
BK [DM15]	1024	$2.59 \cdot 10^{-10}$	149	2^{-7}	1051	60351	1.0064

Figure 6.2: Parameters and security of the Gate bootstrapping - *This table precises the parameters for the key-switching key and the bootstrapping key for our implementation and for the one in [DM15].*

Table 6.2 shows that the strength of the lattice reduction is compatible with the values announced in [DM15]. Our model predicts that the lattice reduction phase is harder ($\delta = 1.0052$ in our analysis and $\delta = 1.0064$ in [DM15]), but the value of ε is bigger in our case. Overall, the security of their parameters-set is evaluated by our model to 149-bits of security, which is larger than the $\geq$ 100-bits of security announced in [DM15]. The main reason is that we take into account the number of times we need to run the SIS-distinguisher to obtain a non negligible advantage. Since our scheme has a smaller noise propagation overhead, we were able to raise the input noise levels in order to strengthen the system, so with the parameters we chose in our implementation, our model predicts 198-bits of security for the bootstrapping key and 163-bits for the key-switching key, which becomes the bottleneck.

Remark 6.3.1 (FHEW)**.** *The scheme proposed in [DM15] and implemented in [DM17] has been recently improved by the authors. The original timing of* 0.69 *seconds, corresponding to the evaluation of a bootstrapped* `NAND` *gate (on a single* 64*-bits Intel core at 3GHz), has been improved in the last version of their library and it is now* 6× *faster than before. The trick they use to improve the bootstrapping is somehow equivalent to the external product we presented in Chapter 3. Additionally, they add the implementation of more homomorphic gates:* `AND`*,* `OR`*,* `NOR` *and the unary* `NOT` *gate.*

Experimental validation of the independence assumption. In order to verify the correctness of our theoretical results and of our implementation when evaluating huge circuits, we decided to run a large test. We generated several random Boolean circuits of depth larger than 10000, composed by a million of bootstrapped homomorphic gates. We executed them and measured the actual noise of the ciphertexts. The results we obtained confirmed the accuracy of the average case theorems of Section 5.1, and that the noise distribution after bootstrapping is Gaussian. We summarize these results in Figure 6.3: the light blue histogram is the measured error distribution after every bootstrapped gate, the purple plain line represents the Gaussian distribution whose variance is predicted by Theorem 5.1.2 using the parameters presented in this section, and the red dashed line represents the critical Gaussian distribution for an amplitude of $\frac{1}{16}$. This experimentally validates our independence heuristic Assumption 3.2.1, even when ciphertexts are re-used in a very large depth homomorphic circuit.

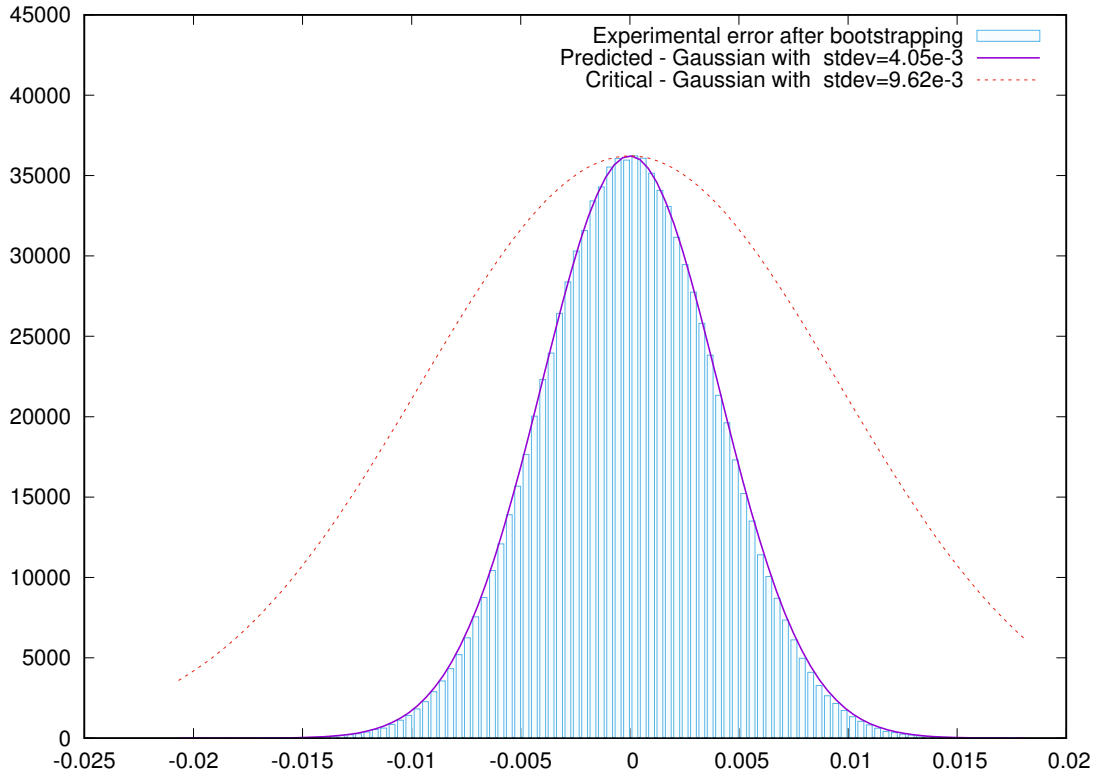

Figure 6.3: Accuracy of the average case theorems, and experimental validation of the independence assumption.

6.3.2 Circuit Bootstrapping

The circuit bootstrapping, described in Section 5.2 is not yet included in a official release of the TFHE library, but we implemented it in the experimental repository [CGGI17b] as a proof of concept. The tests show that we can evaluate a circuit bootstrapping in 0.137 seconds, on a 64-bit (single core) Intel Core i7-4910MQ at 2.90GHz laptop.

One of the main constraints to obtain this performance is to ensure that all the computations are feasible and correct under 53 bits of floating-point precision, in order to use the fast FFT. This requires to refine the parameters of the scheme. We verified the accuracy of the FFT with a slower but exact Karatsuba implementation of the polynomial product.
In our three levels, we used the following TRLWE and TRGSW parameter sets, which have at least 152-bits of security, according to the security analysis from Section 6.1.

Level	**Minimal noise standard dev.**	n	B_g	ℓ	λ
0	$\underline{\alpha} = 6.10 \cdot 10^{-5}$ $(\sim 2^{-14})$	$\underline{n} = 500$	N.A.	N.A.	194
1	$\alpha = 3.29 \cdot 10^{-10}$ $(\sim 2^{-31.5})$	$n = 1024$	$B_g = 2^8$	$\ell = 2$	152
2	$\bar{\alpha} = 1.42 \cdot 10^{-14}$ $(\sim 2^{-46})$	$\bar{n} = 2048$	$\bar{B_g} = 2^9$	$\bar{\ell} = 4$	289

Since we assume circular security, we use only one key per level, and the following key-switching parameters (in the leveled setting, the reader is free to increase the number of keys if he does not wish to assume circularity).

Level	t	**KS noise standard dev.** γ	**Usage**
$1 \to 0$	$\underline{t} = 12$	$\underline{\gamma} = 6.10 \cdot 10^{-5}$ $(\sim 2^{-14})$	Circuit Bootstap, Pre-KS
$2 \to 1$	$\bar{t} = 30$	$\bar{\gamma} = 3.29 \cdot 10^{-10}$ $(\sim 2^{-31.5})$	Circuit Bootstap, Step 4 in Alg. 12
$1 \to 1$	$t = 24$	$\gamma = 2.38 \cdot 10^{-8}$ $(\sim 2^{-25.3})$	TBSR

Thus, we get these noise variances in input or in output.

Output TLWE	**Fresh TRGSW in LHE**	**TRGSW Output of CB**	**Bootst. key**
$\vartheta \leq 2^{-10}$	$\vartheta = 2^{-60}$	$\vartheta \leq 2^{-48.2}$	$\vartheta_{BK} = 2^{-92}$

And finally, this table summarizes the timings, noises overhead, and maximal depth of all our primitives.

	CPU Time	**Var Noise add**	**Max depth**
Circuit bootstrap	$t_{\texttt{CB}} = 137ms$	N.A.	N.A.
Fresh `CMux`	$t_{\texttt{XP}} = 34\mu s$	$2^{-23.99}$	16384
CB `CMux`	$t_{\texttt{XP}} = 34\mu s$	$2^{-20.17}$	1017
PublicKS$_{TBSR}$	$t_{\texttt{KS}} = 180ms$	$2^{-23.42}$	16384

Comparison with previous bootstrappings for TGSW. The circuit bootstrapping we just described evaluates a quasi-linear number of level-2 external products, in the bootstrapping part, and a quasi-linear number of level-1 products, in the private key-switching part. With the parameters proposed, it runs in 0.137 seconds for $\sim$ 152-bits of security. Level 2 operations take about 70% of the running time, and the private key switching the remaining 30%.
Our circuit bootstrapping is not the first bootstrapping algorithm that outputs a TRGSW ciphertext. Many constructions have previously been proposed and achieve valid asymptotic complexities, but very few concrete parameters are proposed. Most of these constructions are recalled in the last section of [GINX14]. In all of them, the bootstrapped ciphertext is obtained as an arithmetic expression on TRGSW ciphertexts involving linear combinations and internal products. First, all the schemes based on scalar variants of GSW suffer from a slowdown of a factor at least quadratic in the security parameter, because the products of small matrices with polynomial coefficients (via FFT) are replaced with large dense matrix products. Thus, bootstrapping on GSW variants would require days of computations, instead of the 0.137 seconds we propose.
Now, assuming that all the bootstrapping uses (Ring) instantiations of GSW, the design in [BV14b] based on the expansion of the decryption circuit via Barrington's theorem, as well as the expression as a minimal deterministic automata of the same function in [GINX14] would require a quadratic number of internal level 2 TRGSW products, which is much slower than what we propose. Finally, the CRT variant in [AP14] and [GINX14] uses only a quasi-linear number of products, but since it uses composition between automata, these products need to run in level 3 instead of level 2, which induces a huge slowdown (a factor 240 in our benchmarks), because elements cannot be represented on 64-bits native numbers.

6.4 Time comparison between different techniques

With these parameters, we analyze the (single-core) execution timings for the evaluation of three use cases: the LUT, the MAX function and the multiplication. We give prediction timings[5] for these applications in both LHE and FHE mode, by analyzing many of the techniques proposed in the main chapters of this manuscript.

In the leveled mode all inputs are fresh ciphertexts, either TRLWE or TRGSW: we compare the previous versions of TFHE, without packing/batching, with the new optimization using horizontal/vertical packing, det-WFA and TBSR techniques.
In the FHE mode all inputs and outputs are TLWE samples on the $\{0, \frac{1}{2}\}$ message space with noise amplitude $\frac{1}{4}$. Each operation starts by bootstrapping its inputs.

[5]The predictions are based on the experimental timings we obtained for the external product, the key switching and both the gate and circuit bootstrapping.

We compare the gate bootstrapping strategy with the mixed versions using leveled optimized evaluations with circuit bootstrapping.
Our goal is to identify which method is better for each of the 6 cases.

In this section $t_{\texttt{XP}} = 34\mu s$ indicates the execution timing per external product, $t_{\texttt{GB}} = 13ms$ indicates the execution timing per gate bootstrapping, $t_{\texttt{CB}} = 137ms$ indicates the execution timing per circuit bootstrapping and $t_{\texttt{KS}} = 180ms$ indicates the execution timing per key switching.

LUT In Figure 6.4 we represent the predicted execution timings for the evaluation of a LUT with $s = 8$-bits of output, in leveled and fully homomorphic modes respectively.
In the graph, the x coordinate represents the number of input bits d, while the y coordinate represents the execution timings in seconds, in logarithmic scale.

In the *leveled* evaluation, we analyze 3 different modes:

- No packing: running in $s(2^d - 1)t_{\texttt{XP}}$.
- Horizontal Packing: running in $(2^d - 1)t_{\texttt{XP}}$.
- Vertical Packing: running in $s(2^d/N - 1 + \log N)t_{\texttt{XP}}$.

In the *fully homomorphic* evaluation, we analyze 3 different modes:

- Gate bootstrapping: running in $(d + 2s(2^d - 1))t_{\texttt{GB}}$.
- Circuit bootstrapping with horizontal packing: running in $dt_{\texttt{CB}} + (2^d - 1)t_{\texttt{XP}}$.
- Circuit bootstrapping with vertical packing: running in $dt_{\texttt{CB}} + s(2^d/N - 1 + \log N)t_{\texttt{XP}}$.

The graph representing the execution timings on leveled evaluation reveals that the horizontal packing is the best technique up to $6 - 7$ bits of input, then vertical packing becomes optimal. In fully homomorphic mode, the circuit bootstrapping with vertical packing is the best technique on the long run.
Observe that compared to the gate bootstrapping, we obtain a huge speed-up for the homomorphic evaluation of arbitrary function FHE mode. In particular, we can evaluate a 8 bits to 1 bit LUT and bootstrap the output in just $137ms$, or evaluate an arbitrary 8 bits to 8 bits function in $1.096s$, and an arbitrary 16 bits to 8 bits function in $2.192s$.

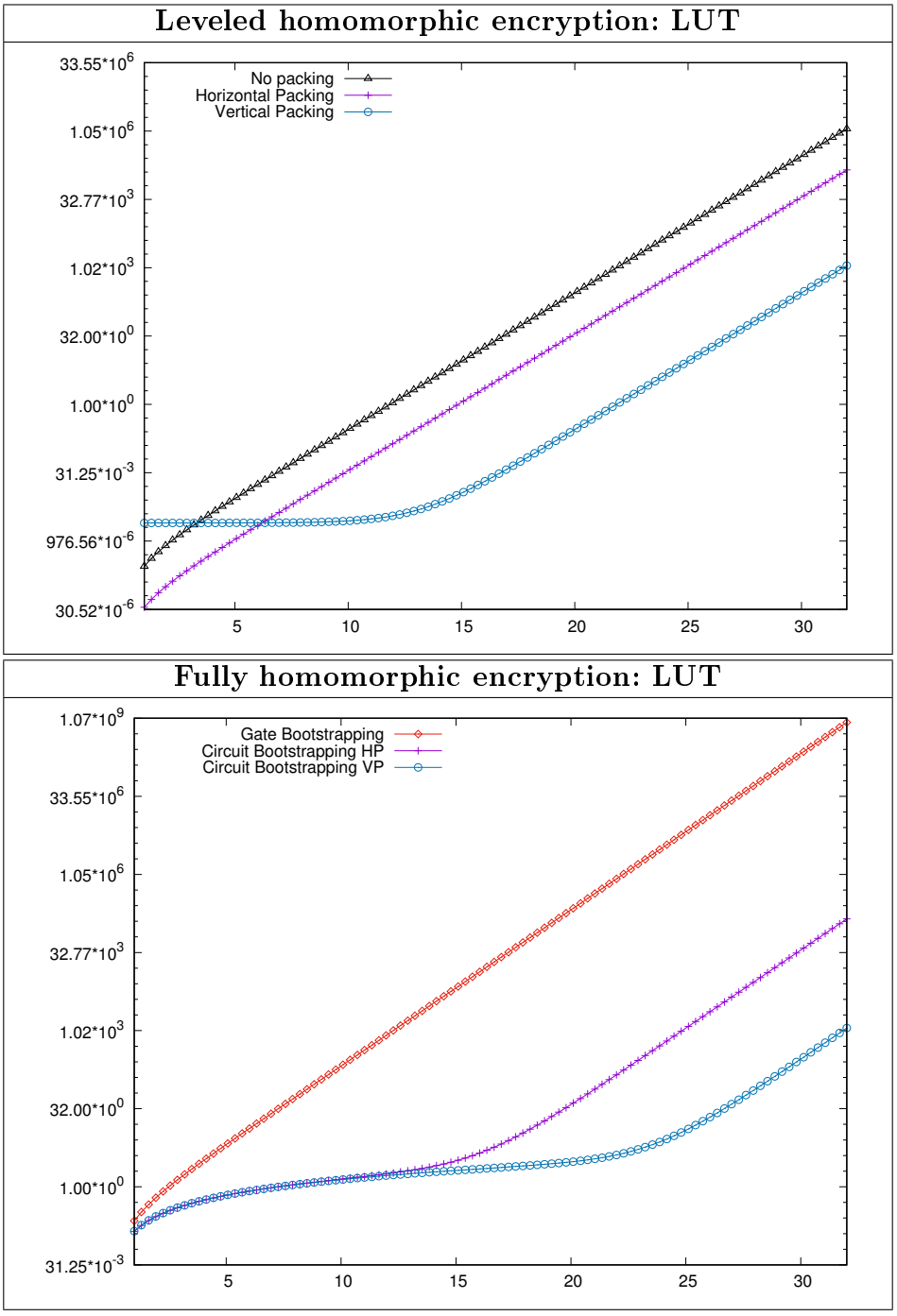

Figure 6.4: Leveled and fully homomorphic evaluation of LUT (d-bits input and $s = 8$-bits output) - *The y coordinate represents the running time in seconds (in logscale), the x coordinate represents the number of bits in the input.*

MAX function In Figure 6.5 we represent the predicted execution timings for the evaluation of the MAX function, computing the maximal value between two d-bit integers, in leveled and fully homomorphic modes respectively.
In the graph, the x coordinate represents the number of input bits d, while the y coordinate represents the execution timings in seconds. Both coordinates are represented in logarithmic scale.

In the *leveled* evaluation, we analyze 2 different modes:

- No packing: running in $3(d(d+1)/2)t_{\texttt{XP}}$.
- Evaluation via det-WFA: running in $5dt_{\texttt{XP}}$.

In the *fully homomorphic* evaluation, we analyze 2 different modes:

- Gate bootstrapping: running in $(2+5)dt_{\texttt{GB}}$.
- Circuit bootstrapping with det-WFA: running in $2dt_{\texttt{CB}} + 5dt_{\texttt{XP}}$.

The graphs reveal that the best technique for leveled evaluation is the use of det-WFA for inputs of more than 3 bits: for less than 3 bits of input, the evaluation with no optimization is optimal. For the fully homomorphic evaluation, the gate bootstrapping mode seems the most interesting choice all the time.

Multiplication Figure 6.6 represents the predicted execution timings for the evaluation of the multiplication between two d-bit integers, in leveled and fully homomorphic modes respectively.
In the graph, the x coordinate represents the number of input bits d, while the y coordinate represents the execution timings in seconds. Both coordinates are represented in logarithmic scale.

In the *leveled* evaluation, we analyze 3 different modes:

- No packing: running in $(d^4 + o(d^4))t_{\texttt{XP}}$.
- Evaluation via det-WFA: running in $\Theta(d^3)t_{\texttt{XP}}$ (computed by optimization program).
- TBSR: running in $2d^2t_{\texttt{XP}} + (2d-2)t_{\texttt{KS}}$.

In the *fully homomorphic* evaluation, we analyze 3 different modes:

- Gate bootstrapping: running in $(6d^2 - 3d)t_{\texttt{GB}}$.
- Circuit bootstrapping with det-WFA: running in $2dt_{\texttt{CB}} + \Theta(d^3)t_{\texttt{XP}}$ (computed by optimization program).

