Contributions to the optimization of TFHE's functional bootstrapping for the evaluation of non-polynomial operators

Pierre-Emmanuel Clet

université PARIS-SACLAY

Contributions to the optimization of TFHE's functional bootstrapping for the evaluation of non-polynomial operators

Contributions à l'optimisation du bootstrapping fonctionnel de TFHE pour l'évaluation d'opérateurs non polynomiaux

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 580, Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique mathématique
Graduate School : Informatique et sciences du numérique
Référent : Université de Versailles-Saint-Quentin-en-Yvelines

Thèse préparée dans l'unité de recherche **Institut LIST (Université Paris-Saclay, CEA)**,
sous la direction de **Renaud SIRDEY**, directeur de recherche CEA-LIST,
le co-encadrement de **Aymen BOUDGUIGA**, ingénieur-chercheur CEA-LIST,
le co-encadrement de **Cédric GOUY-PAILLER**, ingénieur-chercheur CEA-LIST

Thèse soutenue à Palaiseau, le 15 janvier 2024, par

Pierre-Emmanuel CLET

Composition du jury

Membres du jury avec voix délibérative

Caroline FONTAINE Directrice de recherche CNRS, ENS Paris-Saclay	Présidente
Philippe GABORIT Professeur des universités, Université de Limoges	Rapporteur & Examinateur
Melek ÖNEN Maîtresse de conférence, HDR, Eurecom	Rapportrice & Examinatrice
Mariya GEORGIEVA Experte en cryptographie, Inpher	Examinatrice
Pascal PAILLIER Expert en cryptographie, CryptoExpert	Examinateur
David POINTCHEVAL Directeur de recherche CNRS, ENS Paris	Examinateur

THESE DE DOCTORAT
NNT : 2024UPASG001

université PARIS-SACLAY

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

Titre: Contributions à l'optimisation du bootstrapping fonctionnel de TFHE pour l'évaluation d'opérateurs non polynomiaux.

Mots clés: FHE, TFHE, réseaux de neurones, bootstrapping fonctionnel, cryptographie.

Résumé: Avec la création et l'utilisation incessantes de données numériques, ces dernières années ont vu naître des inquiétudes au sujet des données sensibles et personnelles. De nouvelles lois, telles que le Règlement Général sur la Protection des Données, ont alors vu le jour pour assurer le respect de la confidentialité des données des individus. Cependant, l'externalisation grandissante du traitement des données notamment avec l'apparition du "machine learning as a service" soulève la question suivante: est-il possible de laisser un tiers traiter nos données tout en les gardant confidentielles ?

Une solution à ce problème vient des chiffrements dits FHE, de l'anglais Fully Homomorphic Encryption. À l'aide de tels cryptosystèmes, des opérations peuvent être appliquées directement sur des messages chiffrés, sans jamais dévoiler ni le message d'origine, ni le message résultant des opérations. Ce corpus de techniques permet donc en théorie d'externaliser des calculs sans compromettre la confidentialité des données utilisées lors de ces calculs. Cela pourrait ouvrir la voie à de nombreuses applications telle que la possibilité d'ouvrir des services de diagnostic médicaux en ligne offrant une totale confidentialité des données médicales des patients.

Malgré cette promesse alléchante, l'important coût computationnel des opérateurs FHE en limite la portée pratique. En effet, un calcul sur données chiffrées peut prendre plusieurs millions de fois plus de temps que son équivalent sur des données non chiffrées. Cela rend inenvisageable l'évaluation d'algorithme trop complexes sur des données chiffrées. Par ailleurs, le surcoût en mémoire apporté par les chiffrements FHE s'élève à un facteur multiplicatif de plusieurs milliers. Ce surcoût peut donc s'avérer rédhibitoire pour des applications sur des systèmes à basse mémoire tels que des systèmes embarqués.

Dans cette thèse nous développons une nouvelle primitive pour le calcul sur données chiffrées basée sur l'opération de "bootstrapping fonctionnel" supportée par le cryptosystème TFHE. Cette primitive permet un gain en latence et en mémoire par rapport aux autres techniques comparables de l'état de l'art. Aussi, nous introduisons une seconde primitive permettant d'effectuer des calculs sous forme de circuit logique permettant un gain significatif de vitesse de calcul par rapport à l'état de l'art. Cette approche pourra notamment être intéressante auprès des concepteurs de compilateurs homomorphes comme alternative à l'utilisation de chiffrement binaire. Ces deux outils se veulent suffisamment généraux pour être applicables à un large panel de cas d'utilisation et ne sont donc pas limités aux cas d'usage présentés dans ce manuscrit.

En guise d'illustration, nous appliquons nos opérateurs au calcul confidentiel de réseaux de neurones externalisés, montrant ainsi la possibilité d'évaluer des réseaux de neurones avec une relativement faible latence, même dans le cas de réseau de neurones de type récurrents. Enfin, nous appliquons nos opérateurs à une technique dite de transchiffrement permettant de s'affranchir des considérations de limitation en mémoire dûes à la grande taille des chiffrés FHE côté client.

université PARIS-SACLAY

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

Title: Contributions to the optimization of TFHE's functional bootstrapping for the evaluation of non-polynomial operators

Keywords: FHE, TFHE, neural networks, functional bootstrapping, cryptography.

Abstract: In recent years, concerns about sensitive and personal data arose due to the increasing creation and use of digital data. New laws, such as the General Data Protection Regulation, have been introduced to ensure that the confidentiality of individuals' data is respected. However, the growing outsourcing of data processing, particularly with the emergence of "machine learning as a service", raises the following question: is it possible to let a third party process our data while keeping it confidential?

One solution to this problem comes in the form of Fully Homomorphic Encryption, or FHE for short. Using FHE cryptosystems, operations can be applied directly to encrypted messages, without ever revealing either the original message or the message resulting from the operations. In theory, this collection of techniques makes it possible to externalise calculations without compromising on the confidentiality of the data used during these calculations. This could pave the way for numerous applications, such as the possibility of offering online medical diagnostic services while ensuring the total confidentiality of the patients' medical data.

Despite this promise, the high computational cost of FHE operators limits their practical scope. A calculation on encrypted data can take several million times longer than its equivalent on non-encrypted data. This makes it unthinkable to evaluate highly time consuming algorithms on encrypted data. In addition, the memory cost of FHE encryption is several thousand times greater than unencrypted data. This overhead may prove to be prohibitive for applications on low-memory systems such as embedded systems.

In this thesis we develop a new primitive for computing on encrypted data based on the "functional bootstrapping" operation supported by the TFHE cryptosystem. This primitive allows a gain in latency and memory compared to other comparable techniques in the state of the art. We are also introducing a second primitive enabling calculations to be performed in the form of a logic circuit, providing a significant gain in calculation speed compared with the state of the art. This approach could be of particular interest to designers of homomorphic compilers as an alternative to the use of binary encryption. These two tools are intended to be sufficiently generic to be applicable to a wide range of use cases and are therefore not limited to the use cases presented in this manuscript.

As an illustration, we apply our operators to the confidential computation of outsourced neural networks, thus demonstrating the possibility of evaluating neural networks with relatively low latency, even in the case of recurrent neural networks. Finally, we apply our operators to a technique known as transciphering, making it possible to overcome memory limitation on the client side coming with the large size of FHE ciphertexts.

Résumé étendu

Avec la création et l'utilisation incessantes de données numériques, ces dernières années ont vu naître des inquiétudes au sujet des données sensibles et personnelles. De nouvelles lois, telles que le Règlement Général sur la Protection des Données (RGPD) ou la California Consumer Privacy Act (CCPA), ont alors vu le jour pour assurer le respect de la confidentialité des données des individus. Cependant, l'externalisation grandissante du traitement des données notamment avec l'apparition du "machine learning as a service" soulève la question suivante: est-il possible de laisser un tiers traiter nos données tout en les gardant confidentielles ?

Une solution à ce problème vient des chiffrements dits totalement homomorphe ou FHE, de l'anglais Fully Homomorphic Encryption. À l'aide de tels cryptosystèmes, des opérations peuvent être appliquées directement sur des messages chiffrés, sans jamais dévoiler ni le message d'origine, ni le message résultant des opérations. Ce corpus de techniques permet donc en théorie d'externaliser des calculs sans compromettre la confidentialité des données utilisées lors de ces calculs. Cela pourrait ouvrir la voie à de nombreuses applications telles que la possibilité d'ouvrir des services de diagnostic médicaux en ligne offrant une totale confidentialité des données médicales des patients.