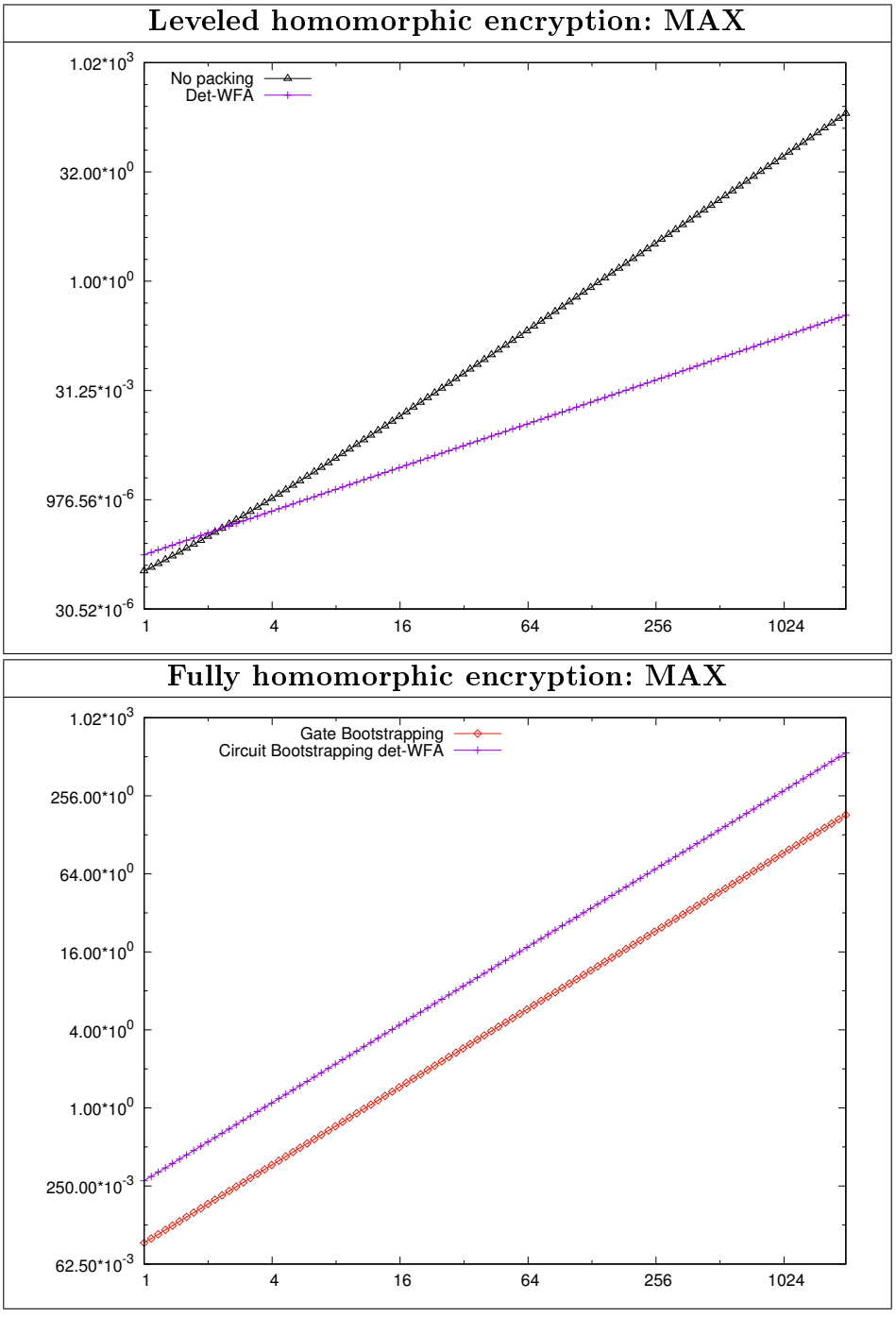

Figure 6.5: Leveled and fully homomorphic evaluation of the MAX function (2 inputs of d-bits each) - *The y coordinate represents the running time in seconds (in logscale), the x coordinate represents the number of bits in the input (in logscale).*

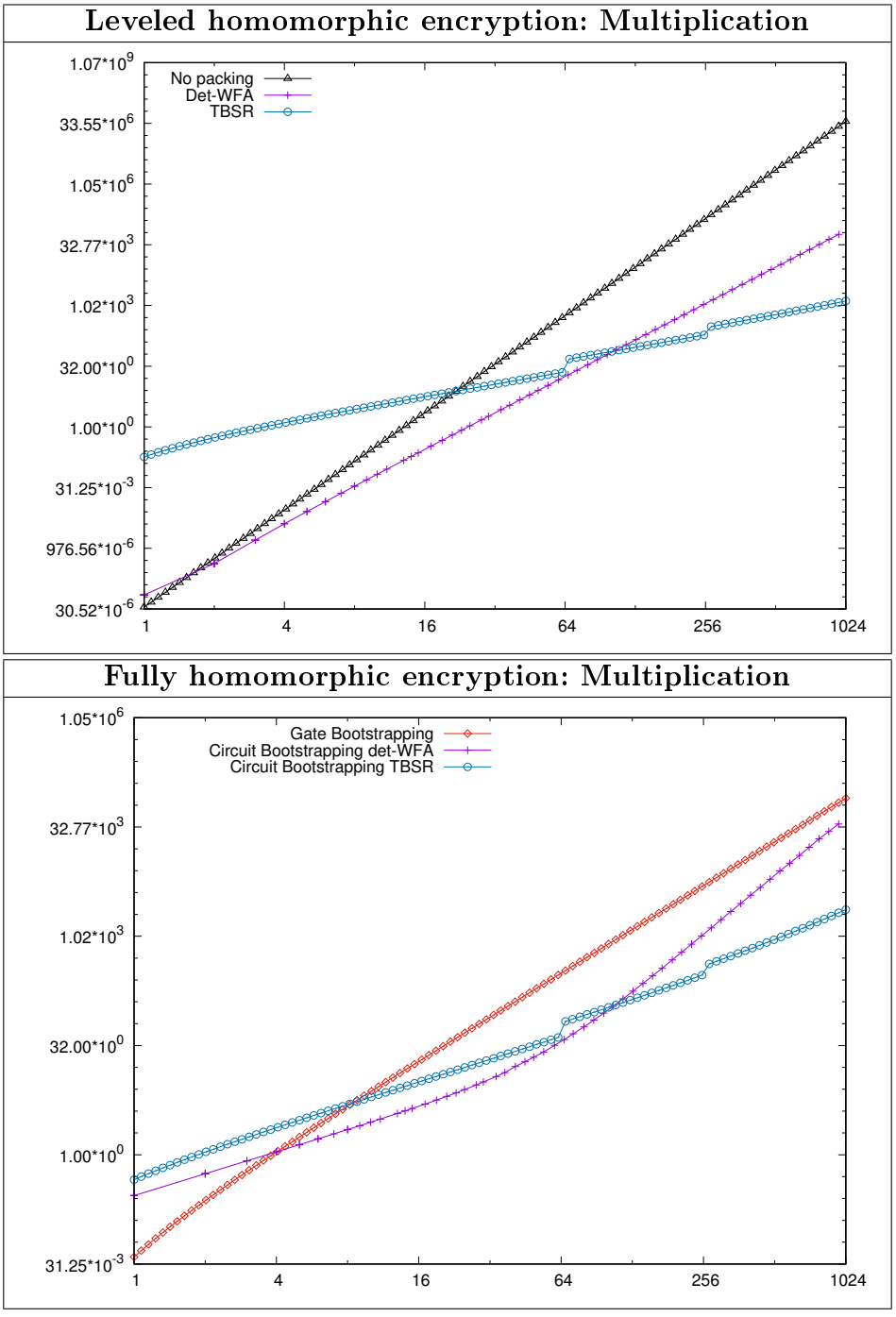

Figure 6.6: Leveled and fully homomorphic evaluation of the Multiplication (2 inputs of d-bits each) - *The y coordinate represents the running time in seconds (in logscale), the x coordinate represents the number of bits in the input (in logscale).*

- Circuit bootstrapping with TBSR: (computed by optimization program).

The graph representing the leveled evaluation of the multiplication revels that for less than 2-bits input the best evaluation is performed without any optimization. Starting from $2-3$ bits of input, the best technique is the evaluation via det-WFA, while after 128 input bits, the TBSR counter is optimal. The graph representing the fully homomorphic evaluation shows that up to 4-bit inputs the best technique is the gate bootstrapping. Then, circuit bootstrapping combined with det-WFA becomes optimal up to about 128 bit inputs, where again the circuit bootstrapping combined with the TBSR techniques seems to be the best choice.

6.4.1 Comparison between TFHE and the other schemes

In previous section we shown how the different techniques presented in the main chapters of this manuscript perform when applied on different use cases. The results show that no one between the techniques proposed is always the best. The performances do not only depend on the specific function to be evaluated, but also on the number of inputs and outputs, as shown in many of the graphs presented. The choice between leveled and fully homomorphic mode depends also on the application, on the dataset used (dynamic or not), on the security assumptions, etc. Many variables have to be taken into account.

Furthermore, TFHE is not the only library on homomorphic encryption proposed. We often compare our results with FHEW [DM17], which is a predecessor of TFHE, but a broader comparison should be done.
Between the open source implementations proposed, we have:

- HElib[HS17]: implements the optimized BGV [BGV12] scheme.
- SEAL [aMR17]: implements a variant of the FV [FV12] scheme.
- NFLlib [CEQ16]: dedicated to ideal lattice cryptography.
- cuHE [Dai16]: implements a variant of the LTV [LTV12] scheme.
- HEAAN [HEA17]: implements the scheme proposed in [CKKS17].
- $\Lambda \circ \lambda$ (Lol) [Lol17]: dedicated to ring-based lattice cryptography.
- PALISADE [PAL17]: dedicated to general lattice cryptography.
- And many other implementations.

In these implementations, different techniques and optimization are proposed: dedicated packing and batching techniques, bootstrapping, FFT and NTT[6],

[6]Number Theoretic Transform.

multi-threading, multi-key, approximate number computations, etc.

A natural question at this point could be: which one of these schemes (including TFHE) is the best?
Maybe the answer can be found by defining what is the meaning of "best". Again there is not an absolute best scheme or implementation for now. Every construction has its pros and cons, and specific functionalities that are well suited for specific applications.

This is still an open question for us, but we think that a good way to answer could be to fix a large number of use cases and try to implement each one of them with the different constructions proposed. The result would not be a single winner, but most probably a portfolio of good solutions.

Conclusion

In this thesis we studied in detail the TFHE scheme, from both a theoretical and a practical point of view. The reformalization of this GSW-based scheme helped us simplify the comprehension of the construction and to better understand all the links between different plaintext spaces.

As a result, both the leveled and bootstrapped constructions where improved: we proposed new packing techniques for the homomorphic evaluation of LUT, an evaluation via det-WFA, the TBSR homomorphic counter, and we improved the bootstrapping in the case when it is evaluated after every homomorphic gate (gate bootstrapping) or after the evaluation of a leveled circuit (circuit bootstrapping).
The techniques proposed can be used separately or mixed, depending on the application, as shown in the examples. Some of the techniques presented have been implemented in the TFHE library, available in open-source on GitHub.

We present three additional works in the appendices. The first one analyzes the security of general homomorphic encryption schemes in a cloud implementation scenario, by presenting safe error attacks and by suggesting some possible countermeasures. The second one presents the first post-quantum e-voting encryption scheme, as a real world application of FHE: the scheme is not already implemented but we predicted the approximated execution timings, showing that a practical set up could be possible. The third one concerns a different solution to perform secure private computations via MPC: o can be used to evaluate real-valued functions with high numerical precision, which is interesting for classification problems aiming to detect rare events.

We would like to conclude this manuscript by summarizing the open problems, that could be the starting point for future works.

Open problems

The first open problem we would like to solve is the implementation of the complete TFHE scheme: in the official release of the library, only the gate bootstrapping

mode is implemented and the circuit bootstrapping is still in the experimental repository. But all the other features described in this manuscript are not already implemented and tested.

TFHE is a valuable solution for FHE computations, but many other schemes exist. There is no one best scheme for now, every one of them proposes different functionalities, that are better suited to some applications rather than others. An example that is often cited is the homomorphic evaluation of the AES decryption, which has been studied for its interest in trans-ciphering. For this evaluation, the BGV-based schemes and the HElib library seem more efficient than other implementations, thanks to the optimal vectorized operations [GHS12]. But recently, it has been shown that stream-ciphers seems more adapted than block-ciphers for trans-ciphering [CCF+16]. In this case, GSW-based schemes and in particular TFHE, optimizing homomorphic gate operations, are more efficient.

A more detailed comparison between different schemes needs to be done on a larger portfolio of functions. Furthermore, it could be interesting to improve every one of these construction by adapting the optimization techniques proposed in other schemes, or to construct new efficient "bootstrappings" allowing to homomorphically switch from a scheme to another. This requires a deep study of all the constructions. Of course, a FHE scheme that does not need to be bootstrapped may be the solution to many of the efficiency problems of the schemes, since bootstrapping is still the more costly part of the homomorphic evaluations. But this is a major open problem that might stay open for a long time.

Once the implementation of TFHE will be more complete, it would be interesting to test other real world applications and to finally implement our e-voting scheme[7], described in Appendix B.

Finally, in preparation for a cloud use of homomorphic encryption constructions, the ability of verifying the correctness of computations in a efficient way seems an important task that needs to be studied more deeply. We discuss this point in Appendix A.
Also other secure solutions for private computations deserve attention. We started investigating one of them, multi-party computation, in our work described in Appendix C. This seems to be a good starting point to better understand the large scenario of solutions proposed in the literature.

[7] The scheme still needs an update of the security proofs and to include further security notions. Only in this way, it could be taken into consideration for future medium-scale elections.

Appendices

Appendix A

Cloud security of homomorphic encryption

In this chapter we analyze the security of homomorphic encryption schemes in general (not only TFHE) when they are used in a cloud context. We start by describing the attacks on generic HE schemes, then we analyze the attacks on bootstrappable schemes.

As we already said in Chapter 1, nowadays we store more and more data on the cloud: e-mails, photos, contacts, GPS information, etc. We upload all this information by using secure protocols and we store it in cloud servers, inside accounts protected by a password. But the cloud provider itself remains the legitimate recipient, so he keeps an unrestricted access to this valuable information. In this case, it is natural to consider an honest but curious model.

But if we start using homomorphic encryption to hide the data to the cloud itself, we restrict this unlimited access and all the advantages the cloud service providers gain from this. It is natural to imagine that some cloud providers will stop offering their services for free, and some others may even switch to more powerful attacks, that could include data tampering, in order to regain access to that information. In order to capture the second scenario, we have to consider the cloud operative systems untrusted and malicious.

Of course, as the services provided constitute a huge economical value, the cloud will do his best to ensure that his attacks remain undetected: indeed, visible attacks would break the client's trust, and would incite him to stop using the cloud services, which is obviously not the goal. In this case , our security model identifies the cloud as a discreet and cautious adversary.

In this chapter, we (improperly) use the term cloud to denote also any possible malicious actor that can access the data stored/processed, inject errors in order to perform an attack and observe all the communications between the cloud server and the clients.

Our goal is to prove that the security of an homomorphic scheme cannot be assessed at the primitive level. The broader construction must also be taken into account in the security analysis. The point we want to stress is that, even if the primitives at the base of homomorphic encryption schemes are secure, the whole construction around presents several failures, exploitable by a malicious attacker.

Some security notions. When it comes to cryptology, several security notions are proposed. In this chapter we take four of them into account:

- **IND-CPA**: "indistinguishability under chosen plaintext attack". Also called semantic security, is considered as the basic security requirement for a cryptosystem. Briefly, semantic security means that an adversary is not able to distinguish encryptions of 0 from encryptions of 1.

- **IND-CCA1**: "indistinguishability under (non-adaptive) chosen ciphertext attack". The adversary can ask an oracle to decrypt polynomially many ciphertexts of his choice, before the challenge ciphertext is proposed by the challenger. Even with such an advantage he should not be able to distinguish the message encrypted in the challenge.

- **IND-CCA2**: "indistinguishability under adaptive chosen ciphertext attack". The adversary can ask an oracle to decrypt polynomially many ciphertexts of his choice (except the challenge itself) also after the challenge ciphertext is proposed. Even with such an advantage he should not be able to distinguish the message encrypted in the challenge. IND-CCA2 security is one of the strongest security levels.

- **IND-CVA**: "indistinguishability under (chosen) ciphertext verification attack". It is generally considered as an intermediate security notion, stronger than IND-CPA, but weaker than IND-CCA2[1]. The adversary can ask an oracle to verify the validity of polynomially many ciphertexts (except the challenge itself). Even with such an advantage he should not be able to distinguish the message encrypted in the challenge. There exist two flavours of the attack, adaptive and non adaptive, as for the IND-CCA. CVA attacks are strictly linked to reaction attacks, the adversary observes the reaction of the private key owner when he decrypts a tempered ciphertext: this could provide the adversary with some information on the secret key or message. IND-CVA security requires that this situation does not happen.

Homomorphic primitives from the state of the art are proved semantically secure by a direct reduction to an intractable problem. But as homomorphic encryption

[1]Figure 1 of [DDA13] summarizes the connections between the different security notions.

schemes are malleable by construction, they cannot be IND-CCA2 secure: in this case, choosing feature over security seems to be the only possible choice.

IND-CCA1 security is more delicate. In the FHE case, the IND-CCA1 security is contradictory: the bootstrapping principle in fact wants the secret key to be given encrypted as a public parameter, which implies a trivial (non-adaptive) chosen ciphertext attack. For SHE primitives instead, papers such as [LMSV11], [ZPS12] and [CT14] show that almost all the schemes proposed are vulnerable to CCA1 attacks, which all reveal the decryption key. The attack can be realized just by following the steps of the decision-to-search reduction: this applies as instance to the somewhat homomorphic primitives which are based on LWE or its ring variant, as we already said in Section 6.1. The idea consists in making a guess for the first bit of the secret key and randomizing the first coordinate of the sample. Then the answer of the decision oracle about this re-randomized sample reveals the value of the private key bit. By repeating this procedure, we can reveal the entire secret key.

Reciprocally, it appears that for other SHE schemes no CCA1 attacks are known. An example is given in [LGM16b, LGM16a], where the GSW-based scheme generates "one-time secret keys" every time the decryption algorithm needs to be run. In [CRRV17], the authors construct three CCA1-secure FHE schemes by using multy-key identity-based FHE schemes, indistinguishability obfuscation and SNARKs.

At a primitive level, it seems that we are faced with an impossible choice between security requirements or features (FHE, bootstrapping, worst-case assumption). To make a choice, we need to see beyond the homomorphic primitive and consider a broader view of the overall system. Namely, we are interested in finding realistic situations where the cloud would have access to some (possibly weak) decryption oracle.

Imagine the scenario cloud-and-client, where the client has some homomorphically encrypted data on the cloud servers and asks for some computations on these data. If the parameters are well chosen and the cloud does computations as asked by the client (without cheating), the decrypted result is correct. If instead an error occurred during the homomorphic computations, the decrypted result is incorrect with overwhelming probability. In this case, the client receiving an incorrect result is going to ask the cloud to repeat the evaluation and send back the result again.

In such scenario, the client acts as an oracle verifying the correctness of the computations, i.e. an oracle for a ciphertext verification attack (CVA). In this case the IND-CVA security is not achieved for homomorphic encryption, independently on the scheme that has been chosen.

The idea of CVA security dates back to the "Reaction Attacks" of Hall, Goldberg and Schneier [HGS99] in 1999. Instead of targeting the hard underlying problem, their idea was to observe the reaction of the private key owner (the client in our scenario),

when he decrypts a tampered ciphertext. Knowing whether it still decrypts into a valid plaintext, can sometimes provide the attacker with some information on the message or on the secret key. The attack by Bleichenbacher [Ble98] in 1998 against the RSA-PKCS#1 uses similar principles. These notions have been later regrouped and re-defined in 2009 by Hu, Sun and Jiang in [HSJ09] as Indistinguishability under Ciphertext Verification Attack (IND-CVA).
At a primitive level, IND-CVA is usually achieved by requiring a strong padding condition on the message space, like the RSA-OAEP[2]. In the context of homomorphic encryption, IND-CVA does not seem to make sense at the primitive level, especially when ciphertexts are full-domain, or if the message space is too small to encode an intrinsic constraint (like the $\{0, 1\}$ message space). However, this notion should be extended at the system level, where many individually valid ciphertexts are combined together to form some possibly meaningful information. In this case, an attacker could replace a few ciphertexts with other valid ciphertexts, and see if the overall information looks still meaningful to the recipient. As already said in [LMSV11] and [ZPS11], this oracle exists in practice, whenever the client asks the cloud for computations. If the client thinks that the data returned by the cloud is incorrect, especially in an economical model where the client pays the cloud per running times, he certainly asks the cloud for a free re-computation of the result, otherwise he just accepts it. This natural behaviour can be viewed as the response of a CVA oracle, and used as an instrument to retrieve sensitive information. As we show in following sections, it leaks one bit of information per "query".

Safe-Errors and Reaction. The attacks we present are strongly inspired by side-channel attacks in the smart-card domain, but without the need for particular equipment (lasers, probes, etc.). In the cloud scenario, they are simple software attacks. Overall, we perform *safe-error attacks* [YJ00], whose principle is the following: the malicious actor changes a few bits during a computation, and then, he checks if this modification triggers an error later in the process.
In the cloud setting, the semantic security of the black box primitive prevents the attacker from directly obtaining a non-trivial information on the input. This is the second main difference with the smart card domain, where the result can instead be measured immediately after the attack. This is the very reason why in the cloud domain, the attacker needs an observable *reaction* from the client. If no one is aware of this attack, the model is quite realistic: for instance, if the attacker's modification impacts the response at a point that the decrypted text looks gibberish, the natural reaction of the client is to notify the cloud and ask him to re-run the computation. We also want to emphasize the contrast between the simplicity of these attacks, which may be qualified as trivial (as they require almost no mathematical background at all), and the fact that they can be conducted in practice for any use-case of the cloud until now. Furthermore, if heavy countermeasures may be deployed for

[2]Optimal Asymmetric Encryption Padding [BR94].

somewhat homomorphic schemes, they seem impractical and useless for fully homomorphic schemes, underlining an additional antagonism between efficiency and security.

A.1 Safe-errors and reaction attacks in the cloud

The parallelism between smart-cards circuits and homomorphic computations in a cloud is easy to see: in both cases there are circuits operating on hidden data. For instance, the most frequent purpose of a smart card is to compute digital signatures, using a private key which should never be extracted, even by its legitimate owner. Similarly, the client may provide the bitwise homomorphic encryption of a private key to the cloud, and let the cloud homomorphically compute (encrypted) signatures of encrypted data.

It has long been known that smart-cards are vulnerable to side-channel attacks. Some of the attacks are passive, where an attacker gets information on the computation by measuring the running time, the power consumption, some electromagnetic field, or any other side channel information. For example, if the computation of an RSA [RSA78] or (EC)DSA signature [Kra93, JMV01] uses a naive square-and-multiply/double-and-add algorithm, the sequence of operations strongly depends on the number of "ones" and on their positions in the secret key. In smart-cards, these attacks are usually prevented by making the circuit data-independent or *oblivious*, if possible even SIMD parallel. Other perturbations may also be introduced, like adding other arbitrary computations that are not used in the sequel.

In homomorphic computations, the above countermeasures are mandatory by design: all circuits must be oblivious, since the semantic security of homomorphic primitives prevents the cloud from getting any information on the data. In practice, it means that the number of iterations of all *for* loops are publicly known in advance, there is no *while* loop, no data-dependent jump, and the standard way of evaluating an *if-then-else* block is to fully evaluate both possibilities, and in the end, to pick the right result. It also means that simple power or running-time analysis are irrelevant in the cloud, however, fault attacks are still meaningful.

In an active attack on smart cards, the attacker tampers the computation and observes the result in order to gain information on the hidden data. For smart cards, the hardest part is to introduce the fault at a precise moment of the computation, because this usually requires a dedicated physical device.
For this reason, and also the fact that most of these attacks require the card pin code, the attacker is in general the legitimate owner of the card, who just wants to retrieve the hidden key. Furthermore, such attacks have variable success

probabilities, because some parts of a circuit are easier to overheat than others, and all this must be taken into account in their analysis.

By contrast, the cloud provider has a direct and unlimited software access to the whole circuit which is evaluated. He may tamper the result of any gate of his choice with probability 1, at any time. The physical protection of the circuit is replaced by the mathematical shield, which usually consists in saying that every bit is encrypted with a semantically secure scheme. This indeed guarantees that, without an external feedback, no attacker may conduct, on his own, any attack (active or passive) that can reveal sensitive data.
However, some composition of schemes may grant the attacker a weak version of a decryption oracle. For instance, the cloud provider can observe the reaction of a person who owns the private key and who processes the result afterwards, and hope that his behaviour reveals information on a part of the data.