Malgré cette promesse alléchante, l'important coût computationnel des opérateurs FHE en limite la portée pratique. En effet, un calcul sur données chiffrées peut prendre plusieurs millions de fois plus de temps que son équivalent sur des données non chiffrées. Cela rend inenvisageable l'évaluation d'algorithme trop complexes sur des données chiffrées. Par ailleurs, le surcoût en mémoire apporté par les chiffrements totalement homomorphes s'élève à un facteur multiplicatif de plusieurs milliers. Ce surcoût peut donc s'avérer rédhibitoire pour des applications sur des systèmes à basse mémoire tels que des systèmes embarqués.

Dans cette thèse nous développons une nouvelle primitive pour le calcul sur données chiffrées basée sur l'opération de "bootstrapping fonctionnel" supportée par le cryptosystème TFHE. Cette primitive permet un gain en latence et en mémoire par rapport aux autres techniques comparables de l'état de l'art. Aussi, nous introduisons une seconde primitive permettant d'effectuer des calculs sous forme de circuit logique permettant un gain significatif de vitesse de calcul par rapport à l'état de l'art. Cette approche pourra notamment être intéressante auprès des concepteurs de compilateurs homomorphes comme alternative à l'utilisation de chiffrement binaire. En effet, notre méthode permet une factorisation des opérateurs binaires à l'aide d'opérateurs sur des espaces de messages plus large, résultant en une diminution du nombre total d'opérations à effectuer sur des données

chiffrées et par conséquent en un gain global de temps de calcul sans complexifier la tâche des ces compilateurs. Ces deux outils se veulent suffisamment généraux pour être applicables à un large panel de cas d'utilisation et ne sont donc pas limités aux cas d'usage présentés dans ce manuscrit.

En guise d'illustration, nous appliquons nos opérateurs au calcul confidentiel de réseaux de neurones externalisés, montrant ainsi la possibilité d'évaluer des réseaux de neurones avec une relativement faible latence, même dans le cas de réseau de neurones de type récurrents. Dans ce cadre, le but est d'ouvrir à la technologie FHE le large panel d'applications utilisant des réseaux de neurones, comprenant de manière non exhaustive la reconnaissance d'image, le traitement du langage et la détection d'erreur dans des systèmes complexes. Enfin, nous appliquons nos opérateurs à une technique dite de transchiffrement, autrement dit de changement de format de chiffrement sans jamais passer par une étape de déchiffrement. En particulier, nous nous intéressons au passage d'un chiffrement symétrique compact vers un chiffrement FHE, permettant ainsi de s'affranchir des considérations de limitation en mémoire dûes à la grande taille des chiffrés FHE côté client.

Acknowledgements

These three years of PhD reached their end smoothly, and I have many people to thank for that. First, i thank my advisors Aymen Boudguiga, Cédric Gouy-Paillier and Renaud Sirdey. Thank you for the trust you put in me from the beginning of my thesis, I definitely enjoyed the freedom you gave me to explore and research a subject I was very much unfamiliar with, and I appreciated all the help you gave me during these three years.

I also thank Oana Stan and Martin Zuber with whom I got the chance to work even before my PhD. This thesis would not have even started if not for the great introduction to the FHE world you gave me during my internship.

I thank Akram Bendoukha, Keiten Han and Daphné Trama whom I got to supervise during their own internships. I learned a lot from imparting my knowledge to new researchers and hope that it was as interesting for you that it was for me.

I thank my office roommates Jonathan Fontaine, Valentin Gilbert, Julien Rodriguez, Simon Tollec, and our group of PhD students Antonina Bondarchuk, Marina Checri, Robin Ollive, Marc Renard, Guillaume Roumage, and the Anonymous-one who made each day more fun.

I was amazed by the amount of rock climbers among researchers at CEA, and on that note, I thank Arnaud Grivet with whom I had more opportunities to go climbing than to do research...

I extend my thanks to everyone else at CEA I got to interact with and learn from during these three years of PhD. In particular, the members of the crypto team with whom I got great cryptographic discussions : Jean-Paul Bultel, Olive Chakraborty and Antoine Choffrut.

I thank all the members of the jury of my thesis, Caroline Fontaine, Philippe Gaborit, Mariya Georgieva, Melek Önen, Pascal Paillier et David Pointcheval, for the interest they have shown in my work and the goodwill they have shown during the reviewing process and the PhD defense.

Special thanks to Mathilde and Tomer, not blood related but still part of the family. I will probably have more time from now on, so prepare those RPG board games!

And above all else, I want to thank my family for their unconditional love, it seems that spoiling the last child can also lead to good things...

Bisous papi, toi qui m'a montré que les sciences peuvent être amusantes et qui aurais certainement été le plus fier d'entre nous de me voir devenir docteur.

Contents

Introduction

Data and privacy: Data is a powerful resource used everywhere to guide decision processes. Indeed, with the advent of numerical technologies, data can be efficiently stored and processed in unfathomable amounts. It can, for instance, help a business target its client's preferences by processing consumer data, or help a doctor find the right diagnosis by accessing a patient past medical data.

If data is a powerful resource, it can also be detrimental to the data owner when leaked to unwanted third parties. Indeed, attackers can also make use of data to build more efficient attacks. According to a Federal Trade Commission report[1], more than one fourth of the people who reported losing money to fraud in 2021 got scammed through social medias, which are indeed an efficient way for attackers to both reach many people and easily gain access to massive amount of data to tailor specific attacks and scams. Those attacks resulted in at least 1.2 billion dollars of loss in 2022 (only counting fraud via social media and reported to the Federal Trade Commission[2]).

However, data leakage may cause harm even when users remain vigilant. AADHAR is an Indian biometric ID system, and also the largest in the world. It links the biometric information of Indian citizens and a unique identification number. Suggestions have been made to make it mandatory to get a passport issued, open a bank account, and more. As such, more than a billion Indian citizens used this biometric system. However, in 2018, the AADHAR biometric data of 1.1 billion Indian citizens became available online due to a data leak from on a government website. This data breach could lead to identity theft, financial fraud, and harassment. It could also lead to further privacy breaches, since personal information such as name, address and date of birth, have also been exposed.

These issues highlight the necessity to enhance the technologies and techniques to

[1]https://www.ftc.gov/system/files/attachments/blog_posts/Social media a gold mine for scammers in 2021/social_media_spotlight.pdf

[2]Statistics found at https://www.ftc.gov/business-guidance/blog/2023/02/ftc-crunches-2022-numbers-see-where-scammers-continue-crunch-consumers

protect the data and privacy of people when personal data are used. The handling of data, and more specifically personal data, has been acknowledged as a sensitive topic all over the world. As such, new regulations were issued to ensure that some rights to privacy are given to data subjects[3]. In the EU, the most prominent regulation on data privacy is the General Data Protection Regulation (GDPR) which also inspired similar regulations across the world, such as the California Consumer Privacy Act (CCPA). Notably, the GDPR requires that organisations consider data protection by design and by default.

In this context, new Privacy Enhancing Technologies (PETs), such as Homomorphic Encryption (HE), Multi-Party Computation (MPC), and Trusted Execution Environments (TEE), are under research to contribute to giving people more control over their data and make organisations compliant with the new legislation.

Neural Network and Homomorphic Encryption: Nowadays, Artificial Intelligence (AI) and especially Neural Networks (NNs) are heavily employed techniques allowing for efficient image recognition [21], speech recognition [49], artistic creations [70] and more.

The creation and fine tuning of neural networks rely on massive amount of data which need to be handled with care. For instance, neural networks are used in healthcare to diagnose patients more efficiently in hospitals. In order to train such networks, available medical databases are needed. Thus, special care must be taken to ensure the privacy and confidentiality of the medical data of patients as medical data is private.

Furthermore, when neural network solutions are deployed to the cloud, the query of a user may hold some sensitive information. ChatGPT from OpenAI is one such neural network accessible online on which more and more users become reliant for various usages. Its uses can range from answering questions to correcting computer programs or even writing books, if used cleverly. Considering the wide range of utilities held by ChatGPT, a lot of users' private information may be accessible to OpenAI through its use. Incorporating PETs in the equation, and more specifically Homomorphic Encryption, would allow the use of such neural networks without leaking the input to the service provider.