A.1.1 Attacking the data

The idea of *safe-error* attacks is that during the execution of an algorithm, a fault is injected in a precise point. For some secret values, this fault has no effect on the final result, and for some other values, it changes the result completely. So, by observing the correctness of the final result (or in our context the reaction of the legitimate receiver), the attacker can retrieve the target secret value.
By using the techniques from smart-cards on the cloud, we can recreate a more realistic scenario than those involving a universal decryption oracle. Suppose that we want to evaluate a function φ on k ciphertexts $c_1, \ldots, c_k$, encrypting k messages $m_1, \ldots, m_k \in \{0, 1\}$ respectively. If the evaluation is performed without any error, the result is a ciphertext c encrypting $\varphi(m_1, \ldots, m_k)$ (Figure A.1).

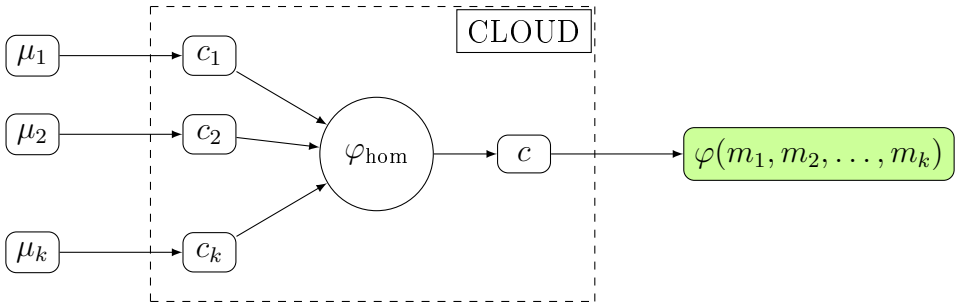

Figure A.1: *Homomorphic computation of the function φ on the cloud.*

But if an error is introduced, the final result could be wrong. For instance, suppose the cloud wants to retrieve the value encrypted in ciphertext c_1. He could just replace this latter with a different ciphertext $\tilde{c_1}$ encrypting 0. Then he performs the

rest of the computations correctly. The result is a faulted ciphertext $\tilde{c}$ encrypting $\varphi(0, m_2, \ldots, m_k)$ (Figure A.2).

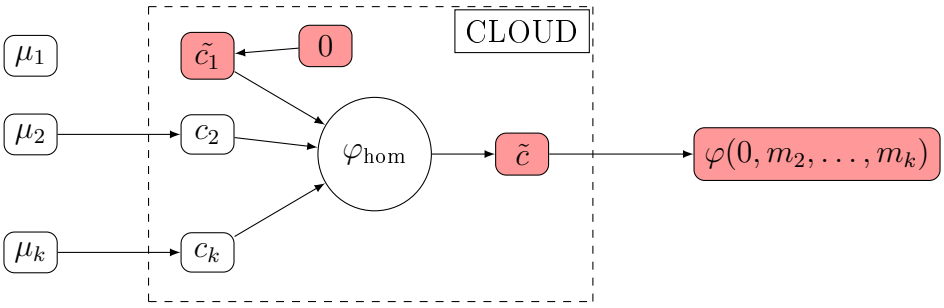

Figure A.2: *Homomorphic computation of the function φ on the cloud with safe-error attack.*

If the decrypted results $\varphi(m_1, m_2, \ldots, m_k)$ and $\varphi(0, m_2, \ldots, m_k)$ are different, it means that an error occurred, so m_1 is different from 0. Otherwise, the guess of the cloud was correct. The cloud cannot verify this equality, because it can access only semantically secure encrypted data.
It is here that the *reaction* of the client plays a crucial role. In our model, upon reception of the ciphertext computed by the cloud, the client decrypts the message and applies some likelihood test. Then, the cloud provider may observe two possible reactions from the client: if the likelihood test failed, the client may directly complain that the result is wrong, and ask for a re-computation. If the likelihood test succeeds, the client simply accepts the result. In both cases, the cloud has received the information he needed to understand if the safe-error was correct or not, and so retrieve the value of the encrypted bit. By repeating the same procedure k times, the cloud retrieves all the k bits.
As a toy example, we can think of a client asking the cloud to compute homomorphically some signatures. He stores on the cloud the encrypted signature secret key and asks to sign some data. If a safe error attack is performed on the signature secret key, the verification of the signature (the "likelihood test") detects immediately if an error was introduced.

A.1.2 Attacking the algorithm

The spectrum of targets of safe-error attacks is wide. As well as an attack on data, the attacker could target the algorithm itself. As instance, he could temper the temporary variables used in the algorithm. We use the RSA square-and-multiply always procedure (Algorithm 13) as a concrete example.
In the process of making an algorithm oblivious, all conditional structures like "*if a then B else C*" are replaced by "*evaluate B, evaluate C, and output $aB + (1-a)C$*".

Algorithm 13 RSA square-and-multiply always

Input: A message m, a RSA secret key $d = (d_0, \ldots, d_{k-1})$, a RSA modulus N.
Output: The RSA signature $m^d \mod N$

1: $t \leftarrow m$
2: **for** $i = k - 2$ to 0 **do**
3: $\quad t_0 \longleftarrow t^2 \mod N$
4: $\quad t_1 \longleftarrow t_0 \cdot m \mod N$ ▷ A safe error could target this line
5: $\quad t \longleftarrow t_{d_i} \mod N$
6: **end for**
7: **return** t

This strategy prevents timing attacks on smart cards, because the set of operations that are executed do not depend anymore on a. However, it is a natural target for safe-errors attacks. Indeed, if an attacker may tamper the execution of the block B, and see if it has a consequence in the following. If it is so, it means that the condition a was true.
In Algorithm 13, in order to homomorphically evaluate $m^d \mod N$, each intermediate step computes a square $t_0 = t^2 \mod N$ and a multiply $t_1 = m \cdot t^2 \mod N$, where t is initially set equal to m. Then the algorithm chooses which value to select depending on the current key bit d_i. If an attacker tampers the multiplication at Line 4, replacing the value of t_1 with any other random (plaintext) data, this safe-error affects the final result with almost certainty if and only if the current private-key bit is equal to 1. Indeed, if the key bit was null, this tampered data would simply have been discarded at Line 5. Once again, the attacker learns one key bit by observing the reaction of the recipient, but this time he does target a portion of the algorithm he is able to understand instead of the data.

A.2 Attacking the bootstrapping principle

In the previous section we showed how to attack a secret value stored in the cloud and encrypted with a generic homomorphic encryption scheme: our target was not the scheme itself. Instead, in this section we apply the attacks directly to the bootstrapping principle to target the secret keys of the HE scheme, highlighting that a safe-error and reaction attack is particularly efficient in this case. We analyze two use-cases of the bootstrapping mechanisms. The first one is used to proxy-re-encrypt a message, encrypted with any scheme, into a bit-wise HE encryption of the same message: this technique is often called *trans-ciphering* and it is commonly proposed as an optimization in order to drastically reduce the communications from the client to the cloud. The second notion is the original bootstrapping proposed by Gentry, used in order to refresh noisy ciphertexts, and to turn a somewhat/leveled homomorphic scheme into a fully homomorphic one.

A.2.1 Trans-ciphering

Homomorphic schemes are known to have a very high ciphertext-versus-plaintext expansion rate. TFHE is one of the most compact schemes, and it uses about 2 KBytes of ciphertext (TLWE) to encrypt one bit of message (more details are given in next chapter). With TRLWE packing (Section 4.1.2), we can reduce this overhead to about 64 bits of ciphertext per bit of message.

The ciphertext expansion rate is particularly annoying when the client has to upload his input data to the cloud over the network. A solution that has been proposed in [NLV11] is to encrypt the data using a traditional symmetric algorithm like AES-CBC, and to send it together with the bitwise-homomorphically encrypted AES key. That way, the data is sent using an amortized 1-1 encryption ratio, which saves a considerable amount of network bandwidth.
In order to perform homomorphic operations on the data, the cloud needs to convert the AES ciphertext into an homomorphic ciphertext, and this is the goal of the trans-ciphering phase: it locally re-encrypts each bit of the AES ciphertext by running the AES decryption algorithm homomorphically. The homomorphic evaluation of the AES circuit has already been intensively studied, and for example [GHS12] takes a particular care optimizing its running time. Recent results from [CCF+16] and [MJSC16], show that different schemes, such as some stream ciphers, can be more "FHE friendly" than AES for trans-ciphering.
This whole trans-ciphering construction has a huge drawback: the secret key of the first scheme becomes a secret data encrypted with the secret key of the homomorphic scheme. This allows to apply the attack presented in previous section, with the difference that, targeting the secret key reveals much more than just the 128 AES key bits. It allows the attacker to decrypt the whole input data. The main security properties of symmetric ciphers guarantees that any modification of a single key bit of a secret key renders the whole data gibberish, and certainly triggers the expected reaction from the client. In a practical attack scenario, if the input data is AES-encrypted with a 128-bit AES key, the attacker may prefer to perform the attack on first 80 key bits, which only triggers about 40 negative feedback from the client. Then, the attacker may simply brute-force the remaining 48 bits off-line and decrypt all the client's data.

A.2.2 Bootstrapping

As we saw in Chapter 5, bootstrapping can be used to refresh noisy ciphertexts, or even to homomorphically evaluate different functions or change the message/ciphertext space. The noise of the output ciphertext only depends on the (fixed) noise of the bootstrapping key, and not on the noise of the input ciphertext, which may reach the maximal level. Bootstrapping has been viewed as a way of bounding the noise growth, or of refreshing the ciphertexts after each (or after some)

elementary homomorphic operation.
In all cases, the algorithm that is homomorphically evaluated during a bootstrapping procedure, or the refreshing procedure for schemes "without bootstrapping", is always equivalent to the decryption function. Thus recovering the bootstrapping key via safe-error totally breaks the onewayness of all schemes based on circular security. Schemes which do not rely on circular security but are still considered fully homomorphic provide, instead, a long chain of distinct bootstrapping keys, where each one is encrypted with the next one. In this case, the safe error attack needs to be applied only on the last bootstrapping key of the chain: once it is recovered, we can use it to decrypt the whole chain of keys. Again, this totally breaks the onewayness.

A.3 Countermeasures

The attacks we pointed out in this chapter are very simple. Yet, they work even on perfect black-box primitives, and they have catastrophic consequences on data privacy if nothing is done at the system level to prevent them. Furthermore, we worked under the optimistic assumption that the requested computation had a unique solution, and that any error would be detected by the client's likelihood test. Of course, the situation is much worse if the problem has multiple valid solutions, and each of those induces different visible reactions later in the process. This includes of course any white-box use of the cloud, where the final result is decrypted and published by the client. The cloud may hide some key-dependent ciphertexts inside the least significant bits of floating point statistics over encrypted medical data, or in the random bits of NTRU-like signatures it was asked to compute. These attacks are even more dangerous than those we presented in the previous sections, because the client may hardly detect the attack, and in the mean time, the cloud provider is left with a universal decryption oracle.
In this section, we study some possible countermeasures/precautions to prevent these simple attacks, and show that the client must be very strict concerning its trust model towards the cloud.

Asymmetric versus symmetric encryption? One possible countermeasure we may think of is to try to use symmetric schemes, instead of public key constructions. Indeed, the direct attack on the hidden data requires the cloud to replace a ciphertext with a valid ciphertext of 0 (or 1), so the goal of the countermeasure would be to prevent him from being able to generate such ciphertext. Unfortunately, this countermeasure does not really apply to homomorphic constructions. The large malleability of these schemes allows them to evaluate potentially any function (including the constant 0 or the constant 1 functions) and thus, to forge valid ciphertexts of 0 or 1 at will. For instance, if the homomorphic scheme includes the (homomorphic) `XOR` or the (homomorphic) `NAND` gates, a valid ciphertext of 0 may be obtained by

computing $\texttt{XOR}(c, c)$ for any valid ciphertext c, and a valid ciphertext of 1 can be obtained as $\texttt{NAND}(c, \texttt{NAND}(c, c))$.

Work on large blocks (no bitwise encryption)? A second proposal would be to increase the length of the message space, so that homomorphic primitives operate on larger blocks (for instance on 128-bits blocks) instead of just zeros and ones. In this case, an attacker would only be able to tamper a whole block of data, and, unless he guesses the right 128-bit value, it would always produce a reaction from the client, which does not bring much information. But once again, this approach is bound to fail for two reasons: it does not prevent the case where the attacker targets a value which is not used in the sequel (as in the algorithm attack), which may be replaced by any random (plaintext) value. And most importantly, once again, the large malleability of homomorphic primitives allows the attacker to homomorphically evaluate any function on a 128-bit block, including the function ϕ_i which takes a block as input and sets its i-th bit to zero. The attacker can therefore replace a ciphertext containing the block message m with the ciphertext containing $\phi_i(m)$ and perform the attack. In this case, the reaction of the client reveals the value of the i-th bit of m, as if the homomorphic primitive was just operating on binary messages.

Do not reuse trans-ciphering keys? Concerning the trans-ciphering attack, one could think that asking the client never to re-use the same AES key twice is enough. In particular, the encrypted AES key cannot be set as permanent key parameter, it must be generated and sent by the client over the network with every new dataset.
Actually, the situation is even more complex than that. The actual requirement is that the cloud must never be allowed to re-use the same data twice. For instance, if the client used the AES trans-ciphering to transfer a whole database to the cloud, this database must only be used to answer a single client's query. Indeed, for the second query, the cloud may just silently re-run locally the trans-ciphering with another safe-error, and get another key bit, and so on until the 128-th query. Obviously, in most practical situations, the client is not able to prevent the cloud from re-using the same data in many computations, so safe-errors should be considered as a huge threat against the trans-ciphering strategy.

Add obfuscation? The nature of the attacks strongly suggest that the location of important data bits must be hidden from the cloud, and the function must be impossible to reverse-engineer or to understand. Achieving these goals is usually called obfuscation. In order to prevent the attack on the data bits, an idea would be to add some error correcting code. For instance, instead of computing $f(x)$ where x represents n bits of data, the algorithm would first compute $y = g(x)$ where g is some random error correcting code of large distance (between different encoded "letters")

$d > n$, and the hidden data y has at least $n + 2d$ bits. In order to successfully invalidate the result, an attacker would then need to flip at least d bits of data, i.e. to fix $2d$ bits to arbitrary values. Of course, this is more a hint than an actual workaround, for the following reasons:

- The code should not be perfect, and if possible non linear. Else, the reaction attack of [HGS99] against the McEliece cryptosystem can be adapted. Basically, fix bits of data one by one to arbitrarily values until the client reacts. At that point, if the code is perfect, there are exactly $d+1$ errors, so the last bit set is wrong, and since there are already d errors among the others bits fixed, each additional error induces a reaction from the client. So the classical safe error can be conducted on the remaining data bits one by one.

- The composition between the homomorphic decoding $y = g(x)$ and the function computation $f(y)$ must be obfuscated to the cloud. Else, the cloud provider may just run the code part homomorphically, and once it has the bits of y, do the regular attack on them.

- For the same reason, the part of the circuit which computes f must be obfuscated, else the attack on the function may be directly performed.

- If g represents the encoding function of a non-linear random looking code, its complexity has a serious impact on the size of the overall circuit, and therefore on the parameter sizes of the homomorphic scheme. Besides, the fact that the whole composition $f \circ g$ must be obfuscated worsens the situation.

- In trans-ciphering and bootstrappable constructions, using obfuscation would considerably increase the parameters and the complexity.

At the time of writing, there exists no systematic obfuscation technique: IO-obfuscation [BGI$^+$01, GGH$^+$13] is not yet practical, neither secure under any standard assumptions.
One of the purposes of homomorphic encryption was to provide a systematic way of obfuscating a computation. But now, we see that in order to resist against simple safe-errors, the computation has to be obfuscated again with something more powerful. Obviously, taking another homomorphic scheme as the larger obfuscation seems to create a vicious circle. And if the computation is protected with another ad hoc obfuscation, then what is the purpose of the first homomorphic scheme?

Do not forgive any mistake. It seems that the safest approach for the client is to distrust the cloud immediately whenever he receives a ciphertext which does not correspond to any realistic result. For instance, if the goal of the homomorphic scheme is to compute signatures and the received signature is invalid, or if the data looks like a random binary sequence instead of a plaintext message. This

also implies that the underlying homomorphic primitive must be error-free. Indeed, many LWE-based homomorphic schemes which were recently proposed, including [AP14], [DM15] and our TFHE construction, were able to drastically reduce the parameter sizes by requesting that the homomorphic operations are randomized, but with the counter-effect that even an honest homomorphic computation has a small error-probability per gate in the circuit. Namely, the parameter set proposed in [DM15], which allows to bootstrap the scheme in less than a second, has an error probability of 2^{-32}, which means that the result of an homomorphic computation may be wrong even if no safe-error was introduced. The same is valid for TFHE: we give detailed parameters in next chapter. Due to the nature of the attack we are pointing out, allowing errors can be really devastating, since the client would not be able to distinguish between an attack and an honest error due to the homomorphic primitives.

Random computations. We could ask the cloud for random computations : for instance if we need to compute k signatures every day, we may ask the cloud for $2k$ signatures, where k are random computations. If the cloud performs its attack on day 1, and the client detects an error, he substitutes the random computations of day 2 with re-randomized ciphertexts from day 1, and send to the cloud $2k$ computations as always. The cloud does not detect any strange reaction from the client, and its attack fails. But this countermeasure is probably really costly, especially in the economical model where the client pays per running time.

High entropy data. In the SHE or LHE settings, we should ensure that the data flow has high entropy at all times, and that changing one bit of a ciphertext is impossible, either because it exceeds the allowed noise bound (LHE), or because the operation itself cannot be expressed as a valid homomorphic operation (SHE). This implies that ciphertexts must encode multibit messages, and it may also require a secret key mode, where the attacker cannot easily obtain valid ciphertexts of arbitrary plaintexts. This countermeasure may be the most promising countermeasure for LHE schemes, but is not applicable to FHE because of its large malleability.

Verifiable computation. The last countermeasure is to require verifiable computation. The most straightforward solution is to completely derandomize all cloud operations, and to ask two independent cloud services to do the exact same computations. If the two encrypted results are equal, then the client can hope that no attack has been performed. This requires of course that the two cloud providers do not collude. In a multi-user setting, this can be achieved by including incentive measures to the system, so that users are rewarded when they verify the computation (like in the Bitcoin protocol). Again, this can be expensive in terms of costs and network bandwidth for the client, unless the function is simple enough and its total computation time is not too high (like in the e-voting protocol of [CGGI16b]).

Another possibility is to force the cloud to produce a zero-knowledge proof of correct computation, to prevent the attack on the algorithm, or a proof to demonstrate that all the ciphertexts used in the computation were effectively the ones sent by the client, to prevent the attack to the data. Recent works [CFW14], [CF13], [CFGN14], [FMNP16], show how to practically construct homomorphic signatures, MACs and authenticators for both leveled and fully homomorphic encryption schemes. Roughly speaking, those tools allow a user (or multiple-users) to verify if the computations done by the cloud are correct and if the set of data taken in input is actually the good one. This indeed prevents safe errors. Intuitively, to verify the validity of a (deterministic) polynomial function against random errors, it suffices to check all the computations modulo a fixed λ-bits number N, where λ is the security parameter. We refer to the above citations for more details. Overall, the verification process has the same number of steps than the full homomorphic computation, but if λ is smaller than the actual data-types of ciphertexts, the verification of the checksum may be faster that the entire computation[3].
Therefore, all these techniques could be used in a larger scenario, where multiple users ask for small computations, rapidly and constantly verified, as it happens in blockchain-type scenarios.
Another way to verify computations could be the use of *Probabilistically Checkable Proofs (PCP)* [Cha01]. They seem similar to zero knowledge proofs, but they are very different in practice: the verifier (i.e. the client) wants to be convinced that the computations done by the prover (i.e the cloud) are correct, without checking every passage, but just a few. The cloud should produce a polynomial size proof of good computation and the client should verify just a constant number of passages to be sure (with high probability) that the proof, and so the computations for which the proof has been produced, are correct. The idea would be to create a compiler able to transform the entire system of computations in an equivalent system, on which a simple error propagates everywhere: this would imply that the verification of a few passages in the second system reveals the error with high probability. This is still an open question.
Another possible trail to solve this problem could be to use zero knowledge SNARGs (succinct non-interactive arguments) and SNARKs (succinct non-interactive arguments of knowledge), to produce short and fastly verifiable proofs [GGPR13, PHGR13]. These proofs are universal/Turing complete, so they are promising in the FHE context. On the other hand, they seem to require a trusted setup phase. Yet, the subject deserves a deeper study.

Homomorphic encryption is vulnerable against malicious attacks, in particular in the cloud scenario: countermeasures exist but they are yet too costly. Before we

[3]This is just an intuition, but this does not apply to [DM15] or the TFHE schemes, which operate on small 32-bit primitive types. Packing multiple 32-bit words together in these systems does not correspond to low degree polynomial functions anymore.

study the complexity of operations on a real-word cloud application, we should understand if there exist scenarios in which the cloud is incentivized to perform correct computations, or if he has to inevitably be considered an adversary.

Appendix B

Application: a homomorphic LWE based e-voting scheme

In this chapter we describe a homomorphic based electronic voting scheme, which is one of the many applications of FHE. The scheme we describe was presented for the first time at PQ Crypto in 2016 [CGGI16b] and it could be considered as the starting point for TFHE. To our knowledge, the scheme is the first post-quantum solution proposed for electronic voting. A new post quantum lattice-based e-voting scheme has been recently proposed in [dPLNS17].

Our scheme is inspired by existing e-voting protocols, such us Helios [AdMP] and its variant Belenios [GCG], and the main construction is based on LWE fully homomorphic encryption. When we designed the scheme, we decided to use the FHE construction by [DM15] with a few modifications. TFHE came later, with the initial goal to be used for a practical realization of the e-voting scheme. Nevertheless, during the process of developing TFHE we lost sight of the e-voting and we "forgot" to verify if the improvements would work as predicted for this application.

The goal of this chapter is to detail the main steps of the scheme, and to give an intuition on a possible practical implementation of the scheme by using TFHE.

General overview of the scheme

Electronic voting can be seen as the electronic analogue of paper voting, which is largely used in most of the countries for elections. To be really comparable with paper voting, electronic voting has to satisfy many security properties: in particular privacy, verifiability and correctness can be considered the basics.
Roughly speaking, a scheme is

- *Private*: if it prevents anyone from retrieving the vote of a particular voter.

- *Verifiable*: if it allows each voter to verify that his vote has been counted (individual verifiability) and ensures that the final count of votes corresponds to the votes of legitimate voters (universal verifiability).

- *Correct*: when the outcome of the election counts only the votes that have been honestly generated.

Among other desirable properties for e-voting schemes, there are strong forms of privacy such as receipt-freeness and coercion-resistance (a voter cannot prove that he voted in a certain way), or ballot independence (a voter cannot produce a valid vote related to the vote of someone else, while observing the interaction of this latter with the election board).
Defining security properties for electronic based systems has long been debated and the design of secure e-voting protocols achieving all these properties happens to be more intricate than for traditional paper-based systems. Many interesting proposals have appeared in the last years, but their security still relies on classical assumptions, which means that these proposed schemes could all be compromised if efficient quantum computers arise. To propose a quantum resistant e-voting scheme could be a good approach to comfort people in using e-voting protocols.
In order to reach this aim, we built a post quantum secure e-voting scheme, by using only post quantum secure building blocks, such as unforgeable lattice-based signatures, LWE-based homomorphic encryption and trapdoors for lattices. Our scheme is simple and transparent since it relies on homomorphic operations: we perform on the ballots (almost) the same operations we would have performed on the clear votes.

The scheme is strongly inspired by existing e-voting protocols, in particular Helios [AdMP], which has already been used for medium-scale elections, and its variant Belenios [GCG]. However, our scheme differs in two principal ways.
The underlying primitive is different: Helios is a remote e-voting protocol based on the additive property of ElGamal (which is broken by Shor's quantum algorithm [Sho97]). In this variant of the ElGamal scheme, the messages are encrypted as exponents instead of as multiplicative factors. This means that the multiplication of two ciphertexts, encrypted with the same key, produces an encryption of the sum of the messages, instead of their product.
Since the additive homomorphism lacks some expressiveness, each voter must ensure that the plaintext encrypted in their ballot has a specific shape, suitable for homomorphic additions. The vote is expressed as a sequence containing as many elements as the candidates of the election: one point is given to the chosen candidate and 0 points are given to the others. Of course, all the points are encrypted, but the voter must prove that the vote is constructed in this way. Proving such property without revealing any information on the content of the vote is usually achieved using zero-knowledge proofs.
In our protocol, the fully homomorphic encryption allows to efficiently transform

full-domain ciphertexts into such ciphertexts with specific semantic. This effectively removes the need of a zero-knowledge proof.

In Helios, another zero-knowledge proof is used at the end of the protocol: the trustees, i.e. the authorities charged with the decryption of the final result, have to prove that they decrypted correctly without revealing their secret. Once again, in our protocol we replaced the zero-knowledge proof with a different technique, we call *publicly verifiable ciphertext trapdoor*. This latter is produced using techniques borrowed from trapdoor-based lattice signatures, GPV [GPV08], or [MP12], based on Ajtai's SIS problem.

In order to prevent the bulletin board (the entity charged of casting and processing the ballots) from stuffing itself the ballots, we add a further authority in charge of providing each user with a private and public credential which allows him to sign his vote. This solution was already used in the variant of Helios proposed in [CGGI14].

Cortier and Smyth [CS11, Smy12] show that homomorphic based e-voting protocols, and in particular Helios, could be vulnerable to replay attacks, allowing a user to cast a vote related to a previously cast ballot. This type of attacks could possibly incur a bias on the vote of other users and break privacy. Although this attack has a small impact in practice, the model for privacy should capture such attacks. Until now, this attack is prevented by removing ballots which contains a ciphertext that does already appear in a previously cast ballot. This operation is called cipherext weeding. This strategy would not work with fully homomorphic schemes, as bootstrapping operations would allow an attacker to re-randomize duplicated ballots beyond anything one can detect.
In order to prevent these attacks, we use the one-wayness of the bootstrapping to create some "plaintext-awareness" auxiliary information. This auxiliary information can be seen as another encryption of the same ballot, it does not leak information on the plaintext vote and its only purpose is to guarantee that the ballot has not been copied or crafted from other ballots in the bulletin board.
The voter encrypts this auxiliary information and sends it with his ballot: the information remains encrypted in the bulletin board until the end of the voting phase and at this point, for the sake of transparency, it could be safely revealed to everyone. In practice, we model this temporarily private channel by giving a public key to the bulletin board, and letting him reveal the private key at the end of the voting phase.

Finally, in order to guarantee privacy even when some of the authorities keys are corrupted, we show that our encryption scheme can be distributed among t trustees. Instead of using a threshold decryption based on Shamir's secret sharing, we rely on a simple concatenated LWE scheme. Each of the trustees carries its own decryption

part, and any attempt to cheat is publicly detected.
On one hand, we lose the optional ability to reconstruct the result if some trustees attempt a denial of service: this can be prevented anyway by taking the appropriate legal measures. On the other hand, once the public key has been set, we detect any attempt to cheat even if all the trustees collude. The privacy of the votes is preserved until at least one trustee is honest. Thus, our protocol can be instantiated with only two trustees which operate independently. In comparison, at least three trustees are needed for Shamir's interpolation, and if they all collude, they could produce a valid proof for a false result.

We use to (informally) call our e-voting scheme *sPQlios*, for Secure Post Quantum scheme inspired by heLIOS. As TFHE was made after the e-voting scheme and we never made the name official, we decided to make it appear somewhere: our FFT implementation was named after that.

Correctness and Security

The security of our scheme is based on classical SIS/LWE assumptions, which are asymptotically as hard as worst-case lattice problems and we rely on the random oracle heuristic.
Our e-voting scheme satisfies the basic security properties of privacy, verifiability and correctness.

- **Privacy -** For the privacy, we use the strongest game-based ballot privacy recently introduced by [BCG+15, Definition 7]. In the privacy definition we simulate a game where two bulletin boards are maintained by the challenger. The adversary has the power to corrupt a subset of trustees, vote for a candidate of his choice and cast ballots in only one of the two bulletin boards. At the end of the voting procedure, the adversary should not be able to distinguish the two bulletin boards.

- **Verifiability -** For verifiability, we use the definition proposed in [JCJ10]. We say that the voting protocol is verifiable if it ensures that the tally verification algorithm does not accept two different results for the same view of the public bulletin board. In the security model for verifiability, we suppose that the adversary is able to corrupt all users but not the authority $\mathcal{A}_1$.

- **Correctness -** For the correctness, we use the definition proposed in [JCJ10]. If the users follow the protocol, an e-voting scheme is correct when the tally leads to the result of the election on the submitted votes. The correctness is proved in the case where the bulletin board is honest, which means that he is not allowed to stuff or suppress valid ballots cast by honest users.

For more details on the proper security definitions and for the proofs we refer to our paper [CGGI16b]. The improvements regarding stronger security proofs in the case of malicious bulletin board and/or corrupted registration authority are still open problems. We would also like to provide a stronger privacy proof without modeling the randomized bootstrapping function as a random oracle.

Bootstrapping as a random oracle In the rest of the chapter we make the hypothesis that the output of the bootstrapping function is indistinguishable from a fresh LWE sample. The bootstrapping seems to behave like a good collision-resistant one-way function, especially if we re-randomize the input sample by adding a random combination of the public key. However, for verifiability purposes, one may also wish to control the randomness to reproduce some computations. To simplify the analysis, we will therefore model bootstrapping as a random oracle.

B.1 E-voting scheme

E-voting schemes can be different depending on the way the result is computed, on the number of votes we have to express, on the type of answer to the vote, etc. Our scheme is basically a single pass e-voting scheme, but it could be adapted for different purposes.
In a single pass e-voting scheme, each user publishes only one message to cast his vote in the bulletin board. A voting scheme is specified by a family of result functions denoted as $\rho : (\mathcal{I} \times \mathbb{V})^* \to \mathcal{R}$ where $\mathbb{V}$ is the set of all possible votes, $\mathcal{I}$ is the set of voters' identifiers, $\mathcal{R}$ specifies the space of possible results. A voting scheme is also associated to a re-vote policy. In our case, we assume that the last vote is taken into account.
The entities implied in the voting procedures, apart from the voters/users, are:

- $\mathcal{A}_1$: the authority that handles the registration of users and updates the public list of legitimate voters.
- BB: the bulletin board, that checks the well-formedness of received ballots before they are cast. In our model, we assume that BB uses a secret key to perform a part of this task but the secret could be revealed after the voting phase. In the following, we improperly use the notation BB to indicate both the bulletin board manager and the bulletin board itself.
- $\mathcal{T}$: a set of trustees in charge of setting up their own decryption keys, and computing the final tally function.

In this chapter we denote as ℓ the number of candidates, L an upper bound on the number of voters and t the number of trustees. We denote as $\mathcal{L}_{\mathcal{U}}$ a public list of users set at empty at the beginning.

To simplify the description, we assume an authenticated private channel between the trustees. We suppose we are given an unforgeable signature scheme we denote $\mathcal{S} = (\mathsf{KeyGenS}, \mathsf{Sign}, \mathsf{VerifyS})$ and a non-malleable encryption scheme $\mathcal{E} = (\mathsf{KeyGenE}_{\mathsf{BB}}, \mathsf{Enc}_{\mathsf{BB}}, \mathsf{Dec}_{\mathsf{BB}})$. We suppose that both, the signature scheme and the non malleable encryption scheme are quantum resistant.

The e-voting procedure is divided in three main parts:

1. The *setup* phase: the parameters and keys are set and the voters register after the authority $\mathcal{A}_1$:

2. The *voting* phase: the voters product and send the ballots to the bulletin board. This latter processes the ballots and computes (homomorphically) the encrypted final result.

3. The *tallying* phase: the trustees decrypt the final result and produce the publicly verifiable ciphertext trapdoors. Thus, everyone can verify the result, even afterwards.

All these phases make use of some algorithms that could be defined as follows:

- $(\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_t), \mathsf{params}) \leftarrow \mathsf{Setup}(1^\lambda, t, \ell, L)$: Each trustee chooses its secret key sk_i and publishes a public information pk_i and proves that it knows the corresponding secret with respect to the published public key. The bulletin board runs $(\mathsf{pk}_{\mathsf{BB}}, \mathsf{sk}_{\mathsf{BB}}) \leftarrow \mathsf{KeyGenE}_{\mathsf{BB}}(1^\lambda)$, it publishes $\mathsf{pk}_{\mathsf{BB}}$ and keeps $\mathsf{sk}_{\mathsf{BB}}$ private. This step implicitly defines the public pk of the e-voting scheme that includes $\mathsf{pk}_{\mathsf{BB}}$. The parameters params includes the public key pk, the numbers t, ℓ, L, the list $\mathcal{L}_{\mathcal{U}}$ and the set of valid votes $\mathbb{V}$. All these parameters are taken as input in all the following algorithms.

- $(\mathsf{usk}, \mathsf{upk}) \leftarrow \mathsf{Register}(1^\lambda, \mathsf{id})$: Takes in input a security parameter and a user identity and provides the secret part of the user credential usk and its public part upk. It updates the public list $\mathcal{L}_{\mathcal{U}}$ with upk.

- $b \leftarrow \mathsf{Vote}(\mathsf{pk}, \mathsf{usk}, \mathsf{upk}, v)$: It takes as input a secret credential and public credential that possibly includes id and a vote $v \in \mathbb{V}$. It outputs a ballot $b = (\mathsf{aux}, \mathsf{upk}, c, \mathsf{num}, \sigma)$ which consists in a content message that includes an auxiliary information aux encrypted using the key $\mathsf{pk}_{\mathsf{BB}}$, the public key upk, an encryption c of the vote v, a version number num, used for the re-vote policy, and a signature of this content message under the secret usk.

- $\mathsf{ProcessBB}(\mathsf{BB}, b, \mathsf{sk}_{\mathsf{BB}})$: As long as the bulletin board is open, when the bulletin board manager receives a ballot b, he parses it as $(\mathsf{aux}, \mathsf{upk}, c, \mathsf{num}, \sigma)$, he verifies that $\mathsf{upk} \in \mathcal{L}_{\mathcal{U}}$ and uses upk to verify the signature of the ballot. Then he decrypts aux using $\mathsf{sk}_{\mathsf{BB}}$, he performs a validity check on b and upk

and finally verifies the re-vote policy with the version number. If b passes all these checks, it is added in BB, otherwise BB remains unchanged.

- $(\Pi_1, \ldots, \Pi_t) \leftarrow \mathsf{Tally}(\mathsf{BB}, \mathsf{sk}_1, \ldots, \mathsf{sk}_t)$: Once the voting phase is closed and the public bulletin board is published together with $\mathsf{sk}_{\mathsf{BB}}$, each trustee $T \in \mathcal{T}$ takes as input the public bulletin board BB, and its own secret key to produce a partial proof Π_i, which is publicly disclosed.

- $\mathsf{VerifyTally}(\mathsf{BB}, (\Pi_1, \ldots, \Pi_t))$: (public) takes as input t partial proofs associated to a given bulletin board BB and verifies that each individual proof Π_i is correct, and uses all of them to decrypt. It outputs a final result r and $\perp$ in case of failure.

B.2 More homomorphic building blocks

In order to construct the e-voting scheme, we describe two additional building blocks: the publicly verifiable ciphertext trapdoors and the concatenated encryption. Both techniques are used by the trustees during the tallying phase.

B.2.1 Publicly verifiable decryption for LWE

In previous chapters, all the homomorphic constructions we described were presented in their symmetric version. In e-voting we need the public key version of the cryptosystem. In fact, the voters need the homomorphic public key generated by the trustees in order to encrypt their ballots.
In homomorphic encryption schemes, one usually publishes a polynomial number $m = \Omega(n \log(1/\alpha))$ of random TLWE samples of the message 0 with standard deviation parameter α. The public key can be written as a matrix $\mathsf{pk} = [M|\mathbf{y}] \in \mathbb{T}^{m \times n} \times \mathbb{T}^{m \times 1}$ where $\mathbf{y} = M\mathfrak{K}^T + \mathbf{err}$, where $\mathfrak{K}$ is the TLWE secret key. The public encryption of a message $\mu \in \mathbb{T}$ can then be achieved by chosing a random subset (of rows) of the public key, and adding them to a trivial ciphertext of μ, as shown in Figure B.1. We call this operation $\mathsf{TLWE}^{\mathsf{Pub}}{}_{\mathsf{pk}}(\mu)$.

In the protocol we present, this allows a voter to encrypt his vote. Then the BB can publicly use the bootstrapping theorem to homomorphically evaluate whatever circuits produces the (encrypted) final result. And in the end, some trustee must decrypt this result using the secret key. If decrypting a TLWE ciphertext on a discrete message space is easy, proving to everyone that the decryption is correct without revealing anything on the TLWE secret key requires some more work.
To do so, we adopt a strategy which is borrowed from lattice-based signatures like GPV [GPV08]. To allow a public decryption of $\mathfrak{c} \in \mathbb{T}^{n+1}$, we reveal a small integer combination $(x_1, \ldots, x_m)$ of the public key pk which could have been used to encrypt

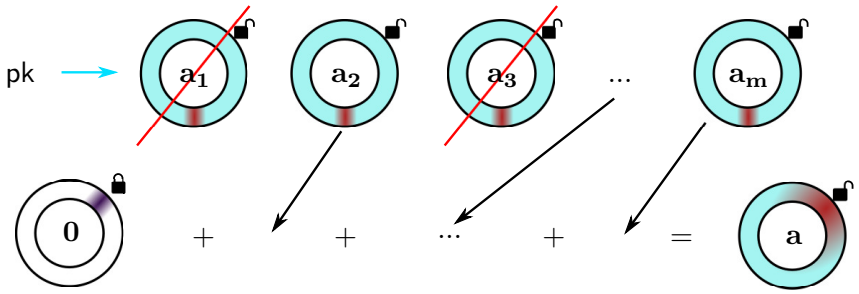

Figure B.1: TLWE asymmetric encryption - *We use the same notations as the figures presented in Chapter 3: the public key elements are encryptions of* 0 *(bottom of the torus). The homomorphic addition between a trivial ciphertext of the message and a random subset of elements of the public key produces a (non-trivial) encryption of the message.*

$\mathfrak{c}$. We call this combination *publicly verifiable ciphertext trapdoor*, and we abbreviate it as PVCT.

Definition B.2.1 (Publicly Verifiable Ciphertext Trapdoor (PVCT))**.** *Let* $\mathfrak{K}$ *be a* TLWE *secret key and* $\mathsf{pk} = [M|\mathbf{y}] \in \mathbb{T}^{m\times(n+1)}$ *be the corresponding public key. Let* $\mathfrak{c} = (\mathfrak{a}, \mathfrak{b})$ *be a valid* TLWE *sample with packing radius* d*. We say that* $\mathbf{x} = (x_1, \ldots, x_m) \in \mathbb{Z}^m$ *is a PVCT of* $\mathfrak{c}$ *if* $\|\mathbf{x}\| \leq \beta$*, with* $\beta = \sqrt{\frac{d^2 - \|\mathsf{Err}(\mathfrak{c})\|_\infty^2}{2\pi\sigma^2}}$*, and if* $\mathbf{x} \cdot M = \mathfrak{a}$ *in* $\mathbb{T}^n$*.*

Anyone who knows the public key can verify the correctness of the ciphertext trapdoor. Furthermore, since the difference $\mathfrak{c} - \mathbf{x} \cdot \mathsf{pk}$ is a trivial ciphertext $(\mathbf{0}, \varphi)$ of a phase φ corresponding to the message μ, this reveals the message at the same time. Figure B.2 summarizes the idea behind PVCT.

Of course, finding a small combination of random group elements which is close to some target is related to the subset sum or the SIS family of problems, which are hard in average. Luckily, the framework proposed in [MP12] introduces an efficient trapdoor solution. We summarize their ideas in the following definitions: Definition B.2.2 is adapted from [MP12, Definition 5.2] and Definition B.2.1 is adapted from [MP12, Theorem 5.1].

Definition B.2.2 (Master Trapdoor)**.** *Let* $\mathfrak{K}$ *be a* TLWE *secret key and let* $\alpha \in \mathbb{R}_+$*. A Gadget* $G \in \mathbb{T}^{m'\times n}$ *is some publicly known superincreasing generating family of* $\mathbb{T}^n$*, such that any element* $\mathbf{a} \in \mathbb{T}^n$ *can be approximatively decomposed as a small (or binary) linear combination of* G*. Let* A *be a uniformly distributed family in* $\mathbb{T}^{(m-m')\times n}$*, and let* R *be a* $m' \times (m - m')$ *integer matrix with (small) SubGaussian entries. We define the matrix* $M = \left[\frac{A}{A'}\right] \in \mathbb{T}^{m\times n}$ *where* $A' = G - R \cdot A$*. We call*

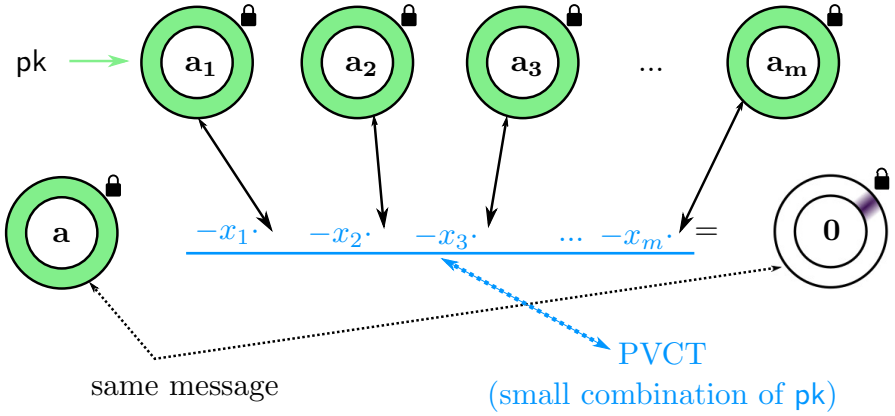

Figure B.2: PVCT - *A PVCT is a small combination of the public key* pk *that could have been used to encrypt the message. By substracting this combination to the ciphertext, we obtain a trivial ciphertext of the message.*

R a master trapdoor*, and its corresponding public key is* $\mathsf{pk} \in \mathbb{T}^{m\times(n+1)}$*, whose i-th row is* $\mathsf{pk}_i = (M_i | M_i \cdot \mathsf{sk} + e_i)$ *for some Gaussian noise e_i of standard deviation α. The master trapdoor verifies the condition $G = [R|Id_{m'}] \cdot M$ and the parameters $m, m', m - m' = O(\log_2(\#\langle G\rangle))$, where we improperly denote with $\langle G\rangle$ the elements of $\mathbb{T}^n$ whose decomposition with respect to G is exact.*

Theorem B.2.1 (Trapdoor)**.** *Let $\mathfrak{K}$ be a* TLWE *secret key and let $\alpha \in \mathbb{R}_+$ a standard deviation. Let $\mathfrak{c} \in \mathbb{T}^{n+1}$ be a* TLWE *sample on a message space of packing radius $> 2d$, R a master trapdoor of standard deviation γ and* pk *an associated public key with noise $< d/\gamma \log(\#\langle G\rangle)^{1.5}$. Given $\mathfrak{c}$ and R, one may efficiently compute a ciphertext trapdoor $\mathbf{x}$ for $\mathfrak{c}$ of norm $O(\beta \log(\#\langle G\rangle)^{1.5})$. This trapdoor can decrypt $\mathfrak{c}$, as in Definition B.2.1. Furthermore, the distribution of the ciphertext trapdoors of $\mathfrak{c}$ is statistically close to some discrete Gaussian distribution on $\mathbb{Z}^m$, of parameter $O(\beta \log(\#\langle G\rangle)^{0.5})$, and thus, does not reveal any information about R.*

Ciphertext trapdoors are trivially vulnerable to chosen ciphertext attacks (CCA), as they correspond to a decryption oracle, so they should only be invoked on the output of some good hash function, or some random oracle. This is the case in our scheme, where every PVCT is produced on an output of the bootstrapping algorithm, which we model as a random oracle.

B.2.2 Concatenated TLWE with distributed decryption

To prevent a single authority from decrypting individual ballots and to guaranty privacy in the long term, even if all but one trustee leaks its private key, we need to split the TLWE secret key among multiple trustees. Instead of adopting a threshold

decryption solution, like Shamir's secret sharing, we decided to use a simple concatenation of TLWE systems where all the trustees must do their part of the decryption, and any cheater is publicly detected.
This requirement seems sufficient for an e-voting scheme. On one hand we lose the ability to reconstruct the final result if not all the trustees collaborate in the decryption process: this problem could be solved in practice by adopting appropriate legal measures. On the other hand, we are able to guarantee privacy until at least one of the trustees is honest, which implies that the scheme is secure even with only two trustees.