Besides, some potential applications of outsourced neural networks are also impossible so far due to the threat to privacy involved with their use. For instance, we can easily see the threat an online medical diagnosis neural network would pose to the privacy of individuals' medical data. However, such an application may

[3]Defined in the GDPR as "an identified or identifiable natural person".

become feasible in the near future without infringing on the secrecy of medical data by leveraging Homomorphic Encryption.

Thesis in Context

We focus on Fully Homomorphic Encryption (FHE), which is a technology allowing for computating on encrypted data. Simply put, a client using FHE can ask a third party to process some of their data while keeping the aforementioned data hidden from the third party.

The wide range of uses of neural networks makes them the perfect target for FHE technology. Indeed, successfully and efficiently computing neural networks over encrypted inputs would provide privacy guarantees to a host of applications, notably most Machine Learning as a Service (MLaaS) applications.

Besides, with the help of FHE, the conception of new secure applications requiring sensitive data becomes possible. For instance, we could imagine having a remote medical diagnosis server working without accessing clear information on the patient. This would allow us to earn meaningful feedback on our health without entrusting any third party with our personal medical data. Thus, FHE does not only add privacy to pre-existing applications, but also enables the deployment of new applications previously impossible due to unachievable privacy requirements.

However, evaluating neural networks in the encrypted domain remains challenging. One such challenge comes from the computation of non-linear functions used in neural networks as FHE cryptosystems mainly compute low degree polynomial functions in practice. Another challenge is to compute efficiently a recurrent or deep neural network as FHE parameters are often chosen depending on the multiplicative depth of computation.

Every fully homomorphic encryption scheme so far rely on a concept of noise to ensure security, which can be managed during computation thanks to the "bootstrapping" operation. The bootstrapping procedure of the Fully Homomorphic Encryption over the Torus cryptosystem (TFHE) is of particular interest as it is the most efficient bootstrapping algorithm in the literature. Besides, the bootstrapping operator of TFHE can be specialized to compute specific Look-Up Tables (LUTs) over encrypted inputs with no overhead compared to a simple bootstrapping. It is then called "functional bootstrapping", and is an operator specific to TFHE that can greatly help with the computation of the non-linear part of neural networks.

In this thesis, we explore the details of the Fully Homomorphic Encryption over

the Torus cryptosystem (TFHE) and its functional bootstrapping operator as we aim to make it efficient enough for real life applications.

Contributions Overview

The main contributions of this thesis lie in the following three categories:

Full Torus Functional Bootstrapping: The usual functional bootstrapping technique comes with restrictions either on the size of messages encrypted with a given set of parameters, or on the type of functions it can perform. We leverage this technique to build a new functional bootstrapping operator without such restrictions. In addition, we analyse and compare ourselves to other state of the art methods and achieve better results, in particular regarding the error rate of the bootstrapping procedure.

Circuits in Any Basis: The functional bootstrapping operation only computes functions with one single input. We investigate ways to compute similar Look-Up Tables (LUTs) over multiple inputs and come with an original way to compute circuits using inputs in any basis with TFHE. We compare ourselves to other generic ways to compute multi-inputs LUTs from the literature which highlights the benefits of our method.

Applications: We build neural networks in the encrypted domain with the aim to reduce the latency of computation as much as possible. We investigate different types of structure for neural networks, such as fully connected feedforward neural networks and LSTMs, and analyse the challenges that come with their "homomorphic" implementation. We also investigate how to make FHE accessible to memory constrained devices by mitigating the memory overhead of homomorphic encryption on the device's side thanks to "transciphering".

Outline of the Manuscript

The manuscript is organized in two parts. The first part is related to the state of the art of fully homomorphic encryption and neural networks over encrypted inputs. The second part describes the main contributions of this thesis.

- **Part one:** In Chapter 1, we give a short history of fully homomorphic encryption and describe its potential as a privacy enhancing technology for outsourced computation. We then give a more in depth description of the mathematical concept underlying fully homomorphic encryption and give a short description of three fully homomorphic cryptosystems: BFV, BGV and

CKKS.

In Chapter 2, we give an in depth description of the TFHE cryptosystem which is at the core of this thesis. Notably, we make an emphasis on the noise growth resulting from each operation as it impacts the probability of error of the bootstrapping operation.

In Chapter 3, we describe the basic structure of neural networks and some of the results from the state of the art about the evaluation of neural networks in the homomorphic domain. We highlight an interesting technique to make the evaluation of homomorphic neural networks viable, called transfer learning. However, we show that many challenges still remain to make FHE efficient enough for real life applications of homomorphic neural networks.

- **Part two:** In Chapter 4, we describe a novel full torus functional bootstrapping and compare it to the state of the art. Simply put, the aim is to efficiently compute look-up tables for any encrypted message. Our technique leverages the properties of the standard functional bootstrapping to build a new operator. This operator achieves less error rate than any other functional bootstrapping from the state of the art for a given set of parameters.

  In Chapter 5, we make use of the functional bootstrapping of TFHE to compute logic gates in any basis B. This allows the construction of logic circuits with inputs in the same basis B and gives more flexibility to homomorphic computation. An example highlights the gain from our approach compared to another technique from the state of the art.

  In Chapter 6, we summarize some of our results on concrete applications of the functional bootstrapping. This includes a comparison of cryptosystem for the evaluation of neural networks, building blocks for the computation of a recurrent neural network with FHE, and an efficient transciphering of a stream cipher. These applications span from the beginning of the thesis to its end, which brings to light the evolution of our understanding of the TFHE cryptosystem.

Personal Publications

Papers

- Pierre-Emmanuel Clet, Oana Stan, and Martin Zuber. “BFV, CKKS, TFHE: Which One is the Best for a Secure Neural Network Evaluation in the Cloud?” In: Applied Cryptography and Network Security Workshops - ACNS 2021 Satellite Workshops, Cloud S&P, June 21-24 2021. doi:10.1007/978-3-030-81645-2_16.

- Aymen Boudguiga, Oana Stan, Abdessamad Fazzat, Houda Labiod, and Pierre-Emmanuel Clet. “Privacy Preserving Services for Intelligent Transportation Systems with Homomorphic Encryption”. In: ICISSP 2021. doi: 10.5220/0010349706840693.

- Daphné Trama, Pierre-Emmanuel Clet, Aymen Boudguiga, and Renaud Sirdey. “Building Blocks For LSTM Homomorphic Evaluation With TFHE”. In: Cyber Security, Cryptology, and Machine Learning: 7th International Symposium, CSCML 2023, June 29–30. doi: 10.1007/978-3-031-34671-2_9.

- Pierre-Emmanuel Clet, Aymen Boudguiga, Renaud Sirdey, and Martin Zuber. "ComBo: a Novel Functional Bootstrapping Method for Efficient Evaluation of Nonlinear Functions in the Encrypted Domain". In: AfricaCrypt 2023, Jul. 2023. doi: 10.1007/978-3-031-37586-6_6.

- Adda-Akram Bendoukha, Pierre-Emmanuel Clet, Aymen Boudguiga, and Renaud Sirdey. "Optimized stream-cipher-based transciphering by means of functional-bootstrapping". In: DBSec 2023, Jul. 2023. doi: 10.1007/978-3-031-37586-6_6

- Pierre-Emmanuel Clet, Aymen Boudguiga, and Renaud Sirdey. "Chocobo: Creating Homomorphic Circuit Operating with Functional Bootstrapping in basis B". (in submission).

Communications

- FHE.org (Workshop 2022): Poster - "ComBo: a Novel Functional Bootstrapping Method for Efficient Evaluation of Nonlinear Functions in the Encrypted Domain".
- HES (2023): Poster - "Chocobo: Creating Homomorphic Circuit Operating with Functional Bootstrapping in basis B".
- FHE.org (Workshop 2023): Poster - "Chocobo: Creating Homomorphic Circuit Operating with Functional Bootstrapping in basis B".
- FHE.org: Invited talk (06/07/2023) - TFHE functional bootstrapping over multiple inputs.
- AIcrypt (2023): Talk - "ComBo: a Novel Functional Bootstrapping Method for Efficient Evaluation of Nonlinear Functions in the Encrypted Domain".