Let $\mathfrak{K}_1, \dots, \mathfrak{K}_t$ be t different λ-bit secure TLWE secret keys, and $\mathsf{pk}_1, \dots, \mathsf{pk}_t$ be corresponding public keys such that $\mathsf{pk}_i = [M_i|\mathbf{y_i}] \in \mathbb{T}^{m\times(n+1)}$ be the corresponding public keys with associated master trapdoors R_i for all $i \in [\![1, t]\!]$.
We call *concatenated* TLWE the instance whose private key is $\mathbf{s} = (\mathbf{s_1}|\dots|\mathbf{s_t})$, and public key is

$$\mathsf{pk} = \left[\begin{array}{c|c|c|c} M_1 & 0 & 0 & \mathbf{y}_1 \\ \hline 0 & \ddots & 0 & \vdots \\ \hline 0 & 0 & M_t & \mathbf{y}_t \end{array}\right]. \tag{B.1}$$

To decrypt a (concatenated) TLWE ciphertext (with publicly verifiable decryption) $\mathfrak{c} = (\mathfrak{a}_1|\dots|\mathfrak{a}_t, \mathfrak{b}) \in \mathbb{T}^{t\times n} \times \mathbb{T}$, each of the t trustees independently uses his master trapdoor R_i to provide a ciphertext trapdoor Π_i of $(\mathfrak{a}_i, 0)$, and like in the previous section, the concatenated ciphertext trapdoor $\Pi = (\Pi_1|\cdots|\Pi_t)$ is a ciphertext-trapdoor for $\mathfrak{c}$.

Observe that, even if all trustees leak their private keys except one of them (we take the first trustee for simplicity), then decrypting $\mathfrak{c}$ rewrites in decrypting the TLWE ciphertext $(\mathfrak{a}_1, \mathfrak{b}')$ where $\mathfrak{b}' = \mathfrak{b} - \sum_{i=2}^{t} \mathfrak{a}_i \cdot \mathfrak{K}_i$. This is by definition still λ-bit secure. In other words, even in case of collusions between the trustees, the whole scheme remains secure as long as one trustee is honest.

B.3 Detailed Description of our E-voting Protocol

We now describe in detail the three phases of the vote: setup, voting and tallying. For every phase, we describe in detail all the different steps, when there is an honest execution of the procedure.
We conclude this section, and also the chapter, by making a simple estimate of the execution costs of this e-voting scheme by using TFHE. The estimates are done only by using the gate bootstrapping. Further improvements could be obtained by applying the leveled techniques and the circuit bootstrapping.
The question deserves a more in-depth study, we left as a future work.

B.3.1 Setup phase

The setup phase is dedicated to the parameter setting and voter registration.

Key generation. Fixed a security parameter λ, the bulletin board manager generates a pair of keys $(\mathsf{pk}_{\mathsf{BB}}, \mathsf{sk}_{\mathsf{BB}}) = \mathsf{KeyGen}_{\mathsf{BB}}(1^\lambda)$ and publishes $\mathsf{pk}_{\mathsf{BB}}$.

The trustees setup the concatenated TLWE scheme presented in previous section: each trustee generates its own separate TLWE secret key $\mathfrak{K}_i \in \mathbb{B}^n$, its own master trapdoor R_i, and a corresponding public key $\mathsf{pk}_i \in \mathbb{T}^{m\times(n+1)}$.
Thus, the secret key sk_i of each trustee consists in R_i and $\mathfrak{K}_i$. Without revealing any information on $\mathfrak{K}_i$, they must provide a proof that the public key pk_i is indeed composed of TLWE samples of 0 encrypted with the corresponding secret. This is in fact a requirement for the correctness of the decryption with ciphertext trapdoors. To do so, we think that the trustees may for instance use NIZK proofs: this passage deserves a deeper study.
Once the existence of $\mathfrak{K}_i$ is established, the trustees do not necessarily need to prove that they know the secrets $\mathfrak{K}_i$ or R_i: the simple fact that they can output valid ciphertext trapdoors proves it anyways, by standard LWE-to-SIS or decision-to-search reduction arguments.
The main public key pk is the concatenated key defined in Equation (B.1).

To perform homomorphic operations efficiently, the trustees define two other TLWE secret keys, $\mathfrak{K}^{(f)}$ (frontend keys) and $\mathfrak{K}^{(m)}$ (middle keys), and their corresponding public key $\mathsf{pk}^{(f)}$ and $\mathsf{pk}^{(m)}$. The frontend keys are used by the voters to encrypt their ballots in the first place. Then the ballots are bootstrapped to produce an encryption of the same vote under the middle key. The initial (frontend) ciphertext is then encrypted with the BB public key to became the auxiliary information. These keys may still use a concatenated scheme, although this time, they don't need a master trapdoor for that. The middle key is used all along the BB computations, until the final bootstrapping performed on each ballot, before computing the final sum of all the votes. The key used in this step is the one using trapdoor material.
Finally, the trustees provide three bootstrapping keys (and eventual key-switching keys), that we note $\mathsf{BK}_1 := \mathsf{BK}_{\mathfrak{K}^{(f)}\to\mathfrak{K}^{(m)}}$, $\mathsf{BK}_2 := \mathsf{BK}_{\mathfrak{K}^{(m)}\to\mathfrak{K}^{(m)}}$, and $\mathsf{BK}_3 := \mathsf{BK}_{\mathfrak{K}^{(m)}\to\mathfrak{K}}$. Since a bootstrapping key essentially consists in a public TLWE encryption of each individual bit of the private key, each trustee can independently provide their part of the bootstrapping keys.
The first two keys can be built by using the same parameter set described for the gate bootstrapping. For the third bootstrapping key, we have two options: the first one consists in using again the same parameter set as in the gate bootstrapping and evaluate the final sum between ballots as a huge addition circuit, gate by gate. The second option, which is the one originally proposed in our paper, consists in using a

larger bootstrapping key in order to have an output message space with low noise amplitude, and then perform the final sum of ballots as a simple addition.
In the last paragraph of this section we make estimates on the execution timings for both solutions.

Voter registration. For every voter that wants to register, the authority $\mathcal{A}_1$ verifies the voter identity id and generates the signature keys $(\mathsf{upk}_{\mathsf{id}}, \mathsf{usk}_{\mathsf{id}}) \leftarrow \mathsf{KeyGenS}(1^\lambda)$. Then $\mathcal{A}_1$ adds $\mathsf{upk}_{\mathsf{id}}$ in $\mathcal{L}_{\mathcal{U}}$, the public list of users, and outputs $(\mathsf{upk}_{\mathsf{id}}, \mathsf{usk}_{\mathsf{id}})$.

B.3.2 Voting phase

In our scheme, we suppose that the number of candidates $\ell = 2^k$ is a power of two. If it is not the case, we can always add null candidates. Then, if we choose a random value for the vote, no candidate will be favorite over the others. A valid vote v is thus assimilated to an integer between 0 and $\ell - 1$.

The ballot. By running $\mathsf{Vote}(\mathsf{pk}, \mathsf{usk}, \mathsf{upk}, v)$, each user computes the binary decomposition $(v_0, \ldots, v_{k-1}) \in \{0,1\}^k$ such that $v = \sum_{j=0}^{k-1} v_j 2^j$. Let $\hat{v}_j$ denote $\frac{1}{2}v_j \in \mathbb{T}$, each user encrypts each bit as $\mathfrak{c}_j^{(f)} = \mathsf{TLWE}^{\mathsf{Pub}}{}_{\mathsf{pk}^{(f)}}(\hat{v}_j)$ with noise amplitude $< \frac{1}{4}$. He then bootstraps every $\mathfrak{c}_j^{(f)}$ by using BK_1 and obtains corresponding bootstrapped ciphertext $\mathfrak{c}_j^{(m)}$.
The auxiliary information is computed as $\mathsf{aux} = \mathsf{Enc}_{\mathsf{BB}}(\mathsf{pk}_{\mathsf{BB}}, (\mathfrak{c}_0^{(f)}, \ldots, \mathfrak{c}_{k-1}^{(f)}) | \mathsf{upk})$.
The final ballot is $b = (\text{content}, \sigma)$, where $\text{content} = (\mathsf{aux}, \mathsf{upk}, (\mathfrak{c}_0^{(m)}, \ldots, \mathfrak{c}_{k-1}^{(m)}), \mathsf{num})$, $\sigma = \mathsf{Sign}(\mathsf{usk}, \text{content})$ and where num is the version number of the ballot for the revote policy[1].

Processing a ballot in BB

Once the voter sends the ballot to the bulletin board, this one does the validity checks and processes it before summing it with the other ballots to obtain the final encrypted result.

Validity checks on a ballot. Upon reception of a ballot b, the bulletin board parses it as $(\text{content}, \sigma)$, with $\text{content} = (\mathsf{aux}, \mathsf{upk}, (\mathfrak{c}_0^{(m)}, \ldots, \mathfrak{c}_{k-1}^{(m)}), \mathsf{num})$. He does:

- Verify that $\mathsf{upk} \in \mathcal{L}_{\mathcal{U}}$;
- Verify the signature σ;

[1] The revote policy consists in accepting the last vote sent for upk: BB accepts to overwrite a ballot for upk if and only if the new version number is strictly larger than the previous one.

- Compute $(\mathfrak{c}_0^{(f)}, \ldots, \mathfrak{c}_{k-1}^{(f)}) | \mathsf{upk}' = \mathsf{Dec}_{\mathsf{BB}}(\mathsf{sk}_{\mathsf{BB}}, \mathsf{aux})$;
- Verify that each $\mathfrak{c}_j^{(f)} \in \mathbb{T}^{n+1}$;
- Verify that each $\mathfrak{c}_j^{(m)} \in \mathbb{T}^{n+1}$;
- Check whether $\mathsf{upk}' = \mathsf{upk}$;
- Check that $\mathfrak{c}_j^{(m)}$ is the output of the bootstrapping with key BK_1 taking in input $\mathfrak{c}_j^{(f)}$, for all $j = 0, \ldots, k-1$;
- Checks the revote policy with the version number num.

If all the validity checks pass, the bulletin board adds the ballot Observe that, unlike classical e-voting protocols, no semantic check or zero-knowledge proof is needed at this step, since all binary message are valid choices.

BB homomorphic operations. BB applies a sequence of public homomorphic operations on the encrypted vote $(\mathfrak{c}_0^{(m)}, \ldots, \mathfrak{c}_{k-1}^{(m)})$. These homomorphic operations do not require the presence of the voter, and can therefore be performed offline by the cloud. To simplify, we just describe what happens on plaintexts.

1. *Pre-bootstrapping* A pre-bootstrapping by using BK_2 is applied on each $\mathfrak{c}_j^{(m)}$ to cancel its uncontrolled input noise and reduce it to $\frac{1}{16}$, and also to re-express its content on the $\{0, \frac{1}{4}\}$ message space, which is suitable for Boolean homomorphic operations.

2. *Homomorphic binary expansion* In order to compute the sum of the votes (homomorphically), BB transforms the vector $\hat{\mathbf{v}} = \frac{1}{4}\mathbf{v} = \frac{1}{4}(v_0, \ldots, v_{k-1}) \in \{0, \frac{1}{4}\}^k$ into its characteristic vector $\hat{\mathbf{w}} = \frac{1}{4}(w_0, \ldots, w_{\ell-1}) = (0, \ldots, 0, \frac{1}{4}, 0, \ldots, 0)$ of length ℓ, the number of the candidates (or choices for the vote), with a $\frac{1}{4}$ at position $v = \sum_{i=0}^{k-1} v_i 2^i$. This transformation is easy to compute: the value of every w_h, for $h = 0, \ldots, \ell - 1$, depends on all the bits of $\mathbf{v}$. For every element h having binary decomposition $h = \sum_{i=0}^{k-1} h_i 2^i$, w_h corresponds to this boolean term:

$$w_h(\mathbf{v}) = \left(\bigwedge_{\substack{i \in [0,k-1] \\ h_i = 0}} \overline{v_i} \right) \wedge \left(\bigwedge_{\substack{i \in [0,k-1] \\ h_i = 1}} v_i \right)$$

The formula seems complicated, but it is just a conjunction of k variables v_i or their negation ($k = \log_2 \ell$ is in general smaller than 5 in typical elections).
These conjunctions can be easily evaluated on ciphertexts using the bootstrapped HomNOT and HomAND gates, described in Chapter 5. The total number of bootstrapped homomorphic gates is $\ell \cdot k$.

3. *Final bootstrapping.* BB uses the main bootstrapping key BK_3 to convert these ℓ ciphertexts into a new ciphertext of $(0, \ldots, 0, \frac{1}{L}, 0, \ldots, 0)$ with noise $O(L^{-3/2})$, where L is an upper bound on the number of voters. The choice of parameters have to be done considering this quantity. The bootstrapping at level 2 used for the circuit bootstrapping, and implemented in the experimental repository of TFHE could be adapted for this purpose.

4. *Homomorphic addition.* At the end of the voting phase, BB sums (homomorphically) all ciphertexts, which yields to the final TLWE ciphertexts $(C_0, \ldots, C_{\ell-1})$ of $(\frac{n_0}{L}, ..., \frac{n_{\ell-1}}{L})$, with noise $O(L^{-1})$. No bootstrapping is needed for this step, it just uses the standard addition on ciphertexts.

A different solution consists in defining BK_3 with the same parameters as the gate bootstrapping: the homomorphic addition corresponds to the evaluation of a circuit composed by homomorphic bootstrapped gates.

B.3.3 Tallying phase

Denote as $(C_0, \ldots, C_{\ell-1})$ the final ciphertext processed by BB. Each TLWE sample C_j encodes the message $\frac{n_j}{L}$ with noise amplitude $O(1/L)$, where n_j is the number of votes for candidate j.

Tally. For each C_j, the trustees independently perform the distributed decryption described in section B.2.2, and publish a ciphertext trapdoor $\Pi_{i,j} \in \mathbb{Z}^m$ (for $i = 1, \ldots, t$ and $j = 0, \ldots, \ell - 1$) as in definition B.2.1.

Verify tally. Given the main public key pk, anyone is able to check the validity of the PVCTs. If a trapdoor $\Pi_{i,j}$ is invalid, it publicly proves that the i-th trustee is not honest and in this case VerifyTally returns $\perp$. If all the trapdoors are valid, anyone can use $(\Pi_{1,j}, \ldots, \Pi_{t,j})$ to decrypt C_j, and thus, recover n_j for all $j = 0, \ldots, \ell - 1$, which gives the number of votes for the candidate j. This gives the result of the election. And VerifyTally returns the result $(n_0, \ldots, n_{\ell-1})$.

The entire procedure is summarized in Figure B.3.

B.4 Practical estimates

The scheme has never been implemented, but we can give an approximate prediction of the execution timings of our e-voting procedure by using the TFHE library. Real timings and more precise parameters deserve a more in-depth study, we left as a future work.

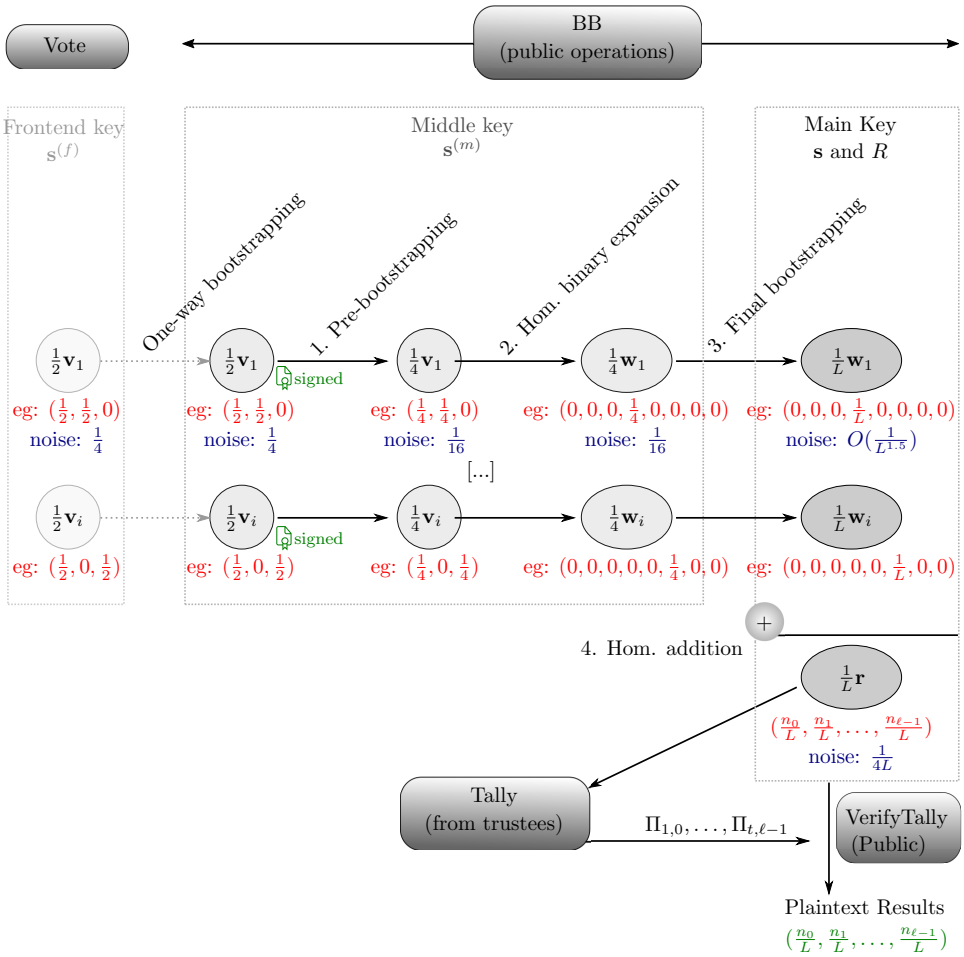

Figure B.3: Schematic of the protocol - *The entire e-voting procedure is schematized in this figure. In red and green an example: red means encrypted, green means decrypted.*

It is quite natural to suppose that the homomorphic operations involving bootstrapping are the most costly part. It is for this reason that we do not take into account other operations. Such homomorphic operations are performed only by the voters and mainly by the bulletin board.

We imagine the scenario of an election where the number of candidates is upper bounded by $\ell = 32 = 2^5$, where 2 trustees are involved, and where the number of voters is 1.000, 10.000, 100.000 and 1.000.000.

As we use concatenated TLWE encryption with two trustees, we may upper bound the size of the TLWE samples to $4KBytes$ and the double the timing for a gate bootstrapping execution, i.e. $26ms$.

Voters. Each voter generates his ballot: the voter has to encrypt only 5 bits of information, corresponding to his vote, with a TLWE encryption, and bootstrap it. The amount of memory required by the TLWE encryptions is about $40KBytes$.
The 5 gate bootstrappings he has to perform, are executed in about 2 seconds (single-core) on a common laptop.

Bulletin board. The bulletin board has to process every ballot separately, before adding them homomorphically. A pre-bootstrapping to verify the auxiliary operation, an initial bootstrapping to reduce the uncontrolled noise and the homomorphic expansion, need a total of 170 gate bootstrappings, i.e. about 4.5 seconds.

The final bootstrapping and the addition can be performed in two ways: by bootstrapping to a lower level of noise and performing a simple leveled homomorphic addition between ballots, or by evaluating the gate bootstrapping addition circuit.
In the first solution, a bootstrapping with level 2 parameters (as in the circuit bootstrapping) should be sufficient to perform the homomorphic additions. We could upper bound the cost of the final bootstrapping of each sample to approximately $91ms$. The leveled homomorphic addition between all the ballots has negligible execution timing compared to the bootstrapping.
The second option consists in using the gate bootstrapping evaluation for the addition circuit: it can be represented as a full adder circuit with bootstrapped homomorphic gates.

The following table summarizes the approximate execution timings (for final bootstrapping and homomorphic addition) for both techniques. The timings correspond to the computation of the final encrypted result per candidate with single core execution. Again, the timings are approximate theoretical predictions: a practical implementation should be done to verify them in practice. The timings we propose seem to confirm our original prediction, stating that a bootstrapping to a lower level of noise with leveled evaluation of the addition circuit is the best solution.

	Gate bootstr. addition		**Leveled addition**
# Voters	# Gates	Timing	Timing
1.000	$\approx$ 6800	$\approx$ 3 min	$\approx$ 1 min, 30 sec
10.000	$\approx$ 66900	$\approx$ 29 min	$\approx$ 16 min
100.000	$\approx$ 667200	$\approx$ 4 hours, 50 min	$\approx$ 2 hours, 30 min
1.000.000	$\approx$ 6667300	$\approx$ 2 days	$\approx$ 25 hours, 22 min

Appendix C

A different cloud solution: MPC

Fully homomorphic encryption is not the only secure private computing solution. Many other methods deserve to be studied and one of them is *Multi-Party Computation* (MPC): many parties compute the common result of a function by using private inputs they do not want to share with the others.
In MPC there are no keys: the data is not encrypted, just securely masked, and the final common result is in clear, contrarily to FHE.

In this chapter we present the results of a paper we wrote on this subject [BCG+18], recently accepted at the conference Financial Cryptography and Data Security 2018. The main contribution of the paper is a new MPC protocol that can be used to evaluate real valued functions with high numerical precision. This technique is mainly useful to solve classification problems aiming to detect rare events. The MPC computations include an offline phase, for which we model two security scenarios, and a short online phase, needing at most two rounds of communication to compute the result. The protocol has been implemented and multiple experiments shown that our method is efficient.