Part I

Background

Chapter 1

Introduction to Fully Homomorphic Encryption

1.1 What is Fully Homomorphic Encryption?

Informally, a cryptosystem is an encryption mechanism that depends on a set of parameters p. The parameter set p defines: the set $\mathcal{P}_p$ of possible plaintexts, the set $\mathcal{C}_p$ of possible encryptions, the set $\mathcal{K}_p$ of possible secret and public keys necessary for the scheme, and the sets $\mathcal{E}_p$ and $\mathcal{D}_p$ of procedures to encrypt and decrypt messages. A nuance exists between a message and a plaintext since messages usually need to be mapped into the plaintext space first before being encrypted. However, these two words are sometimes used interchangeably. We say that a message is "in clear" if it is not encrypted.

Some extra properties may exist on top. This is the case in Homomorphic Encryption (HE) schemes where some operations, such as additions or multiplications, can be evaluated on messages while encrypted. Multiple types of HE schemes exist:

- **Partially Homomorphic Encryption:** A cryptosystem that supports the evaluation of one type of operation (either additions or multiplications). This includes cryptosystems such as RSA and ElGamal which can compute multiplications as well as the Paillier cryptosystem which can compute additions. Note that RSA is widely used, notably to encrypt online transactions. However, it is usually coupled with padding schemes depriving RSA from its homomorphic properties [104].
- **Somewhat Homomorphic Encryption (SHE):** A cryptosystem that

supports both the evaluation of additions and multiplications but only for a subset of arithmetic circuits. For instance, Sanders et al., [83] specify such a cryptosystem allowing for the computation of binary circuits in the NC^1 complexity class. The BGV scheme [12] and TFHE scheme [25] are also SHE schemes when used without bootstrapping.

- **Levelled Fully Homomorphic Encryption (LHE):** A cryptosystem that supports the evaluation of both additions and multiplications. The depth of the arithmetic circuits that can be computed, which can be roughly interpreted as the maximum amount of multiplications performed on any given ciphertext, depends on the parameters used for the cryptosystem. Schemes such as BFV [39] and CKKS [23] are all LHE schemes when used without their bootstrapping procedure.

- **Fully Homomorphic Encryption (FHE):** There exists at least one set of parameters so that the cryptosystem supports the evaluation of both additions and multiplications without restrictions. Schemes such as BFV, BGV, CKKS and TFHE are all FHE schemes when used with their bootstrapping procedure.

At first glance, fully homomorphic seems like a strong word to describe a cryptosystem that can only compute additions and multiplications. However, using a binary plaintext space, any logic circuit can be built using additions, multiplications and constants. As such, any function that can be evaluated on a computer can in theory be evaluated with an FHE scheme.

Naturally, one can wonder why we should care about evaluating functions over encrypted inputs. We claim that it cannot only enhance privacy in existing applications, but also open completely new possibilities by challenging previously "obvious" assumptions. Indeed, it seems obvious that in order to use a GPS, our location must be send to another entity. It also seems obvious that in order to have a diagnosis, we must share a part of our medical data with a doctor. But what if it was not actually needed? That is one of the promises of FHE: to be able to receive meaningful answers without anyone else knowing the question. Or asking the shortest path to destination without letting anyone know what is the destination. Indeed, those applications can be seen as a function to compute (finding the answer) over an encrypted input (the question).

One of the most obvious place for FHE to shine is in healthcare since medical data are sensitive data, and spreading one's medical data can lead to undesirable consequences. However, FHE could allow us to have access to fast and secure diagnosis online, even in medical deserts.

The use of FHE could also help generalizing AI as an outsourced helper. ChatGPT has shown the potential of AI chat bots to help in many domains if used correctly. Its applications would be further broaden if sensitive data could be fed to the AI without risks on privacy.

However, FHE does not come for free. Indeed, in the current state of the art, fully homomorphic encryption comes with a massive time overhead (approximately $\times 10,000,000$ on a CPU for logic gates using TFHE [27]) and memory overhead (at least $\times 32,000$ for the encryption of a bit). As such, further research is needed to discover the full potential of FHE and materialize its use in our daily lives to protect our privacy.

1.2 A Short Story of Homomorphic Encryption

The first published scheme with homomorphic properties was the RSA [81] cryptosystem, publicly described in 1978 by Ronald Rivest, Adi Shamir and Leonard Adleman. This cryptosystem has the peculiarity to allow for the evaluation of multiplications on encrypted data. This interesting property soon lead to the following question [80]: is it possible to build a cryptosystem enabling the evaluation of any operation on encrypted data? Which is to say, is it possible to achieve fully homomorphic encryption?

New cryptosystems with homomorphic properties followed the publication of RSA. Notably the ElGamal cryptosystem [38] in 1985 with similar multiplicative properties as RSA, and the Paillier cryptosystem [77] and Goldwasser-Micali cryptosystem [46] with additive properties. In addition to the possibility to compute homomorphic additions with the Paillier cryptosystem, multiplications by a plaintext are also possible, bringing the community one step closer to an FHE scheme. However, the dream of an actual FHE scheme was still out of reach.

Craig Gentry built in 2009 the first FHE scheme [42] thanks to a self-made noisy lattice-based somewhat homomorphic cryptosystem. Noisy means that a random noise term is part of the encryption process, and this happened to be the only limiting factor preventing the scheme to be fully homomorphic. The key component of his thesis is the concept of "bootstrapping" which allows him to reset the noise level of his ciphertexts, making the cryptosystem effectively fully homomorphic. The core idea of the bootstrapping procedure is to homomorphically compute the decryption of a ciphertext, leading to a less noisy encryption of the same message. Gentry's cryptosystem requires a bitwise encryption of each message, allowing the computation of functions in the encrypted domain as binary circuits.

The blueprint of his approach was used as the basis for all the newer FHE cryp-

tosystems. The main improvements came from using cryptosystems based on variants of the Learning With Error problem [79] allowing for more efficient bootstrapping procedures and arithmetic operations on larger plaintext spaces [39, 12, 23, 25]. This notably allows to compute arithmetic circuits much more efficiently. Thus, it has become usual to compute functions in the encrypted domain as low degree approximations instead of generating homomorphic logic circuits, even though this approach remains relevant.

FHE cryptosystems can roughly be grouped in three generations:

- **First generation:** This generation is limited to the lattice-based cryptosystem from Gentry's thesis. It builds the basis for all newer FHE schemes so far. Gentry shows how to turn a somewhat homomorphic encryption scheme into a FHE scheme as long as it is bootstrappable. He also shows the existence of such bootstrappable scheme by building his own lattice based cryptosystem. However, it is not used in practice, notably due to an excessive lack of efficiency.

- **Second generation:** This generation builds on the previous one and brings some much needed improvements regarding efficiency. First, the schemes from the second generation mainly rely on the Learning With Errors problem rather than standard lattice problems, making things easier to implement and allowing for larger plaintext spaces. Furthermore, by leveraging tensoring operations such as the constructions of Aguilar-Melchor et al., [71] and Brakerski [11], the multiplication of ciphertexts becomes less noisy, improving the overall efficiency of FHE cryptosystems. Even better, Single Input Multiple Data (SIMD) operations become doable in the encrypted domain with a technique called "batching".
  The best representatives of this second generation are the BFV and BGV cryptosystem which are based on the Ring Learning With Errors problem. They both allow for efficient evaluation of polynomial functions without requiring binary decomposition of messages. However, one bootstrapping with BFV and BGV usually takes between 1 minute and 30 minutes for 80-bit security parameter sets [40], making it a bottleneck for the efficiency of computation. As such, these cryptosystems are usually used in LHE mode.

- **Third generation:** This generation includes two different types of cryptosystems:

  - The first type includes FHEW and TFHE. They allow for a more efficient bootstrapping procedure than any other schemes. Indeed, one bootstrapping usually takes less than a second and as low as 0.01 second with standard 128-bit security parameter sets [27].

– The second type includes CKKS. It allows evaluating polynomial functions efficiently with approximate arithmetic (float type operations) over batched inputs. One bootstrapping takes between 30 seconds and a couple of minutes with standard 128-bit security parameter sets [55].