C.1 Overview of the work

Privacy-preserving computing allows multiple parties to evaluate a function while keeping the inputs private and revealing only the output of the function and nothing else. Recent advances in MPC, homomorphic encryption, and differential privacy made these models practical. An example of such computations, with applications in medicine and finance, among others, is the training of supervised models where the input data comes from distinct secret data sources [GSB+17], [LP00], [MZ17], [NWI+13] and the evaluation of predictions using these models.
In machine learning *classification problems*, one trains a model on a given dataset to predict new inputs, by mapping them into discrete categories. The classical *logistic regression* model predicts a class by providing a probability associated with the

prediction. The quality of the model can be measured in several ways, the most common one being the *accuracy* that indicates the percentage of correctly predicted answers.
It appears that for a majority of the datasets (e.g., the MNIST database [Datb] of handwritten digits, or the ARCENE [Data] mass-spectrometry dataset for cancer detection), the classification achieves very good accuracy after only a few iterations of the gradient descent using a piecewise-linear approximation of the sigmoid function $\texttt{sigmo} : \mathbb{R} \to [0, 1]$ defined as

$$\texttt{sigmo}(x) = \frac{1}{1 + e^{-x}},$$

although the current cost function is still far from the minimum value [MZ17]. Other approximation methods of the sigmoid function have also been proposed in the past. In [WTK+13], an approximation with low degree polynomials resulted in a more efficient but less accurate algorithm. Conversely, a higher-degree polynomial approximation applied to deep learning algorithms in [LSS14] yielded more accurate, but less efficient algorithms (and thus, less suitable for privacy-preserving computing).
In parallel, approximation solutions for privacy-preserving methods based on homomorphic encryption [AHPW16], [PAH+17], [GDL+16], [JA16] and differential privacy [ACG+16], [CM08] have been proposed in the context of both classification algorithms and deep learning.

Nevertheless, accuracy itself is not always a sufficient measure for the quality of the model, especially if, as mentioned in [GBC16, Page 423], our goal is to detect a rare event such as a rare disease or a fraudulent financial transaction. If, for example, one out of every one thousand transactions is fraudulent, a naïve model that classifies all transactions as honest achieves 99.9% accuracy; yet this model has no predictive capability. In such cases, measures such as *precision*, *recall* and *F1-score* allow for better estimating the quality of the model. They bound the rates of false positives or negatives relative to only the positive events rather than the whole dataset. Precision is the fraction of true positives, i.e. the number of positive elements that are labeled as such, among all the elements that are classified as positive by the model. Instead, recall is the fraction of true positives among all the positive elements. F1-score is the harmonic mean of previous measures.
The techniques cited above achieve excellent accuracy for most balanced datasets, but since they rely on a rough approximation of the sigmoid function, they do not converge to the same model and thus, they provide poor scores on datasets with a very low acceptance rate.

In this chapter, we show how to regain this numerical precision in MPC, and to reach the same score as the plaintext regression. Our MPC approach is mostly based on additive secret shares with precomputed multiplication triples [Bea91]. This means

that the computation is divided in two phases: an *offline* phase that can be executed before the data is shared between the players, and an *online phase* that computes the actual result. For the offline phase, we propose a first solution based on a *trusted dealer*, and then discuss a protocol where the dealer is *honest-but-curious*.

Fourier approximation of the sigmoid function. Evaluation of real-valued functions has been widely used in privacy-preserving computations. For instance, in order to train linear and logistic regression models, one is required to compute real-valued functions such as the square root, the exponential, the logarithm, the sigmoid or the softmax function and use them to solve non-linear optimization problems. In order to train a logistic regression model, one needs to minimize a cost function which is expressed in terms of logarithms of the continuous sigmoid function. This minimum is typically computed via iterative methods such as the gradient descent. For datasets with low acceptance rate, it is important to get much closer to the exact minimum in order to obtain a sufficiently precise model. We thus need to significantly increase the number of iterations (naïve or stochastic gradient descent) or use faster-converging methods (e.g., IRLS [Bjö96, Section 4.3]). The latter require a numerical approximation of the sigmoid that is much better than what was previously achieved in an MPC context, especially when the input data is not normalized or feature-scaled[1]. Different approaches have been considered previously such as approximation by Taylor series around a point (yielding only good approximation locally at that point), or polynomial approximation (by e.g., estimating least squares). Although better than the first one, this method is numerically unstable due to the variation of the size of the coefficients. An alternative method based on approximation by piecewise-linear functions has been considered as well. In MPC, this method performs well when used with garbled circuits instead of secret sharing and masking, but does not provide enough accuracy.

In our case, we approximate the sigmoid using Fourier series, an approach applied for the first time in this context. This method works well as it provides a better uniform approximation assuming that the function is sufficiently smooth (as is the case with the sigmoid). In particular, we virtually re-scale and extend the sigmoid to a periodic function that we approximate with a trigonometric polynomial which we then evaluate in a stable privacy-preserving manner. To approximate a generic function with trigonometric polynomials that can be evaluated in MPC, one either uses the Fourier series of a smooth periodic extension or finds directly the closest trigonometric polynomial by the method of least squares for the distance on the half-period. The first approach yields a super-algebraic convergence at best, whereas the second converges exponentially fast. On the other hand, the first one is numerically

[1]Feature scaling is a method used to bring all the features to a similar scale. If data are not feature scaled, it could be sometimes complicated to rapidly converge to a global minimum of the cost function.

stable whereas the second one is not (under the standard Fourier basis). In the case of the sigmoid, we show that one can achieve both properties at the same time.

Floating-point representation and masking. A typical approach to multiparty computation protocols with masking is to embed fixed-point values into finite groups and use uniform masking and secret sharing. Arithmetic circuits can then be evaluated using, e.g., precomputed multiplication triples and following Beaver's method [Bea91]. This idea has been successfully used in SPDZ [DPSZ, DPSZ12]. Whereas the method works well on low multiplicative depth circuits like correlations or linear regression [GSB+17], in general, the required group size increases exponentially with the multiplicative depth. In [MZ17], this exponential growth is mitigated by a two-party rounding solution, but the technique does not extend to three or more players where an overflow in the most significant bits can occur.

In this work, we introduce an alternative sharing scheme, where fixed-point values are shared directly using (possibly multibit) floating points, and present a technique to reduce the share sizes after each multiplication. This technique easily extends to an arbitrary number of players.

Significant reduction in communication time. We follow the same approach as in [MZ17] and define dedicated triples for high-level instructions, such as large matrix multiplications, a system resolution, or an oblivious evaluation of the sigmoid. This approach is less generic than masking low-level instructions as in SPDZ, but it allows to reduce the communication and memory requirements by large factors.
Masks and operations are aware of the type of vector or matrix dimensions and benefit from the vectorial nature of the high-level operations. For example, multiplying two matrices requires a single round of communication instead of up to $O(n^3)$ for coefficient-wise approaches, depending on the batching quality of the compiler. Furthermore, masking is defined per immutable variable rather than per elementary operation, so a constant matrix is masked only once during the whole algorithm. Combined with non-trivial local operations, these triples can be used to achieve much more than just ring additions or multiplications. In a nutshell, the amount of communications is reduced as a consequence of reusing the same masks, and the number of communication rounds is reduced as a consequence of masking directly matrices and other large structures. Therefore, the total communication time becomes negligible compared to the computing cost.

New protocol for the honest but curious offline phase extendable to n players. We introduce a new protocol for executing the offline phase in the honest-but-curious model that is easily extendable to a generic number n of players while remaining efficient. To achieve this, we use a broadcast channel instead of peer-to-peer communication which avoids a quadratic explosion in the number of commu-

nications. This is an important contribution, as none of the previous protocols for $n > 3$ players in this model are efficient. In [GSB+17], for instance, the authors propose a very efficient algorithm in the trusted dealer model. Yet the execution time of the oblivious transfer protocol is quite slow.

C.2 Secret sharing and MPC: a short background

Assume that $P_1, \ldots, P_n$ are distinct computing parties (players). We recall some basic concepts from multi-party computation that will be needed to better understand this chapter.

C.2.1 Secret sharing and masking

Let $(G, \bullet)$ be a group and let $x \in G$ be a group element. A *secret share* of x, denoted by $[\![x]\!]_\bullet$, is a tuple $(x_1, \ldots, x_n) \in G^n$ such that $x = x_1 \bullet \cdots \bullet x_n$. If $(G, +)$ is abelian, we call the secret shares $x_1, \ldots, x_n$ *additive secret shares*.
A secret sharing scheme is computationally secure if for any two elements $x, y \in G$, strict sub-tuples of shares $[\![x]\!]_\bullet$ or $[\![y]\!]_\bullet$ are indistinguishable. If G admits a uniform distribution, an information-theoretic secure secret sharing scheme consists of drawing $x_1, \ldots, x_{n-1}$ uniformly at random and choosing $x_n = x_{n-1}^{-1} \bullet \cdots \bullet x_1^{-1} \bullet x$. When G is not compact, the condition can be relaxed to statistical or computational indistinguishability.
A closely related notion is the one of *group masking*. Given a subset $\mathcal{X}$ of G, the goal of masking $\mathcal{X}$ is to find a distribution $\mathcal{D}$ over G such that the distributions of $x \bullet \mathcal{D}$ for $x \in \mathcal{X}$ are all indistinguishable. Indeed, such distribution can be used to create a secret share: one can sample $\lambda \leftarrow \mathcal{D}$, and give λ^{-1} to a player and $x \bullet \lambda$ to the other. Masking can also be used to evaluate non-linear operations in clear over masked data, as soon as the result can be privately unmasked via homomorphisms, as in the Beaver's triple multiplication technique [Bea91].

C.2.2 Arithmetic with secret shares via masking

Computing secret shares for a sum $x + y$, or a linear combination if $(G, +)$ has a module structure, can be done non-interactively by each player by adding the corresponding shares of x and y.
Computing secret shares for a product is more challenging. One way to do that is to use Beaver's idea, based on precomputed and secret shared multiplicative triples. From a general point of view, let $(G_1, +)$, $(G_2, +)$ and $(G_3, +)$ be three abelian groups and let $\pi \colon G_1 \times G_2 \to G_3$ be a bilinear map. Given additive secret shares $[\![x]\!]_+$ and $[\![y]\!]_+$ for two elements $x \in G_1$ and $y \in G_2$, we would like to compute secret shares for the element $\pi(x, y) \in G_3$. With Beaver's method, the players must employ precomputed single-use random triples $([\![\lambda]\!]_+, [\![\mu]\!]_+, [\![\pi(\lambda, \mu)]\!]_+)$ for $\lambda \in G_1$

and $\mu \in G_2$, and then use them to mask and reveal $a = x + \lambda$ and $b = y + \mu$. The players then compute secret shares for $\pi(x, y)$ as follows:

- Player 1 computes $z_1 = \pi(a, b) - \pi(a, \mu_1) - \pi(\lambda_1, b) + (\pi(\lambda, \mu))_1$;
- Player i (for $i = 2, \ldots, n$) computes $z_i = -\pi(a, \mu_i) - \pi(\lambda_i, b) + (\pi(\lambda, \mu))_i$.

The computed $z_1, \ldots, z_n$ are the additive shares of $\pi(x, y)$. A given λ (or μ respectively) can be used to mask only one variable, so one triple must be precomputed for each multiplication during the *offline phase*, i.e. before the data is made available to the players. Instantiated with the appropriate groups, this abstract scheme allows to evaluate a product in a ring, but also a vector dot product, a matrix-vector product, or a matrix-matrix product.

C.2.3 MPC evaluation of real-valued continuous functions

For various applications, such as logistic regression, we need to compute continuous real-valued functions over secret shared data. For non-linear functions (e.g. exponential, log, power, cos, sin, sigmoid, etc.), different methods are proposed in the literature.

A straightforward approach consists of implementing a full floating point arithmetic framework, and to compile a data-oblivious algorithm that evaluates the function over floats. This is for instance what Sharemind [BLW08] and SPDZ [DPSZ, DPSZ12] use. However, these two generic methods lead to prohibitive running times if the floating point function has to be evaluated millions of times.

The second approach is to replace the function with an approximation that is easier to compute. As instance, [MZ17] uses garbled circuits to evaluate fixed point comparisons and absolute values, then it replaces the sigmoid function in the logistic regression with a piecewise-linear function. Otherwise, [LSS14] approximates the sigmoid with a polynomial of fixed degree and evaluates that polynomial with Horner's method, thus requiring a number of rounds of communications proportional to the degree.

Another method, that is close to how SPDZ computes inverses in a finite field, is based on polynomial evaluation via multiplicative masking: using a precomputed triple of the form $([\![\lambda]\!]_+, [\![\lambda^{-1}]\!]_+, \ldots, [\![\lambda^{-p}]\!]_+)$, players can evaluate $P(x) = \sum_{i=0}^{p} a_p x^p$ by revealing $u = x\lambda$ and outputting the linear combination $\sum_{i=0}^{p} a_i u^i [\![\lambda^{-i}]\!]_+$.

Multiplicative masking, however, involves some leakage: in finite fields, it reveals whether x is null. The situation gets even worse in finite rings where the multiplicative orbit of x is disclosed (for instance, the rank would be revealed in a ring of matrices), and over $\mathbb{R}$, the order of magnitude of x would be revealed.

For real-valued polynomials, the leakage could be mitigated by translating and rescaling the variable x so that it falls in the range $[1, 2)$. Yet, in general, the coefficients of the polynomials that approximate the translated function explode, thus causing serious numerical issues.

C.2.4 Full threshold honest-but-curious protocol

Since our goal is to emphasize new functionalities, such as efficient evaluation of real-valued continuous functions and good quality logistic regression, we often consider a scenario where all players follow the protocol without introducing any errors. The players may, however, record the whole transaction history and try to learn illegitimate information about the data. During the online phase, the security model imposes that any collusion of at most $n-1$ players out of n cannot distinguish any semantic property of the data beyond the aggregated result that is legitimately and explicitly revealed.
To achieve this, Beaver triples can be generated and distributed by a single entity called the *trusted dealer*. In this case, no coalition of at most $n-1$ players should get any computational advantage on the plaintext triple information. However, the dealer himself knows the plaintext triples, and hence the whole data, which only makes sense on some computation outsourcing use-cases.
It is for this reason that we give an alternative *honest-but-curious* (or semi-honest) protocol to generate the same triples, involving this time bi-directional communications between the players and the dealer. In this case, the dealer and the players collaborate during the offline phase in order to generate the precomputed material, but none of them have access to the whole plaintext triples. This makes sense as long as the dealer does not collude with any player, and at least one player does not collude with the other players. We leave the design of actively secure protocols for future work.

C.3 Statistical Masking and Secret Share Reduction

In this section, we present our masking technique for fixed-point arithmetic and provide an algorithm for the MPC evaluation of real-valued continuous functions. In particular, we show that to achieve p bits of numerical precision in MPC, it suffices to have $p+2\tau$-bit floating points where τ is a fixed security parameter.

Our secret shares are real numbers and we would like to mask these shares using floating point numbers. Yet, as there is no uniform distribution on $\mathbb{R}$, no additive masking distribution over reals can perfectly hide the arbitrary inputs. In the case when the secret shares belong to some known range of numerical precision, it is possible to carefully choose a masking distribution, depending on the precision range, so that the masked value computationally leaks no information about the input. A distribution with sufficiently large standard deviation could do the job: we refer to this type of masking as "statistical masking". In practice, we choose a normal distribution with standard deviation $\sigma = 2^{40}$.

On the other hand, by using such masking, we observe that the sizes of the secret shares increase every time we evaluate the multiplication via Beaver's technique. We take care of this problem by introducing a technique that allows to reduce the secret share sizes by discarding the most significant bits of each secret share. This is possible thanks to the fact that the sum of the secret shares is still much smaller than their size.

C.3.1 Fixed point, floating point and interval precision

Let B be an integer and p be a non negative integer. We define the class of fixed point numbers of exponent B and numerical precision p as

$$\mathcal{C}(B,p) = \{x \in 2^{B-p}\mathbb{Z} \mid |x| \leq 2^B\}.$$

Each class $\mathcal{C}(B,p)$ is finite, and contains $2^{p+1}+1$ numbers. They could be rescaled and stored as $(p+2)$-bit integers.
Alternatively, a number $x \in \mathcal{C}(B,p)$ can also be represented by its floating point value, provided that the floating point representation has at least p bits of mantissa. In this case, addition and multiplication of numbers across classes of the same numerical precision are natively mapped to floating point arithmetic. The main arithmetic operations on these classes are:

- **Lossless Addition:** $\mathcal{C}(B_1,p_1) \times \mathcal{C}(B_2,p_2) \to \mathcal{C}(B,p)$ where $B = max(B_1,B_2)+1$ and $p = B - \min(B_1-p_1, B_2-p_2)$;
- **Lossless Multiplication:** $\mathcal{C}(B_1,p_1) \times \mathcal{C}(B_2,p_2) \to \mathcal{C}(B,p)$ where $B = B_1+B_2$ and $p = p_1+p_2$;
- **Rounding:** $\mathcal{C}(B_1,p_1) \to \mathcal{C}(B,p)$, that maps x to its nearest element in $2^{B-p}\mathbb{Z}$.

Lossless operations require p to increase exponentially in the multiplication depth, whereas fixed precision operations maintain p constant by applying a final rounding. Finally, note that the exponent B should be incremented to store the result of an addition. B is a user-defined parameter in fixed point arithmetic: if the user chooses to keep B unchanged, any result $|x| > 2^B$ will not be representable in the output domain. In this case, we say that there is a *plaintext overflow*.

C.3.2 Floating point representation

Given a security parameter τ, we say that a set $\mathcal{S}$ is a τ-secure *masking set* for a class $\mathcal{C}(B,p)$ if the following distinguishability game cannot be won with advantage $\geq 2^{-\tau}$.

The adversary chooses two plaintexts m_0, m_1 in $\mathcal{C}(B,p)$, a challenger picks $b \in \{0,1\}$ and $\alpha \in \mathcal{S}$ uniformly at random, and sends $c = m_b + \alpha$ to the adversary. The

adversary has to guess b.

Note that increasing such distinguishing advantage from $2^{-\tau}$ to $\approx 1/2$ would require to give at least 2^τ samples to the attacker, so $\tau = 40$ is sufficient in practice.

Lemma C.3.1. *The class* $\mathcal{C}(B, p, \tau) = \{\alpha \in 2^{B-p}\mathbb{Z} \mid |\alpha| \leq 2^{B+\tau}\}$ *is a* τ*-secure masking set for* $\mathcal{C}(B, p)$

Proof. Observe that the number of elements in $\mathcal{C}(B, p, \tau)$ is $2^{p+\tau+1} + 1$. If $a, b \in \mathcal{C}(B, p)$ and $\mathcal{U}$ is the uniform distribution on $\mathcal{C}(B, p, \tau)$, the statistical distance between $a+\mathcal{U}$ and $b+\mathcal{U}$ is $(b-a)\cdot 2^{p-B}/\#\mathcal{C}(B, p, \tau) \leq 2^{-\tau}$. This distance upper-bounds any computational advantage. □

As the class $\mathcal{C}(B, p, \tau) = \mathcal{C}(B+\tau, p+\tau)$ fits in floating point numbers of $(p+\tau+1)$-bits of mantissa, they can be used to securely mask fixed point numbers with numerical precision p. By extension, all additive shares for $\mathcal{C}(B, p)$ will be taken in $\mathcal{C}(B, p, \tau)$.

We now analyze what happens if we use Beaver's protocol to multiply two plaintexts $x \in \mathcal{C}(B_1, p)$ and $y \in \mathcal{C}(B_2, p)$. The masked values $x + \lambda$ and $y + \mu$ are bounded by $2^{B_1+\tau}$ and $2^{B_2+\tau}$ respectively. Since the mask λ is also bounded by $2^{B_1+\tau}$, and μ by $2^{B_2+\tau}$, the computed secret shares of $x \cdot y$ will be bounded by $2^{B_1+B_2+2\tau}$.
So the lossless multiplication sends $\mathcal{C}(B_1, p, \tau) \times \mathcal{C}(B_2, p, \tau) \to \mathcal{C}(B, 2p, 2\tau)$ where $B = B_1 + B_2$, instead of $\mathcal{C}(B, p, \tau)$. Reducing p is just a matter of rounding, and it is done automatically by the floating point representation. However, we still need a method to reduce τ, so that the output secret shares are bounded by $2^{B+\tau}$. We describe our method in the next section.

C.3.3 Secret share reduction algorithm

The algorithm we propose depends on two auxiliary parameters:

- The *cutoff*, defined as $\eta = B + \tau$ so that 2^η is the desired bound in absolute value;
- An auxiliary parameter $M = 2^\kappa$, for a small number κ we will specify below, larger than the number of players.

The main idea is that the initial share contains large components $z_1, \ldots, z_n$ that sum up to the small secret shared value z. Additionally, the most significant bits of the share beyond the cutoff position (say $\mathrm{MSB}(z_i) = \lfloor z_i/2^\eta \rceil$) do not contain any information on the data, and are all safe to reveal.
We also know that the MSB of the sum of the shares is null, so the sum of the MSB of the shares is very small. The share reduction algorithm simply computes

this sum, and redistributes it evenly among the players. Since the sum is guaranteed to be small, the computation is done modulo M rather than on large integers.

More precisely, using the cutoff parameter η, for $i = 1, \ldots, n$, player i writes his secret share z_i of z as $z_i = u_i + 2^\eta v_i$, with $v_i \in \mathbb{Z}$ and $u_i \in [-2^{\eta-1}, 2^{\eta-1})$. Then, he broadcasts $v_i \mod M$, so that each player computes the sum. The individual shares can optionally be re-randomized using a precomputed share $[\![\nu]\!]_+$, with $\nu = 0 \mod M$. Since $\sum v_i$ is guaranteed to be between $-M/2$ and $M/2$, it can be recovered from its representation mod M.
Thus, each player locally updates its share as $u_i + 2^\eta(\sum v_i)/n$, which have by construction the same sum as the original shares, but are bounded by 2^η.

Algorithm 14 details our method for reducing the size of the secret shares. This procedure is used inside the classical MPC multiplication involving floating points.

Algorithm 14 Mask reduction.

Input: $[\![z]\!]_+$ and one triple $[\![\nu]\!]_+$, with $\nu = 0 \mod M$.
Output: Secret shares for the same value z with smaller absolute values of the shares.

1: Each player P_i computes $u_i \in [-2^{\eta-1}, 2^{\eta-1})$ and $v_i \in \mathbb{Z}$, such that $z_i = u_i + 2^\eta v_i$.
2: Each player P_i broadcasts $v_i + \nu_i \mod M$ to other players.
3: The players compute $w = \frac{1}{n}(\sum_{i=1}^{n}(v_i + \nu_i) \mod M)$.
4: Each player P_i computes the new share of z as $z'_i = u_i + 2^\eta w$

C.4 Fourier Approximation

In order to regain numerical precision in the MPC evaluation of real valued functions, we decided to rely on Fourier theory, which allows to approximate certain periodic functions with trigonometric polynomials.