1.3 Learning With Errors and Ring Learning With Errors

Learning With Errors [79], abbreviated as LWE, is a mathematical problem introduced by Oded Regev in 2005. He showed the existence of a quantum reduction from some well known worst-case lattice problems, such as the approximate Shortest Vector Problem (Approx-SVP), to LWE. Most recent FHE cryptosystems rely on the hardness of LWE as the basis of their security. The original problem is defined over the following elements:

- Let n be an integer.
- Let p be a prime number.
- Let $(\boldsymbol{a}_i)_{i\in\mathbb{N}} \in (\mathbb{Z}_p^n)^{\mathbb{N}}$ where each $\boldsymbol{a}_i$ are sampled independently and uniformly from $\mathbb{Z}_p^n$.
- Let $\boldsymbol{s} \in \mathbb{Z}_p^n$ be a secret vector. The secret can also be sampled with binary coefficients.
- Let $(e_i)_{i\in\mathbb{N}} \in (\mathbb{Z}_p)^{\mathbb{N}}$ where e_i is sampled from a distribution $\mathcal{X}$. The elements e_i are called error terms or noise.
- Let $(b_i)_{i\in\mathbb{N}} \in (\mathbb{Z}_p)^{\mathbb{N}}$ where $b_i = \boldsymbol{a}_i \cdot \boldsymbol{s} + e_i$.
- A pair $(\boldsymbol{a}_i, b_i)$ is called an $\mathrm{LWE}_{p,\mathcal{X}}$ sample. We may note LWE sample to simplify the notation.

The LWE search problem is to find $\boldsymbol{s}$ given p, n, $(\boldsymbol{a}_i)_{i\in\mathbb{N}}$, and $(b_i)_{i\in\mathbb{N}}$. This problem is considered as difficult even against a quantum attacker when $\mathcal{X}$ is a gaussian distribution with sufficient standard deviation.

The Ring Learning With Errors problem, abbreviated as RLWE problem, is an extension of the LWE problem described and analysed by Stehlé et al., [93] in 2009 and further studied by Lyubashevsky et al., [68] in 2010. It is defined as follows:

- Let N be a power of 2.
- Let k be a positive integer.

- Let $p \equiv 1[2N]$ be a prime number.
- Let $(\boldsymbol{a}_i)_{i\in\mathbb{N}} \in ((\mathbb{Z}_p[X]/(X^N+1))^k)^{\mathbb{N}}$ where each $\boldsymbol{a}_i$ are sampled independently and uniformly from $(\mathbb{Z}_p[X]/(X^N+1))^k$. Note that since N is a power of 2, X^N+1 is cyclotomic.
- Let $\boldsymbol{s} \in (\mathbb{Z}_p[X]/(X^N+1))^k$ be a secret polynomial. The coefficients of the secret $\boldsymbol{s}$ can be restricted to binary values.
- Let $(e_i)_{i\in\mathbb{N}} \in (\mathbb{Z}_p[X]/(X^N+1))^{\mathbb{N}}$ where e_i is sampled from a distribution $\mathcal{X}$ usually a gaussian for each coefficient.
- Let $(b_i)_{i\in\mathbb{N}} \in (\mathbb{Z}_p[X]/(X^N+1))^{\mathbb{N}}$ where $b_i = \boldsymbol{a}_i \cdot \boldsymbol{s} + e_i$.
- A pair $(\boldsymbol{a}_i, b_i)$ is called an $\text{RLWE}_{p,N,\mathcal{X}}$ sample. We may note RLWE sample to simplify the notation.

The RLWE search problem is to find $\boldsymbol{s}$ given p, N, $(\boldsymbol{a}_i)_{i\in\mathbb{N}}$, and $(b_i)_{i\in\mathbb{N}}$. In 2010, Vadim Lyubashevsky, Chris Peikert and Oded Regev proved that RLWE is as hard as the approximation problem Approx-SVP via a quantum reduction [68]. This result supports the hardness of RLWE as Approx-SVP is assumed as a hard problem [72].

We obtain the Torus Learning With Errors problem (respectively Torus Ring Learning With Errors), abbreviated as TLWE problem (respectively TRLWE), simply by switching each $\mathbb{Z}_p$ for $\mathbb{T} = \mathbb{R}/\mathbb{Z}$ in the LWE problem (respectively RLWE problem), and sampling the secret $\boldsymbol{s}$ from $\mathbb{Z}^n$ (respectively $\mathbb{Z}[X]/(X^N+1)$). The TLWE problem (respectively TRLWE problem) is at least as hard as the LWE problem [28] (respectively RLWE problem).

Each search problem is associated to a decision problem. The decision problem is to distinguish between LWE samples and uniformly random samples. The search and decision problems are of equal hardness [79].

In practice, LWE problems are used as the hardness assumption of most FHE schemes. As such, parameters of the LWE problems need to be taken large enough to ensure a sufficient level of security for the cryptosystems. This security is directly linked to the most efficient cryptanalysis techniques found so far in the state of the art, which turns parameter selection into a arduous task. Thankfully, the lattice-estimator [2] is here to give an up-to-date security level approximation of an LWE problem for any parameter set.

1.4 BFV & BGV

1.4.1 BFV

In this section, we give a brief summary of the BFV cryptosystem. The BFV cryptosystem conceived by Fan and Vercauteren [39] is an amelioration of a cryptosystem by Brakerski [11]. It is based on the RLWE problem and can be defined as follows:

- **Plaintext space :** $\mathcal{P} := \mathbb{Z}_t[X]/(X^N+1)$.
- **Ciphertext space :** $\mathcal{C} := \big(\mathbb{Z}_q[X]/(X^N+1)\big)^2$. Note that q must be greater than t. We note $\Delta := \lfloor \frac{q}{t} \rfloor$.
- **SecretKeyGen :** $\boldsymbol{s} \leftarrow \mathcal{U}\big(\mathbb{B}[X]/(X^N+1)\big)$.
- **Encrypt**(m) **:** $(a, [-a\cdot\boldsymbol{s} + \Delta\cdot m + e]_q)$ with $a \leftarrow \mathcal{U}(\mathcal{C})$, $e \leftarrow \mathcal{X}$ where $\mathcal{X}$ is the error distribution, and $[\cdot]_x$ denotes the modulo x operation.
- **Decrypt**(a,b) **:** $\left[\left\lceil \frac{t\cdot[a\cdot\boldsymbol{s}+b]_q}{q} \right\rfloor\right]_t$.

Let us consider the two ciphertexts $c_1 = (a_1, b_1) = (a_1, [-a_1\cdot\boldsymbol{s} + \Delta\cdot m_1 + e_1]_q)$ and $c_2 = (a_2, b_2) = (a_2, [-a_2\cdot\boldsymbol{s} + \Delta\cdot m_2 + e_2]_q)$, and note $q\cdot r_i = a_i\cdot\boldsymbol{s} + b_i - \Delta\cdot m_i - e_i$. The homomorphic addition of c_1 and c_2 can be computed as (a_1+a_2, b_1+b_2).

However, their homomorphic multiplication is much trickier to compute. To that end, we first consider the tuple $(c_0, c_1, c_2) = (a_1\cdot a_2, a_1\cdot b_2 + a_2\cdot b_1, b_1\cdot b_2)$ which holds enough information to decrypt the message $m_1\cdot m_2$. Indeed,

$$\begin{gathered} c_2 + c_1\cdot\boldsymbol{s} + c_0\cdot\boldsymbol{s}^2 \\ = \\ \Delta^2\cdot m_1\cdot m_2 + \Delta\cdot(e_1\cdot m_2 + e_2\cdot m_1) + e_1\cdot e_2 \\ +q\cdot\Delta(r_1\cdot m_2 + r_2\cdot m_1) + q\cdot(r_1\cdot e_2 + r_2\cdot e_1) + q^2\cdot r_1\cdot r_2 \end{gathered}$$

With the right sequence of rescaling and modulo operations, it seems possible to decrypt the message $m_1\cdot m_2$ assuming that the noise terms are small enough. The first rescaling operation is to consider $\left(\left\lceil \frac{t\cdot c_0}{q} \right\rfloor, \left\lceil \frac{t\cdot c_1}{q} \right\rfloor, \left\lceil \frac{t\cdot c_2}{q} \right\rfloor\right)$. Then, denoting $\epsilon_t(q) = 1 - \frac{t\cdot\Delta}{q} \in [0, \frac{t}{q}]$ and $\epsilon = \frac{t}{q}\cdot(c_2 + c_1\cdot\boldsymbol{s} + c_0\cdot\boldsymbol{s}^2) - \left\lceil \frac{t\cdot c_2}{q} \right\rfloor + \left\lceil \frac{t\cdot c_1}{q} \right\rfloor\cdot\boldsymbol{s} + \left\lceil \frac{t\cdot c_0}{q} \right\rfloor\cdot\boldsymbol{s}^2$, we get:

$$\begin{gathered} \left\lceil \frac{t\cdot c_2}{q} \right\rfloor + \left\lceil \frac{t\cdot c_1}{q} \right\rfloor\cdot\boldsymbol{s} + \left\lceil \frac{t\cdot c_0}{q} \right\rfloor\cdot\boldsymbol{s}^2 \\ = \\ \Delta\cdot m_1\cdot m_2 - \Delta\cdot\epsilon_t(q)\cdot m_1\cdot m_2 + (1-\epsilon_t(q))(e_1\cdot m_2 + e_2\cdot m_1) + \frac{t\cdot e_1\cdot e_2}{q} \\ +t\cdot\Delta(r_1\cdot m_2 + r_2\cdot m_1) + t\cdot(r_1\cdot e_2 + r_2\cdot e_1) + t\cdot q\cdot r_1\cdot r_2 + \epsilon \end{gathered}$$

If we can find a ciphertext $c = (a, b)$ so that $[a \cdot \boldsymbol{s} + b]_q = \left\lceil \frac{t \cdot c_2}{q} \right\rfloor + \left\lceil \frac{t \cdot c_1}{q} \right\rfloor \cdot \boldsymbol{s} + \left\lceil \frac{t \cdot c_0}{q} \right\rfloor \cdot \boldsymbol{s}^2$, then c can be decrypted as $[m_1 \cdot m_2 + t(r_1 \cdot m_1 + r_2 \cdot m_1)]_t = [m_1 \cdot m_2]_t$ as long as the noise terms and the ϵ term are small enough and the ciphertext space is large enough compared to the plaintext space. To that end, an encryption of $\boldsymbol{s}^2$, called a relinearisation key, is needed to build the ciphertext c. Indeed, if we note $\mathrm{RK} = (\mathrm{RK}_0, \mathrm{RK}_1)$ the relinearisation key, then

$$\left(c_1 + \left\lceil \frac{t \cdot c_0}{q} \right\rfloor \cdot \mathrm{RK}_0, c_2 + \left\lceil \frac{t \cdot c_0}{q} \right\rfloor \cdot \mathrm{RK}_1\right)$$

is a valid encryption of $m_1 \cdot m_2$. However, this crude relinearisation introduce a lot of extra noise. Hence, more subtle relinearisation techniques are usually used.

Relinearisation 1: The first technique is to decompose $\left\lceil \frac{t \cdot c_0}{q} \right\rfloor = \sum_{i=0}^{\lfloor \log_B(q) \rfloor} c^{(i)} \cdot B^i$ in a given basis B and give encryptions $\mathrm{RK}^{(i)} = (\mathrm{RK}_0^{(i)}, \mathrm{RK}_1^{(i)})$ of $B^i \cdot \boldsymbol{s}^2$ for $i \in [\![0, \log_B(q)]\!]$ as the relinearisation key. Then,

$$\left(c_1 + \sum_{i=0}^{\lfloor \log_B(q) \rfloor} c^{(i)} \cdot \mathrm{RK}_0^{(i)}, c_2 + \sum_{i=0}^{\lfloor \log_B(q) \rfloor} c^{(i)} \cdot \mathrm{RK}_1^{(i)}\right)$$

is an encryption of $m_1 \cdot m_2$ with less noise than the previous naive method.

Relinearisation 2: The second technique relies on a rescaling technique called modulus switching to reduce the noise. The relinearisation key RK is then an encryption of $p \cdot \boldsymbol{s}^2$ using a plaintext space of size $p \cdot q$ for some integer p. Then $\left\lceil \frac{\left\lceil \frac{t \cdot c_0}{q} \right\rfloor \cdot \mathrm{RK}}{p} \right\rfloor + (c_1, c_2)$ is an encryption of $m_1 \cdot m_2$. Note that since the relinearisation key uses a different plaintext space than the other messages, its encryption and security must be analysed separately.

1.4.2 BGV

The BGV cryptosystem [12] developed by Brakerski, Gentry, and Vaikuntanathan is very similar to BFV. It is also based on the RLWE problem and can be defined as follows:

- **Plaintext space :** $\mathcal{P} := \mathbb{Z}_t[X]/(X^N + 1)$.
- **Ciphertext space :** $\mathcal{C} := \left(\mathbb{Z}_q[X]/(X^N + 1)\right)^2$. Note that q must be greater than t.

- **SecretKeyGen :** $\boldsymbol{s} \leftarrow \mathcal{U}\left(\mathbb{B}[X]/(X^N+1)\right)$.
- **Encrypt**(m) **:** $(a, [-a \cdot \boldsymbol{s} + m + t \cdot e]_q)$ with $a \leftarrow \mathcal{U}(\mathcal{C})$ and $e \leftarrow \mathcal{X}$ where $\mathcal{X}$ is the error distribution.
- **Decrypt**(a, b) **:** $[[a \cdot \boldsymbol{s} + b]_q]_t$.

Let us consider the two ciphertexts $c_1 = (a_1, b_1) = (a_1, [-a_1 \cdot \boldsymbol{s} + m_1 + t \cdot e_1]_q)$ and $c_2 = (a_2, b_2) = (a_2, [-a_2 \cdot \boldsymbol{s} + m_2 + t \cdot e_2]_q)$, and note $q \cdot r_i = a_i \cdot \boldsymbol{s} + b_i - m_i - t \cdot e_i$. The homomorphic addition of c_1 and c_2 can be computed as $(a_1 + a_2, b_1 + b_2)$.

Their homomorphic multiplication is similar to that of BFV. We first consider the tuple $(c_0, c_1, c_2) = (a_1 \cdot a_2, a_1 \cdot b_2 + a_2 \cdot b_1, b_1 \cdot b_2)$ and then relinearise it.

Note that:

$$\begin{gathered} c_2 + c_1 \cdot \boldsymbol{s} + c_0 \cdot \boldsymbol{s}^2 \\ = \\ m_1 \cdot m_2 + t \cdot (e_1 \cdot m_2 + e_2 \cdot m_1) + t^2 \cdot e_1 \cdot e_2 \\ +q(r_1 \cdot m_2 + r_2 \cdot m_1) + q \cdot t(r_1 \cdot e_2 + r_2 \cdot e_1) + q^2 \cdot r_1 \cdot r_2 \end{gathered}$$

Which can be decrypted as $[[c_2 + c_1 \cdot \boldsymbol{s} + c_0 \cdot \boldsymbol{s}^2]_q]_t = [m_1 \cdot m_2]_t$ as long as $(e_1 \cdot m_2 + e_2 \cdot m_1) + t \cdot e_1 \cdot e_2$ is small enough. It is then followed by a relinearisation procedure similar to that of BFV.

As opposed to BFV, no scaling is required inside of this multiplication to go back to a valid ciphertext format. However this lead to a quadratic noise growth $t \cdot e_1 \cdot e_2$. A modulus switching is then used to reduce the noise to a linear growth.

Modulus switching: This operation switches the ciphertext modulus from a large integer q to a smaller integer p with less noise under some condition over the cryptosystem parameters. Notably, given a ciphertext $c = (a, b)$ using a ciphertext space of size q, it requires that $[q]_t = [p]_t = 1$, $q > p >> t$ and $||[a \cdot \boldsymbol{s} + b]||$ small enough compared to q. Then, we simply compute $\frac{p}{q}c$ and round it to the closest vector $c' = (a', b')$ for the l^1 norm which satisfies $[c']_t = [c]_t$.