In this section, we show how to evaluate trigonometric polynomials in MPC. Then we describe our approximation technique for the evaluation of the sigmoid function.

C.4.1 Evaluation of trigonometric polynomials

Recall that a complex trigonometric polynomial is a finite sum of the form $t(x) = \sum_{m=-P}^{P} c_m e^{imx}$ where $c_m \in \mathbb{C}$ is equal to $a_m + ib_m$, with $a_m, b_m \in \mathbb{R}$. Each trigonometric polynomial is a periodic function with period 2π. If $c_{-m} = \overline{c_m}$ for all $m \in \mathbb{Z}$, then t is real-valued, and corresponds to the more familiar cosine decomposition $t(x) = a_0 + 2\sum_{m=1}^{P}(a_m \cos(mx) - b_m \sin(mx))$.

Here, we describe how to evaluate trigonometric polynomials in an MPC context, and explain why it is better than regular polynomials.

We suppose that, for all m, the coefficients a_m and b_m of t are publicly accessible and they are $0 \leq a_m, b_m \leq 1$. As t is 2π periodic, we can evaluate it on inputs modulo 2π. Remark that as $\mathbb{R} \mod 2\pi$ admits a uniform distribution, we can use a uniform masking: this method completely fixes the leakage issues that were related to the evaluation of classical polynomials via multiplicative masking. On the other hand, the output of the evaluation is still in $\mathbb{R}$: in this case we continue using the statistical masking described in previous sections.
The inputs are secretly shared and additively masked: for sake of clarity, to distinguish the classical addition over reals from the addition modulo 2π, we temporarily denote this latter by $\oplus$. In the same way, we denote the additive secret shares with respect to the addition modulo 2π by $[\![\cdot]\!]_\oplus$. Then, the transition from $[\![\cdot]\!]_+$ to $[\![\cdot]\!]_\oplus$ can be achieved by trivially reducing the shares modulo 2π.
Then, a way to evaluate t on a secret shared input $[\![x]\!]_+ = (x_1, \ldots, x_n)$ is to convert $[\![x]\!]_+$ to $[\![x]\!]_\oplus$ and additively mask it with a shared masking $[\![\lambda]\!]_\oplus$, then reveal $x \oplus \lambda$ and rewrite our target $[\![e^{imx}]\!]_+$ as $e^{im(x\oplus\lambda)} \cdot [\![e^{im(-\lambda)}]\!]_+$. Indeed, since $x \oplus \lambda$ is revealed, the coefficient $e^{im(x\oplus\lambda)}$ can be computed in clear. Overall, the whole trigonometric polynomial t can be evaluated in a single round of communication, given a precomputed triple such as $([\![\lambda]\!]_\oplus, [\![e^{-i\lambda}]\!]_+, \ldots, [\![e^{-i\lambda P}]\!]_+)$, that we call Fourier triple in the following, and thanks to the fact that $x \oplus \lambda$ has been revealed.
Also, we notice that to work with complex numbers of absolute value 1 makes the method numerically stable, compared to power functions in regular polynomials. It is for this reason that the evaluation of trigonometric polynomials is a better solution in our context.

C.4.2 Approximating the sigmoid function

If one is interested in uniformly approximating a non-periodic function f, with trigonometric polynomials on a given interval, e.g. $[-\pi/2, \pi/2]$, one cannot simply use the Fourier coefficients. Indeed, even if the function is analytic, its Fourier series need not converge uniformly near the end-points.
A way to remedy to this problem is to look for a *periodic extension* of the function to a larger interval and look at the convergence properties of the Fourier series for that extension. An alternative approach is to use the *least-square approximations*, i.e. to search for this non-periodic function on a larger interval (say $[-\pi, \pi]$), such that the restriction to the original interval of the L^2-distance between the original function and the approximation is minimized. The first technique is interesting because numerically stable, the second one instead converges exponentially fast.
In this section we only focus on how we approximate the sigmoid function. In the case of the sigmoid, we obtain numerical stability and fast convergence at the same

time. We just give the idea of our technique: for more details on the subject and on the techniques used to approximate a non-periodic function, we refer to our paper [BCG+18].

Sigmoid function. We restrict to the case of the sigmoid function over the interval $[-B/2, B/2]$ for some $B > 0$. We can rescale the variable to approximate $g(x) = \texttt{sigmo}(Bx/\pi)$ over $[-\pi/2, \pi/2]$. If we extend g by anti-periodicity (odd-even) to the interval $[\pi/2, 3\pi/2]$ with the mirror condition $g(x) = g(\pi - x)$, we obtain a continuous 2π-periodic piecewise C^1 function. By Dirichlet's global theorem, the Fourier serie of g converges uniformly over $\mathbb{R}$, so for all $\varepsilon > 0$, there exists a degree N and a trigonometric polynomial g_N such that $\|g_N - g\|_\infty \leq \varepsilon$.

To compute $\texttt{sigmo}(t)$ over secret shared t, we first apply the affine change of variable (which is easy to evaluate in MPC), to get the corresponding $x \in [-\pi/2, \pi/2]$. Then, we evaluate the trigonometric polynomial $g_N(x)$ using a Fourier triple. This method is sufficient to get 24 bits of precision with a polynomial of only 10 terms,
However asymptotically, the convergence rate is only in $\Theta(N^{-2})$ due to discontinuities in the derivative of g. In other words, approximating g with λ bits of precision requires to evaluate a trigonometric polynomial of degree $2^{\lambda/2}$. Luckily, in the special case of the sigmoid function, we can make this degree polynomial by explicitly constructing a 2π-periodic analytic function that is exponentially close to the rescaled sigmoid on the whole interval $[-\pi, \pi]$ (not the half interval). Besides, the geometric decay of the coefficients of the trigonometric polynomial ensures perfect numerical stability.

C.5 Honest but curious model

In the previous sections, we defined the shares of multiplication, power and Fourier triples, but we did not explain how to generate them. Of course, a single *trusted dealer* (TD) approved by all players could generate and distribute all the necessary shares to the players. Since the trusted dealer knows all the masks, and thus all the data, the TD model is only legitimate for few computation outsourcing scenarios.

We now explain how to generate the same triples efficiently in the more traditional *honest but curious* (HBC) model. To do so, we keep an external entity, called again the dealer, who participates in an interactive protocol to generate the triples, but sees only masked information. Since the triples in both the HBC and TD models are similar, the online phase is unchanged. Notice that in this HBC model, even if the dealer does not have access to the secret shares, he still has more power than the players. In fact, if one of the players wants to gain information on the secret data, he has to collude with all other players, whereas the dealer would need to

collaborate with just one of them.

In what follows, we suppose that, during the offline phase, a private channel exists between each player and the dealer. In the case of an HBC dealer, we also assume that an additional private broadcast channel (a channel to which the dealer has no access) exists between all the players. Figure C.1 summarizes the idea for both models.

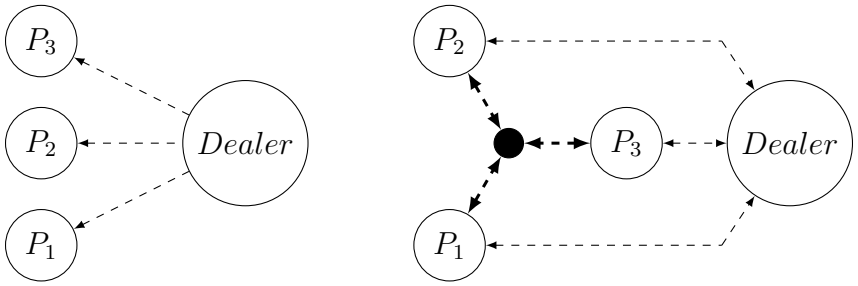

Figure C.1: Communication channels in the offline phase - *The figure represents the communication channels in both the TD model (left) and in the HBC model (right) used during the offline phase. In the first model, the dealer sends the triples to each player via a private channel. In the second model, the players have access to a private broadcast channel, shared between all of them and each player shares an additional private channel with the dealer. The private channels are denoted with dashed lines. The figure represents 3 players, but each model can be extended to an arbitrary number n of players.*

Afterwards, the online phase only requires a public broadcast channel between the players, for both the TD and the HBC models, as shown in Figure C.2. In practice, because of the underlying encryption, private channels (e.g., SSL connections) have a lower throughput (generally $\approx$ 20MB/s) than public channels (plain TCP connections, generally from 100 to 1000MB/s between cloud instances).

The majority of HBC protocols proposed in the literature present a scenario with only 2 players. In [CDN15] and [AFL+16], the authors describe efficient HBC protocols that can be used to perform a fast MPC multiplication in a model with three players. The two schemes assume that the parties follow correctly the protocol and that two players do not collude. The scheme proposed in [CDN15] is very complex to scale for more than three parties, while the protocol in [AFL+16] can be extended to a generic number of players, but requires a quadratic number of private channels (one for every pair of players).
We propose a different protocol for generating the multiplicative triples in the HBC scenario, that is efficient for any arbitrary number n of players. In our scheme, the dealer evaluates the non-linear parts in the triple generation, over the masked

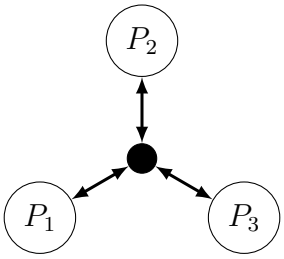

Figure C.2: Communication channels in the online phase - *The figure represents the communication channels (the same type for both the HBC and the TD model) used during the online phase. The players send and receive masked values via a public broadcast channel (public channels are denoted with plain lines). Their number, limited to 3 in the example, can easily be extended to a generic number n of players.*

data produced by the players, then he distributes the masked shares. The mask is common to all players, and it is produced thanks to the private broadcast channel that they share. Finally, each player produces his triple by unmasking the precomputed data received from the dealer.

To give more details on the generation of the triples, we detail three algorithms that can be used in the HBC model:

- Algorithm 15: used for the generation of multiplicative Beaver's triples.
- Algorithm 16: used for the generation of the triples needed in the computation of the power function.
- Algorithm 17: used for the generation of the triples needed in the evaluation of a trigonometric polynomial.

All the algorithms follow the same footstep: the dealer and the players collaborate for the generation of triples and none of them is supposed to have access to the whole information. The players generate their secret shares (of λ and μ in Algorithm 15, and of λ only in Algorithms 16 and 17), that each one keeps secret. They also generate secret shares of a common mask, that they share between each other via the broadcast channel, but which remains secret to the dealer. The player then mask their secret shares with the common mask and sends them to the dealer, who evaluates the non-linear parts (product in Algorithm 15, power in Algorithms 16 and trigonometric function in Algorithm 17). The dealer generates new additive shares for the result and sends these values back to each player via the private channel. This way, the players don't know each other's shares. Finally, the players, who know the common mask, can independently unmask their secret shares, and obtain their final share of the triple, which is therefore unknown to the dealer.

Algorithm 15 Honest but curious triples generation for multiplication.

Output: Shares $([\![\lambda]\!], [\![\mu]\!], [\![z]\!])$ with $z = \lambda\mu$.

1: Each player P_i generates $a_i, b_i, \lambda_i, \mu_i$ (from the according distribution).
2: Each player P_i shares with all other players a_i, b_i.
3: Each player computes $a = a_1 + \cdots + a_n$ and $b = b_1 + \cdots + b_n$.
4: Each player P_i sends to the dealer $a_i + \lambda_i$ and $b_i + \mu_i$.
5: The dealer computes $a + \lambda$, $b + \mu$ and $w = (a + \lambda)(b + \mu)$.
6: The dealer creates $[\![w]\!]_+$ and sends w_i to player P_i, for $i = 1, \ldots, n$.
7: Player P_1 computes $z_1 = w_1 - ab - a\mu_1 - b\lambda_1$.
8: Player i for $i = 2, \ldots n$ computes $z_i = w_i - a\mu_i - b\lambda_i$.

Algorithm 16 Honest but curious triples generation for the power function.

Output: Shares $[\![\lambda]\!]$ and $[\![\lambda^{-\alpha}]\!]$.

1: Each player P_i generates λ_i, a_i (from the according distribution).
2: Each player P_i shares with all other players a_i.
3: Each player computes $a = a_1 + \cdots + a_n$.
4: Each player P_i generates z_i in a way that $\sum_{i=1}^{n} z_i = 0$.
5: Each player P_i sends to the dealer $z_i + a\lambda_i$.
6: The dealer computes $a\lambda$ and $w = (a\lambda)^{-\alpha}$.
7: The dealer creates $[\![w]\!]_+$ and sends w_i to player P_i, for $i = 1, \ldots, n$.
8: Each player P_i right-multiplies w_i with a^{α} to obtain $(\lambda^{-\alpha})_i$.

C.6 Application to Logistic Regression

In a classification problem one is given a data set, also called a *training set*, that we will represent here by a matrix $X \in \mathcal{M}_{N,k}(\mathbb{R})$, and a *training vector* $\mathbf{y} \in \{0,1\}^N$. The data set consists of N input vectors of k features each, and the coordinate y_i of the vector $\mathbf{y}$ corresponds to the class (0 or 1) to which the i-th element of the data set belongs to. Formally, the goal is to determine a function $h_\theta : \mathbb{R}^k \to \{0,1\}$ that takes as input a vector $\mathbf{x}$, containing k features, and which outputs $h_\theta(\mathbf{x})$ predicting reasonably well y, the corresponding output value.

In logistic regression typically one uses hypothesis functions $h_\theta : \mathbb{R}^{k+1} \to [0,1]$ of the form

$$h_\theta(\mathbf{x}) = \texttt{sigmo}(\theta^T \mathbf{x}),$$

where $\theta^T \mathbf{x} = \sum_{i=0}^{k} \theta_i x_i \in \mathbb{R}$ and $x_0 = 1$. The vector θ, also called *model*, is the parameter that needs to be determined. In order to do this, a convex *cost function* $C_{\mathbf{x},y}(\theta)$ measuring the quality of the model at a data point $(\mathbf{x}, y)$ is defined as

$$C_{\mathbf{x},y}(\theta) = -y \log h_\theta(\mathbf{x}) - (1-y) \log(1 - h_\theta(\mathbf{x})).$$

The cost for the whole dataset is thus computed as $\sum_{i=1}^{N} C_{\mathbf{x_i},y_i}(\theta)$. The overall goal

Algorithm 17 Honest but curious triples generation for a trigonometric polynomial.

Output: Shares $([\![\lambda]\!], [\![e^{im_1\lambda}]\!]_+, \ldots, [\![e^{im_N\lambda}]\!]_+)$.

1: Each player P_i generates λ_i, a_i (uniformly modulo 2π)
2: Each player P_i broadcasts a_i to all other players.
3: Each player computes $a = a_1 + \cdots + a_n \mod 2\pi$.
4: Each player P_i sends to the dealer $\lambda_i + a_i \mod 2\pi$.
5: The dealer computes $\lambda + a \mod 2\pi$ and $w^{(1)} = e^{im_1(\lambda+a)}, \ldots, w^{(N)} = e^{im_N(\lambda+a)}$
6: The dealer creates $[\![w^{(1)}]\!]_+, \ldots, [\![w^{(N)}]\!]_+$ and sends $w_i^{(1)}, \ldots, w_i^{(N)}$ to player P_i.
7: Each player P_i multiplies each $w_i^{(j)}$ by $e^{-im_j a}$ to get $(e^{im_j\lambda})_i$, for all $j \in [1, N]$.

is to determine a model θ whose cost function is as close as possible to 0. A common method to achieve this is the so called *gradient descent* which consists of constantly updating the model θ as

$$\theta := \theta - \alpha \nabla C_{\mathbf{x},y}(\theta),$$

where $\nabla C_{\mathbf{x},y}(\theta)$ is the gradient of the cost function and $\alpha > 0$ is a constant called the *learning rate*.
Choosing the optimal α depends largely on the quality of the dataset: if α is too large, the method may diverge, and if α is too small, a very large number of iterations are needed to reach the minimum. Unfortunately, tuning this parameter requires either to reveal information on the data, or to have access to a public fake training set, which is not always feasible in private MPC computations. This step is often silently ignored in the literature. Similarly, preprocessing techniques such as feature scaling, or orthogonalization techniques can improve the dataset, and allow to increase the learning rate significantly. But again, these techniques cannot easily be implemented when the input data is shared, and when correlation information should remain private.

In this work, we choose to implement the IRLS (Iteratively Reweighted Least Squares) method [Bjö96, Section 4.3], which does not require feature scaling, works with learning rate 1, and converges in much less iterations, provided that we have enough floating point precision. In this case, the model is updated as:

$$\theta := \theta - H(C_{\mathbf{x},y}(\theta))^{-1} \cdot \nabla C_{\mathbf{x},y}(\theta),$$

where $H(C_{\mathbf{x},y}(\theta))$ is the Hessian matrix of the cost function in θ.

C.6.1 Implementation and Experimental Results

We implemented an MPC proof-of-concept of the logistic regression algorithm in C++. We represented numbers in $\mathcal{C}(B, p)$ classes with 128-bit floating point numbers, and set the masking security parameter to $\tau = 40$ bits. Since a 128-bit number has 113 bits of precision, and the multiplication algorithm needs $2\tau = 80$ bits of

masking, we still have 33 bits of precision that we can freely use throughout the computation. Since our benchmarks are performed on a regular x86_64 CPU, 128-bit floating point arithmetic is emulated using GCC's quadmath library. Additional speed-ups could be achieved on more recent hardware that natively support these operations, such as the IBM's next POWER9 processor.
In our proof of concept, our main focus was to improve the running time, the floating point precision, and the communication complexity of the online phase, so we implemented the offline phase only for the TD scenario, leaving the HBC dealer variant as a future work.

Algorithm 18 Model training: $\mathsf{Train}(X, \mathbf{y})$

Input: A dataset $X \in \mathcal{M}_{N,k}(\mathbb{R})$ and a training vector $\mathbf{y} \in \{0,1\}^N$
Output: The model $\theta \in \mathbb{R}^k$ that minimizes $\mathsf{Cost}_{X,\mathbf{y}}(\theta)$

1: Precompute $\text{Prods}_i = X_i^T X_i$ for $i \in [0, N-1]$
2: $\theta \leftarrow [0, \ldots, 0] \in \mathbb{R}^k$
3: **for** iter $= 1$ to IRLS_ITERS **do** ▷ In practice IRLS_ITERS $= 8$
4: $\quad \mathbf{a} \leftarrow X \cdot \theta$
5: $\quad \mathbf{p} \leftarrow [\texttt{sigmo}(a_0), \ldots, \texttt{sigmo}(a_{N-1})]$
6: $\quad \text{pmp} \leftarrow [p_0(1-p_0), \ldots, p_{N-1}(1-p_{N-1})]$
7: $\quad \text{grad} \leftarrow X^T(\mathbf{p} - \mathbf{y})$
8: $\quad H \leftarrow \text{pmp} \cdot \text{Prods}$
9: $\quad \theta = \theta - H^{-1} \cdot \text{grad}$
10: **end for**
11: return θ

Model-training algorithm with the IRLS method. The algorithm is explained over the plaintext. In the MPC instantiation, each player gets a secret share for each variable. Every product is evaluated using Beaver's triples, and the sigmoid using the Fourier method.

We implemented the logistic regression model training described in Algorithm 18. Each iteration of the main loop evaluates the gradient (grad) and the Hessian (H) of the cost function at the current position θ, and solves the Hessian system (line 7) to find the next position. Most of the computation steps are bilinear on large matrices or vectors, and each of them is evaluated via a Beaver's triple in a single round of communication.
In step 5 the sigmoid functions are approximated (in parallel) by an odd trigonometric polynomial of degree 23, which provides 20 bits of precision on the whole interval. We therefore use a vector of Fourier triples, as described in Section C.4.
The Hessian system (step 9) is masked by two (uniformly random) orthonormal matrices on the left and the right, and revealed, so the resolution can be done in plaintext. Although this method reveals the norm of the gradient, which is predictable anyway, it hides its direction entirely, which is enough to ensure that the

final model remains private.
Finally, since the input data is not necessarily feature-scaled, it is essential to start from the zero position (step 2) and not a random position, because the first one is guaranteed to be in the IRLS convergence domain.

To build the MPC evaluation of Algorithm 18, we wrote a small compiler to pre-process this high level listing, unroll all for loops, and turn it into a sequence of instructions on immutable variables (which are read-only once they are affected). More importantly, the compiler associates a single additive mask λ_U to each of these immutable variables U.
This solves two important problems that we saw in the previous sections:

- The masking information for huge matrices that are re-used throughout the algorithm are transmitted only once during the whole protocol (this optimization already appears in [MZ17], and in our case, it has a huge impact for the constant input matrix, and their precomputed products, which are re-used in all IRLS iterations).

- It mitigates the attack that would retrieve information by averaging its masked distribution, because an attacker never gets two samples of the same distribution. This justifies the choice of 40 bits of security for masking.

During the offline phase, the trusted dealer generates one random mask value for each immutable variable, and secret shares these masks. For all matrix-vector or matrix-matrix products between any two immutable variables U and V (coming from Lines 1, 4, 6, 7 and 8 of Algorithm 18), the trusted dealer also generates a specific multiplication triple using the masks λ_U of U and λ_V of V. More precisely, he generates and distributes additive shares for $\lambda_U \cdot \lambda_V$ as well as integer vectors/matrices of the same dimensions as the product for the share-reduction phase. These integer coefficients are taken modulo 256 for efficiency reasons.

Practical results We implemented all the described algorithms and we tested our code for two and three parties, using cloud instances on both the AWS and the Azure platforms, having Xeon E5-2666 v3 processors. In our application each instance communicates via its public IP address. Furthermore, we use the zeroMQ library to handle low-level communications between the players (peer-to-peer, broadcast, central nodes etc...).