We now verify that c' and c encrypt the same value. We note $q \cdot r = a \cdot \boldsymbol{s} + b - [a \cdot \boldsymbol{s} + b]_q$ and $e_p = a' \cdot \boldsymbol{s} + b' - p \cdot r$. Note that

$$\begin{array}{rcl} ||e_p||_1 & \leq & ||a' \cdot \boldsymbol{s} + b' - \frac{p}{q}(a \cdot \boldsymbol{s} + b)||_1 + ||\frac{p}{q}(a \cdot \boldsymbol{s} + b - q \cdot r)||_1 \\ & \leq & ||a' \cdot \boldsymbol{s} + b' - \frac{p}{q}(a \cdot \boldsymbol{s} + b)||_1 + \frac{p}{q}||(a \cdot \boldsymbol{s} + b - q \cdot r)||_1 \\ & \leq & \quad ||a' \cdot \boldsymbol{s} + b' - \frac{p}{q}(a \cdot \boldsymbol{s} + b)||_1 + \frac{p}{q}||[a \cdot \boldsymbol{s} + b]_q||_1 \end{array}$$

The first term is bounded by definition of c' and can be considered small if $t << p$. The second term is small by compared to p since $||[a \cdot \boldsymbol{s} + b]_q||_1$ is small compared to q by assumption. More details on the bound are given in [12]. As such, parameters

can be chosen so that $||e_p||_1 \leq \frac{p}{2}$, which proves that $a' \cdot \boldsymbol{s} + b' - p \cdot r = e_p = [a' \cdot \boldsymbol{s} + b']_p$. Given the assumptions that $[c']_t = [c]_t$ and $[q]_t = [p]_t$ we get that $[a' \cdot \boldsymbol{s} + b' - p \cdot r]_t = [a \cdot \boldsymbol{s} + b - q \cdot r]_t$ and thus $[[a' \cdot \boldsymbol{s} + b']_p]_t = [[a \cdot \boldsymbol{s} + b]_q]_t$, which finalize the proof that c' and c are encryptions of the same value. Besides, the noise of c' is approximately the noise of c rescaled by $\frac{p}{q}$. If $\frac{q}{p}$ is of similar size to the bound on the noise, we can reduce the noise of a homomorphic multiplication from quadratic to linear at the cost of the size of the ciphertext space.

1.4.3 Batching and Bootstrapping for BFV & BGV

The BFV and BGV are very similar cryptosystems. Their batching and bootstrapping procedures thus hold similar performances. The aim of this section is to give a brief understanding of these two concepts.

Batching: Using the Chinese Remainder Theorem (CRT), we get a ring homomorphism between the plaintext space and a product of smaller spaces. Intuitively, a message in the original space can be seen as a vector of messages in smaller spaces. Each position in this vector is called a slot, and ring operations on the initial plaintext space correspond to slot wise operations on the product of smaller spaces. The parameters t and N define the CRT representation of the plaintext space, and as a consequence, the number and the size of the slots usable for batching. Interactions between slots are also possible thanks to ring automorphisms which effectively apply permutations of the slots of a plaintext message. More details on batching techniques can be found in [44, 91, 43].

Bootstrapping: The aim of this operation is to reduce the noise of a ciphertext even when using batching techniques. Note that rounding operations cannot be applied coefficient wise to end up with a proper result slot wise. The basic structure of the bootstrapping procedure is as follows:

- A parameterized rescaling procedure directly on the coefficient of the ciphertext. It is the only operation which differs between BFV and BGV.
- An inner product relying on the bootstrapping key (an encryption of the secret $\boldsymbol{s}$) to compute $a \cdot \boldsymbol{s} + b$ homomorphically.
- A rescaling procedure.
- A homomorphic rounding.

Both rescaling and the inner product are straightforward operations with FHE cryptosystems. However, the rounding must be applied on the coefficients of the message polynomial and not on the slots. This requires to decompose the rounding operation accordingly:

- Apply a linear transformation to send the coefficients of the message into the slots of a ciphertext (or multiple ciphertexts if needed).
- Approximate the rounding operation on the slots of each ciphertext with polynomials. This step is also called digit extraction and requires the evaluation of multiple polynomials to retrieve recursively each useful digit.
- Reverse the linear transformation.

This procedure usually takes multiple minutes as shown in [40]. However, a faster bootstrapping procedure, called thin bootstrapping, can be computed in about a minute for sparsely packed ciphertexts. More details on both bootstrapping procedure can be found in [41, 40].

1.5 CKKS

1.5.1 The CKKS Cryptosystem

The CKKS cryptosystem [23] developed by Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song, is build with a specific batching technique in mind to create an approximate arithmetic scheme. As such, the message space and the way to encode messages into the plaintext space are directly taken into account in the definition of the scheme. Instead of relying on CRT decomposition over a finite field or ring, they use the CRT on complex numbers and introduce a rounding error, which allows for computation on approximate complex numbers. The cryptosystem can be defined as follows:

- **Message space :** $\mathcal{M} := \mathbb{C}^{\frac{N}{2}}$.
- **Plaintext space :** $\mathcal{P} := \mathbb{Z}_{q_l}[X]/(X^N + 1)$ where $q_l = \prod_{i=1}^{l} p_i \cdot q_0$ for given integers p_i and q_0.
- **Ciphertext space :** $\mathcal{C} := \big(\mathbb{Z}_{q_l}[X]/(X^N + 1)\big)^2$.
- **SecretKeyGen :** $\boldsymbol{s} \leftarrow \mathcal{U}\big(\mathbb{B}[X]/(X^N + 1)\big)$.
- **Encode($\boldsymbol{m}$) :** We note $\omega = e^{\frac{2i\pi}{2N}}$ and P_m the only real polynomial satisfying $\forall$ odd $k \in [\![0, \frac{N}{2} - 1]\!], P_m(\omega^{2\cdot k+1}) = \overline{P_m(\omega^{-(2\cdot k+1)})} = m_k$ where m_k is the k^{th} coefficient of the vector $\boldsymbol{m}$. We denote by Δ a scaling factor which is a parameter defining the precision of computation. Then, Encode($\boldsymbol{m}$) $= \lceil \Delta \cdot P_m \rfloor$.

- **Decode**(P_m) **:** $\boldsymbol{m} = \left(\frac{P_m(\omega^{2\cdot k+1})}{\Delta}\right)_{k\in[\![0,\frac{N}{2}-1]\!]}$.
- **Encrypt**(m) **:** $(a, [-a\cdot \boldsymbol{s} + m + e]_{q_l})$ with $a \leftarrow \mathcal{U}(\mathcal{C})$ and $e \leftarrow \mathcal{X}$ where $\mathcal{X}$ is the error distribution.
- **Decrypt**(a, b) **:** $[a\cdot \boldsymbol{s} + b]_{q_l}$.

Let us consider the two ciphertexts $c_1 = (a_1, b_1) = (a_1, [-a_1\cdot \boldsymbol{s} + m_1 + e_1]_{q_l})$ and $c_2 = (a_2, b_2) = (a_2, [-a_2\cdot \boldsymbol{s} + m_2 + e_2]_{q_l})$, and note $q_l \cdot r_i = a_i \cdot \boldsymbol{s} + b_i - m_i - e_i$. The homomorphic addition of c_1 and c_2 can be computed as $(a_1 + a_2, b_1 + b_2)$.

Their homomorphic multiplication is similar to that of BFV. The only distinction is that the rescaling operation is done by a factor p instead of a factor $\frac{t}{q}$ as in BFV, and that the ciphertext modulus is reduced from q_l to q_{l-1} during the rescaling.

We first consider the tuple $(c_0, c_1, c_2) = (a_1\cdot a_2, a_1\cdot b_2 + a_2\cdot b_1, b_1\cdot b_2)$ and then relinearise it. A rescaling operation, similar to the modulus switching of BGV, is then performed as a noise management step.

Note that:

$$\begin{gathered} c_2 + c_1\cdot \boldsymbol{s} + c_0\cdot \boldsymbol{s}^2 \\ = \\ m_1\cdot m_2 + (e_1\cdot m_2 + e_2\cdot m_1) + e_1\cdot e_2 \\ +q_l(r_1\cdot m_2 + r_2\cdot m_1) + q_l(r_1\cdot e_2 + r_2\cdot e_1) + q_l^2\cdot r_1\cdot r_2 \end{gathered}$$

Which can be decrypted as $[c_2+c_1\cdot \boldsymbol{s}+c_0\cdot \boldsymbol{s}^2]_{q_l} = [m_1\cdot m_2+(e_1\cdot m_2+e_2\cdot m_1)+e_1\cdot e_2]_{q_l}$ where $(e_1\cdot m_2+e_2\cdot m_1)+e_1\cdot e_2$ is a small approximation error compared to $m_1\cdot m_2$. It is then followed by a relinearisation procedure similar to that of BFV.