We tested our method by fixing the number of IRLS iterations to 8, which is enough to reach a perfect convergence for most datasets, and we experimentally verified that the MPC computation outputs the same model as the one with plaintext iterations. We also tested different sizes for the dataset, from 10.000 to 150.000 points, having 8, 12 or 20 features each. As instance, for a dataset of 150.000 points, the total

running time of the online phase ranges from 1 to 5 minutes. This running time is mostly due to the use of emulated quad-float arithmetic, and this MPC computation is no more than 20 times slower than the plaintext logistic regression on the same datasets, if we implement it using the same 128-bit floats.

More interestingly, we see that the overall size of the totality of the triples and the amount of online communications are small: for instance, a logistic regression on 150000 points with 8 features requires only $756MBytes$ of triples per player, and out of it, only $205MBytes$ of data are broadcast during the online phase per player. This is due to the fact that a Fourier triple is much larger than the value that is masked and exchanged. Because of this, the communication time is insignificant compared to the whole running time, even with regular WAN bandwidth.

Finally, when the input data is guaranteed to be feature-scaled, we can improve the whole time, memory and communication complexities by about 30% by performing 3 classical gradient descent iterations followed by 5 IRLS iterations instead of 8 IRLS iterations.

We have tested our platform on datasets that were provided by the banking industry. For privacy reasons, these datasets cannot be revealed. However, the behaviour described in this paper can be reproduced by generating random data sets, for instance, with Gaussian distribution, setting the acceptance threshold to 0.5%, and adding some noise by randomly swapping a few labels.

Notations

Abbreviations

- BDD: bounded distance decoding.
- BK: bootstrapping key.
- Dec: decryption algorithm.
- Enc: encryption algorithm.
- Eval: evaluation algorithm.
- FHE: fully homomorphic encryption.
- GCD: great common divisor.
- GLWE: general learning with errors problem/encryption.
- GSW: Gentry, Sahai and Waters (known as GSW) scheme/encryption.
- HBC: honest but curious.
- HE: homomorphic encryption.
- i.e.: (Latin abbreviation of *id est*) means "that is".
- IND-CCA1: indistinguishability under chosen ciphertext attack.
- IND-CCA2: indistinguishability under adaptive chosen ciphertext attack.
- IND-CPA: indistinguishability under chosen plaintext attack.
- IND-CVA: indistinguishability under (chosen) ciphertext verification attack.
- KeyGen: key generation algorithm.
- KS: key-switching key.
- LHE: leveled homomorphic encryption.
- LSB: least significant bit.

- LWE: learning with errors problem/encryption.
- MPC: multi-party computation.
- MSB: most significant bit.
- NIZK: non-interactive zero-knowledge.
- pk: public key.
- PK: public key (matrix).
- PPT: probabilistic polynomial time.
- RingGSW: ring GSW scheme/encryption.
- RingLWE: ring learning with errors problem/encryption.
- SHE: somewhat homomorphic encryption.
- SIS: short integer solution.
- sk: secret key.
- $t_{\mathtt{CB}}$: time per circuit bootstrapping.
- TD: trusted dealer.
- $t_{\mathtt{GB}}$: time per gate bootstrapping.
- TGSW: torus GSW scheme/encryption.
- $t_{\mathtt{KS}}$: time per key switching.
- TLWE: torus learning with errors problem/encryption.
- TRGSW: torus ring GSW scheme/encryption.
- TRLWE: torus ring learning with errors problem/encryption.
- $t_{\mathtt{XP}}$: time per external product.

Mathematical and algorithmic notations

- $\cdot$: depending on the context, it represents the classical multiplication, the polynomial multiplication, an external product or the scalar product (sometimes omitted).
- $\times$: represents an internal product.
- $|\ldots|$: absolute value.

- $[\ldots|\ldots]$ or $(\ldots|\ldots)$: concatenation.
- $\|\ldots\|$: norm.
- $[\![a, b]\!]$: interval including all the integers between a and b.
- $a \leftarrow A$: a has been sampled from A.
- $\mathbb{B}$: $\{0, 1\}$.
- $\mathbb{B}_N[X]$: polynomials in $\mathbb{Z}_N[X]$ with binary coefficients.
- $\mathbb{C}$: complex numbers.
- $\mathcal{D}$: distibution.
- dist : distance.
- $\mathbb{E}$: expectation.
- E^p : vectors of size p with entries in E.
- Id_m : the identity matrix of size $m \times m$.
- $\mathcal{M}_{p,q}(E)$: $p \times q$-size matrices with elements in E.
- $f(x) = O(g(x))$: $\exists k > 0$ such that $|f(x)| \leq kg(x)$ asymptotically.
- $f(x) = \tilde{O}(g(x))$: $\exists k > 0$ such that $f(x) = O(g(x)\log^k(g(x)))$.
- $f(x) = \Theta(g(x))$: $\exists k_1, k_2 > 0$ such that $k_1 g(x) \leq |f(x)| \leq k_2 g(x)$ asymptotically.
- $f(x) = \Omega(g(x))$: $\exists k > 0$ such that $|f(x)| \geq kg(x)$ asymptotically.
- $poly(\ldots)$: polynomial time.
- $\mathbb{R}$: real numbers.
- $\texttt{sigmo}(\ldots)$: sigmoid function.
- $\mathbb{T}$: the real torus $\mathbb{R}/\mathbb{Z}$, i.e. real numbers modulo 1.
- $\mathbb{T}_N[X]$: $\mathbb{R}[X]/(X^N + 1) \mod 1$.
- $\mathcal{U}$: uniform distribution.
- Var : variance.
- $\mathcal{X}$: probability distribution.
- $\mathbb{Z}$: integer numbers.
- $\mathbb{Z}_N[X]$: $\mathbb{Z}[X]/(X^N + 1)$.

List of publications

[BCG+18] Christina Boura, Ilaria Chillotti, Nicolas Gama, Dimitar Jetchev, Stanislav Peceny, and Alexander Petric. High-precision privacy-preserving real-valued function evaluation. 2018.

[CGG16] Ilaria Chillotti, Nicolas Gama, and Louis Goubin. Attacking fhe-based applications by software fault injections. *IACR Cryptology ePrint Archive*, 2016:1164, 2016.

[CGGI16a] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, pages 3–33, 2016.

[CGGI16b] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. A homomorphic LWE based e-voting scheme. In *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, pages 245–265, 2016.

[CGGI16c] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: tfhe repository. https://github.com/tfhe/tfhe, 2016.

[CGGI16d] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption library. https://tfhe.github.io/tfhe/, August 2016.

[CGGI17a] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, pages 377–408, 2017.

[CGGI17b] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: experimental-tfhe repository. https://github.com/tfhe/experimental-tfhe, 2017.

Bibliography

[ABD16] Martin R. Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, pages 153–178, 2016.

[ACD+17] Martin R Albrecht, Benjamin R Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn Postlethwaite, Fernando Virdia, and Thomas Wunderer. Estimate all the {LWE, NTRU} schemes. *https://estimate-all-the-lwe-ntru-schemes.github.io/docs*, 2017.

[ACF+15] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. On the complexity of the BKW algorithm on LWE. *Des. Codes Cryptography*, 74(2):325–354, 2015.

[ACG+16] Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 308–318, 2016.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, pages 595–618, 2009.

[AD17] Martin R. Albrecht and Amit Deo. Large modulus ring-lwe $\geq$ module-lwe. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, pages 267–296, 2017.

[AdMP] Ben Adida, Olivier de Marneffe, and Olivier Pereira. Helios: Trust the vote. https://heliosvoting.org/.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 327–343, 2016.

[AFL+16] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 805–817, 2016.

[AHPW16] Yoshinori Aono, Takuya Hayashi, Le Trieu Phong, and Lihua Wang. Privacy-preserving logistic regression with distributed data sources via homomorphic encryption. *IEICE Transactions*, 99-D(8):2079–2089, 2016.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 99–108, 1996.

[Alb17] Martin R. Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in helib and SEAL. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, pages 103–129, 2017.

[aMR17] Cryptography Research Group at Microsoft Research. Seal - simple encrypted arithmetic library. https://www.microsoft.com/en-us/research/project/simple-encrypted-arithmetic-library/, 2017.

[AP14] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 297–314, 2014.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Mathematical Cryptology*, 9(3):169–203, 2015.

[BBL17] Daniel Benarroch, Zvika Brakerski, and Tancrède Lepoint. FHE over the integers: Decomposed and batched in the post-quantum regime. In *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part II*, pages 271–301, 2017.

[BCG+15] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. Sok: A comprehensive analysis of game-based ballot privacy definitions. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 499–516, 2015.

[BCG+18] Christina Boura, Ilaria Chillotti, Nicolas Gama, Dimitar Jetchev, Stanislav Peceny, and Alexander Petric. High-precision privacy-preserving real-valued function evaluation. 2018.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, pages 420–432, 1991.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 1–18, 2001.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC*, volume 3378, pages 325–341. Springer, 2005.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325, 2012.

[BGW00] Adam L. Buchsbaum, Raffaele Giancarlo, and Jeffery Westbrook. On the determinization of weighted finite automata. *SIAM J. Comput.*, 30(5):1502–1531, 2000.

[Bjö96] Åke Björck. *Numerical methods for least square problems*. SIAM, 1996.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, 2003.

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, pages 1–12, 1998.

[BLLN13] Joppe W. Bos, Kristin E. Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *Cryptography and Coding - 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings*, pages 45–64, 2013.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 575–584, 2013.

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings*, pages 192–206, 2008.

[BP16] Zvika Brakerski and Renen Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, pages 190–213, 2016.

[BR94] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, pages 92–111, 1994.

[BR15] Jean-François Biasse and Luis Ruiz. FHEW with efficient multibit bootstrapping. In *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*, pages 119–135, 2015.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. *IACR Cryptology ePrint Archive*, 2012:78, 2012.

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:109, 2011.

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology*

Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings, pages 505–524, 2011.

[BV14a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM J. Comput.*, 43(2):831–871, 2014.

[BV14b] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 1–12, 2014.

[CCF+16] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. In *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, pages 313–333, 2016.

[CCK+13] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi, and Aaram Yun. Batch fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 315–335, 2013.

[CDN15] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[CDS15] Sergiu Carpov, Paul Dubrulle, and Renaud Sirdey. Armadillo: A compilation chain for privacy preserving applications. In *Proceedings of the 3rd International Workshop on Security in Cloud Computing, SCC@ASIACCS '15, Singapore, Republic of Singapore, April 14, 2015*, pages 13–19, 2015.

[CDW17] Ronald Cramer, Léo Ducas, and Benjamin Wesolowski. Short stickelberger class relations and application to ideal-svp. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, pages 324–348, 2017.

[CEQ16] CryptoExperts, INP ENSEEIHT, and Quarkslab. Nfllib - an ntt-based fast lattice library. https://github.com/quarkslab/NFLlib, 2016.

[CF13] Dario Catalano and Dario Fiore. Practical homomorphic macs for arithmetic circuits. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 336–352, 2013.

[CFGN14] Dario Catalano, Dario Fiore, Rosario Gennaro, and Luca Nizzardo. Generalizing homomorphic macs for arithmetic circuits. In *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, pages 538–555, 2014.

[CFW14] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 371–389, 2014.

[CGG16] Ilaria Chillotti, Nicolas Gama, and Louis Goubin. Attacking fhe-based applications by software fault injections. *IACR Cryptology ePrint Archive*, 2016:1164, 2016.

[CGGI14] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Election verifiability for helios under weaker trust assumptions. In *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II*, pages 327–344, 2014.

[CGGI16a] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, pages 3–33, 2016.

[CGGI16b] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. A homomorphic LWE based e-voting scheme. In *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, pages 245–265, 2016.

[CGGI16c] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: tfhe repository. https://github.com/tfhe/tfhe, 2016.

[CGGI16d] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption library. https://tfhe.github.io/tfhe/, August 2016.

[CGGI17a] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, pages 377–408, 2017.

[CGGI17b] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: experimental-tfhe repository. https://github.com/tfhe/experimental-tfhe, 2017.

[Cha01] Bernard Chazelle. The pcp theorem. In *Bourbaki Seminar*, volume 895, 2001.

[CHK+18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, pages 360–384, 2018.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, pages 409–437, 2017.

[CLT14] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Scale-invariant fully homomorphic encryption over the integers. In *Public Key Cryptography*, volume 8383, pages 311–328, 2014.

[CM08] Kamalika Chaudhuri and Claire Monteleoni. Privacy-preserving logistic regression. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pages 289–296, 2008.

[CMNT11] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In *Crypto*, volume 6841, pages 487–504. Springer, 2011.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, pages 1–20, 2011.

[CNT12] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 446–464, 2012.

[CRRV17] Ran Canetti, Srinivasan Raghuraman, Silas Richelson, and Vinod Vaikuntanathan. Chosen-ciphertext secure fully homomorphic encryption. In *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part II*, pages 213–240, 2017.

[CS11] Véronique Cortier and Ben Smyth. Attacking and fixing helios: An analysis of ballot secrecy. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF 2011, Cernay-la-Ville, France, 27-29 June, 2011*, pages 297–311, 2011.

[CS15] Jung Hee Cheon and Damien Stehlé. Fully homomophic encryption over the integers revisited. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 513–536, 2015.

[CS16] Ana Costache and Nigel P. Smart. Which ring based somewhat homomorphic encryption scheme is best? In *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*, pages 325–340, 2016.

[CT14] Massimo Chenal and Qiang Tang. On key recovery attacks against existing somewhat homomorphic encryption schemes. In *Progress in Cryptology - LATINCRYPT 2014 - Third International Conference on Cryptology and Information Security in Latin America, Florianópolis, Brazil, September 17-19, 2014, Revised Selected Papers*, pages 239–258, 2014.

[Dai16] Wei Dai. cuhe - homomorphic and fast. https://github.com/vernamlab/cuHE, 2016.

[Data] Dataset. Arcene Data Set. `https://archive.ics.uci.edu/ml/datasets/Arcene`.

[Datb] Dataset. MNIST Database. `http://yann.lecun.com/exdb/mnist/`.

[DDA13] Angsuman Das, Sabyasachi Dutta, and Avishek Adhikari. Indistinguishability against chosen ciphertext verification attack revisited: The complete picture. In *Provable Security - 7th International Conference, ProvSec 2013, Melaka, Malaysia, October 23-25, 2013. Proceedings*, pages 104–120, 2013.

[DG09] Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Handbook of Weighted Automata, EATCS Monographs in Theoretical Computer Science*, pages 175–211, 2009.

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

[DM15] Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 617–640, 2015.

[DM17] Leo Ducas and Daniele Micciancio. FHEW: A fully homomorphic encryption library. https://github.com/lducas/FHEW, 2017.

[dPLNS17] Rafaël del Pino, Vadim Lyubashevsky, Gregory Neven, and Gregor Seiler. Practical quantum-safe voting from lattices. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1565–1581, 2017.

[DPSZ] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. SPDZ Software. `https://www.cs.bris.ac.uk/Research/CryptographySecurity/SPDZ/`.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 643–662, 2012.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[FJ05] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[FMNP16] Dario Fiore, Aikaterini Mitrokotsa, Luca Nizzardo, and Elena Pagnin. Multi-key homomorphic authenticators. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, pages 499–530, 2016.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[GBC16] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016.

[GCG] Stéphane Glondu, Véronique Cortier, and Pierrick Gaudry. Belenios: Verifiable online voting system. http://www.belenios.org/.

[GDL+16] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 201–210, 2016.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178, 2009.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49, 2013.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 626–645, 2013.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 850–867, 2012.

[GINX14] Nicolas Gama, Malika Izabachène, Phong Q. Nguyen, and Xiang Xie. Structural lattice reduction: Generalized worst-case to average-case reductions. *IACR Cryptology ePrint Archive*, 2014:283, 2014.

[GN08a] Nicolas Gama and Phong Q. Nguyen. Finding short lattice vectors within mordell's inequality. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 207–216, 2008.

[GN08b] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, pages 31–51, 2008.

[GNR10] Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, pages 257–278, 2010.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 197–206, 2008.

[GSB+17] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. Privacy-preserving distributed linear regression on high-dimensional data. *PoPETs*, 2017(4):345–364, 2017.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. *IACR Cryptology ePrint Archive*, 2013:340, 2013.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. *J. ACM*, 62(6):45:1–45:33, 2015.

[HAO15] Ryo Hiromasa, Masayuki Abe, and Tatsuaki Okamoto. Packing messages and optimizing bootstrapping in GSW-FHE. In *Public-Key Cryp-*

tography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings, pages 699–715, 2015.

[HEA17] Heaan. https://github.com/kimandrik/HEAAN, 2017.

[HG01] Nick Howgrave-Graham. Approximate integer common divisors. In *CaLC*, volume 1, pages 51–66. Springer, 2001.

[HGS99] Chris Hall, Ian Goldberg, and Bruce Schneier. Reaction attacks against several public-key cryptosystems. In *Information and Communication Security, Second International Conference, ICICS'99, Sydney, Australia, November 9-11, 1999, Proceedings*, pages 2–12, 1999.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*, pages 267–288, 1998.

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 447–464, 2011.

[HS14] Shai Halevi and Victor Shoup. Algorithms in helib. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 554–571, 2014.

[HS17] S. Halevi and Igor V. Shoup. Helib - an implementation of homomorphic encryption. https://github.com/shaih/HElib/, 2017.

[HSJ09] ZhenYu Hu, FuChun Sun, and JianChun Jiang. Ciphertext verification security of symmetric encryption schemes. *Science in China Series F: Information Sciences*, 52(9):1617–1631, 2009.

[JA16] Angela Jäschke and Frederik Armknecht. Accelerating homomorphic computations on rational numbers. In *Applied Cryptography and Network Security - 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings*, pages 405–423, 2016.

[JCJ10] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Towards Trustworthy Elections, New Directions in Electronic Voting*, pages 37–63, 2010.

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

[Kan83] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 193–206. ACM, 1983.

[Kra93] David W Kravitz. Digital signature algorithm, July 27 1993. US Patent 5,231,668.

[LGM16a] Zengpeng Li, Steven D. Galbraith, and Chunguang Ma. Preventing adaptive key recovery attacks on the gentry-sahai-waters leveled homomorphic encryption scheme. *IACR Cryptology ePrint Archive*, 2016:1146, 2016.

[LGM16b] Zengpeng Li, Steven D. Galbraith, and Chunguang Ma. Preventing adaptive key recovery attacks on the GSW levelled homomorphic encryption scheme. In *Provable Security - 10th International Conference, ProvSec 2016, Nanjing, China, November 10-11, 2016, Proceedings*, pages 373–383, 2016.

[LLL82] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[LMSV11] Jake Loftus, Alexander May, Nigel P. Smart, and Frederik Vercauteren. On cca-secure somewhat homomorphic encryption. In *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, pages 55–72, 2011.

[LN13] Mingjie Liu and Phong Q. Nguyen. Solving BDD by enumeration: An update. In *Topics in Cryptology - CT-RSA 2013 - The Cryptographers' Track at the RSA Conference 2013, San Francisco,CA, USA, February 25-March 1, 2013. Proceedings*, pages 293–309, 2013.

[LN14] Tancrede Lepoint and Michael Naehrig. A comparison of the homomorphic encryption schemes fv and yashe. In *International Conference on Cryptology in Africa*, pages 318–335. Springer, 2014.

[Lol17] $\lambda \circ \lambda$ (lol). https://github.com/cpeikert/Lol, 2017.

[LP00] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Advances in Cryptology - CRYPTO 2000, 20th Annual International*

Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings, pages 36–54, 2000.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23. Springer, 2010.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015.

[LSS14] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 855–863, 2014.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1219–1234, 2012.

[Mic18] Daniele Micciancio. On the hardness of learning with errors with binary secrets. 2018.

[MJSC16] Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards stream ciphers for efficient FHE with low-noise ciphertexts. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, pages 311–343, 2016.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 700–718, 2012.

[MW16] Daniele Micciancio and Michael Walter. Practical, predictable lattice basis reduction. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, pages 820–849, 2016.

[MZ17] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 19–38, 2017.

[Nay] Project Nayuki: Fast fourier transform in x86 assembly. https://www.nayuki.io/page/fast-fourier-transform-in-x86-assembly.

[NK15] Koji Nuida and Kaoru Kurosawa. (batch) fully homomorphic encryption over integers for non-binary message spaces. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 537–555, 2015.

[NLV11] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011*, pages 113–124, 2011.

[NWI+13] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 334–348, 2013.

[PAH+17] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. Privacy-preserving deep learning: Revisited and enhanced. In *Applications and Techniques in Information Security - 8th International Conference, ATIS 2017, Auckland, New Zealand, July 6-7, 2017, Proceedings*, pages 100–110, 2017.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 223–238, 1999.

[PAL17] Palisade - lattice cryptography library. https://git.njit.edu/palisade/PALISADE, 2017.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 238–252, 2013.

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

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93, 2005.

[Rot10] Ron Rothblum. Homomorphic encryption: from private-key to public-key. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:146, 2010.

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

[SE94] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66(1-3):181–199, 1994.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.

[Smy12] Ben Smyth. Replay attacks that violate ballot secrecy in helios. *IACR Cryptology ePrint Archive*, 2012:185, 2012.

[SS11] Damien Stehlé and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, pages 27–47, 2011.

[SSTX09] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, pages 617–635, 2009.

[SV10] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings*, pages 420–443, 2010.

[SV14] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Des. Codes Cryptography*, 71(1):57–81, 2014.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 24–43. Springer, 2010.

[WTK+13] Shuang Wu, Tadanori Teruya, Junpei Kawamoto, Jun Sakuma, and Hiroaki Kikuchi. Privacy-preservation for stochastic gradient descent application to secure logistic regression. volume 27, pages 1–4, 2013.

[YJ00] Sung-Ming Yen and Marc Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Trans. Computers*, 49(9):967–970, 2000.

[ZPS11] Zhenfei Zhang, Thomas Plantard, and Willy Susilo. Reaction attack on outsourced computing with fully homomorphic encryption schemes. In *Information Security and Cryptology - ICISC 2011 - 14th International Conference, Seoul, Korea, November 30 - December 2, 2011. Revised Selected Papers*, pages 419–436, 2011.

[ZPS12] Zhenfei Zhang, Thomas Plantard, and Willy Susilo. On the CCA-1 security of somewhat homomorphic encryption over the integers. In *Information Security Practice and Experience - 8th International Conference, ISPEC 2012, Hangzhou, China, April 9-12, 2012. Proceedings*, pages 353–368, 2012.