The rescaling procedure then reduces the size of the ciphertext modulus from q_l to q_{l-1} by multiplying each coefficient by $\frac{1}{p_l}$ and rounding the result. Each p_i are generally chosen close to the scaling factor Δ. A more detailed explanation of the CKKS cryptosystem can be found in [23].

1.5.2 Bootstrapping for CKKS

In the CKKS cryptosystem, the noise is intertwined with the least significant bits of the message. As such, the noise is considered as a part of the message, or more precisely, as an approximation error. Thus, the noise cannot really be reduced. However, the multiplicative depth of a circuit computable with given parameters is limited by the number l of rescaling operations which can be performed. The "bootstrapping" of CKKS aims at switching a small ciphertext modulus $q_{l'}$ back

to a large ciphertext modulus q_l. This renews the ability of the scheme to use rescaling operations, which effectively improves the multiplicative depth of circuits computable with low error approximation. Since it does not reduce the ratio between the noise and the message, it can be considered as an upward modulus switching rather than an actual bootstrapping.

The computation of the CKKS bootstrapping can roughly be summarized as follows:

- Upward modulus switching. No computation is required but an error term multiple of $q_{l'}$ is introduced.
- Coefficient to slot operation. This is performed either as a matrix multiplication or as a FFT like computation.
- Modulo $q_{l'}$ operation. Uses a polynomial approximation of the modulo operation.
- Slot to coefficient operation. This is the inverse of the coefficient to slot operation.

Different approaches are discussed for the slot-coefficient transformations and for the approximation of the modulo operation. However, the latency never reaches lower than 20 seconds for a single bootstrapping in the literature. More details are given in [62, 55, 22, 19] regarding the efficient implementation of each step of this bootstrapping procedure.

Chapter 2

TFHE

The Fully Homomorphic encryption over the Torus cryptosystem [25] or TFHE for short, first described in 2016, is the cryptosystem used at the core of this thesis. It builds on the FHEW cryptosystem [37], and specifically on its accumulator-based bootstrapping, to achieve the fastest bootstrapping procedure from the state of the art. This chapter aims at giving a detailed description of this cryptosystem.

2.1 Specificity and Strengths of TFHE

Two main drawbacks plague the majority of current FHE cryptosystem.

- The first drawback is their inefficient bootstrapping taking dozens of seconds at least, as mentioned in Section 1.2. This limits the usage of bootstrapping to use cases where high latency is allowed. This also means that using those cryptosystems requires specific knowledge on tailoring the set of parameters to each use case in order to avoid using the bootstrapping procedure.
- The second drawback is their inability to compute non polynomial functions without decomposing messages into bits. As such, polynomial approximations must be evaluated, requiring further specific knowledge to find balance between speed of computation and precision of the result.

These two drawbacks also make the computation of high degree polynomials very inefficient. Indeed, they would require either the use of large parameters or the use of the cryptosystem's inefficient bootstrapping, slowing the overall computation down.

The TFHE cryptosystem avoid both of these drawbacks:

- TFHE's bootstrapping is the most efficient of the state of the art. It can be as fast as 10ms, which does not jeopardize its usage even in some use cases requiring relatively low latency.
- TFHE's bootstrapping can become "functional" or "programmable", allowing for the computation of non polynomial functions via Look-Up Tables (LUTs) within its procedure.

Besides, the use of bootstrapping put TFHE as a good candidate to become usable by non expert. Indeed, if we can both standardize secure sets of parameters and find efficient ways to compute any functions under these parameter sets, we could prevent the users from having to tediously find appropriate parameter sets for each of their use cases. It could also lead to the creation of automated tools to transform standard algorithms into homomorphic circuits. Attempts at building such technologies already exist in the form of Cingulata/Armadillo [15], Transpiler from Google [47] and the compiler from the Concrete library [29].

However, the interesting properties of TFHE come with some restrictions. Indeed, TFHE is limited to fairly small plaintext spaces compared to cryptosystems such as CKKS or BFV, and as of now, there is no known practical way to compute TFHE's bootstrapping with batched inputs, even though some research exists on the subject [63].

2.2 Preliminary: Probability

Considering that the TFHE cryptosystem is inherently noisy, a proper analysis of this noise needs to be taken into account to ensure proper decryption of messages as well as to ensure the proper computation of some operations such as the bootstrapping procedure. This noise analysis relies on the following assumption called the independence heuristic and formalised in the TFHE paper [25]:

"**Independence Heuristic:** All the coefficients of the errors of samples that occur in all the linear combinations we consider are independent and concentrated. More precisely, they are σ-subgaussian where σ is the square-root of their variance."

Note that many ciphertexts, notably bootstrapped ciphertexts, may have coefficients with correlated inputs. However, the independence heuristic allows for a much easier study of the noise by assuming that the complex relations between the distribution of the noise of these ciphertexts behave as if they were independent. This heuristic is empirically validated as long as it is not used for the linear combination of a ciphertext with itself. In order to fully understand this heuristic,

let us define some specific terms.

Concentrated Distribution Over $\mathbb{T}$: Due to the modular nature of the torus, it is often hard to define an expectation or a variance for a given distribution such as the uniform distribution over $\mathbb{T}$. However, it becomes much more natural to define these values when the support of the distribution lies on a small interval by considering the distribution over the small interval rather than on the full torus. More formally, a distribution $\mathcal{X}$ is said to be concentrated if its support is almost surely included in a ball of radius $\frac{1}{4}$ over $\mathbb{T}$. Then, the variance of $\mathcal{X}$ is defined as $\mathrm{Var}(\mathcal{X}) = \min_{\mathrm{x}\in\mathbb{T}}(\int_I |x-y|^2 \mathrm{d}\mathcal{X}(y))$ where I is an interval of length $l < \frac{1}{2}$ which contains the support of $\mathcal{X}$. The expectation $\mathbb{E}(\mathcal{X})$ is then the value x for which the minimum in the definition of the variance is reached. These definitions can easily be extended to $\mathbb{T}^n$ and $(\mathbb{T}[X]/(X^N+1))^k$ with the following rules:

- A distribution over $\mathbb{T}^n$ is concentrated if the distributions of the projections over each coefficient are concentrated.
- A distribution over $(\mathbb{T}[X]/(X^N+1))^k$ is concentrated if the distribution of each coefficient of each polynomial is concentrated.
- The expectation of a vector is the vector of its expectations.
- The expectation of a polynomial is the vector of expectations of its coefficients.
- The variance of a vector is the greatest variance among the variances of its coefficients.
- The variance of a polynomial is the greatest variance among the variances of its coefficients.

These variance and expectation follow the same linearity rule as their counterpart over real numbers.

Subgaussian Distribution: In addition to the noise sampled from gaussian distributions for fresh ciphertexts, some rounding operation may appear which leads to additional noise term sampled from uniformly random distributions over small intervals. As such, the study of gaussian distribution is not enough for a precise noise analysis. This is where subgaussian distributions come into play as both gaussian distributions and uniform distributions over small intervals are subgaussian. Intuitively, a subgaussian distribution is a distribution that is somewhat bounded by a gaussian.
More formally, a random variable X sampled from a distribution $\mathcal{X}$ over $\mathbb{R}$ is σ-subgaussian if for all $t > 0, \mathcal{P}_{\mathcal{X}}(|X| \geq t) \leq 2\exp(\frac{-t^2}{2\sigma^2})$. Notably, a gaussian distribution with standard deviation σ and a uniform distribution over $[-\sqrt{3}\sigma, \sqrt{3}\sigma]$ are

both σ-subgaussian. The linear combination of independent subgaussian variables follows similar rules to the linear combination of independent gaussian variables. As such, the sum of two independent variables, one being σ_1-subgaussian and the other being σ_2-subgaussian, gives a $\sqrt{\sigma_1^2 + \sigma_2^2}$-subgaussian variable.
Besides, the definition and linear properties are naturally extended to concentrated distributions over $\mathbb{T}$ as long as the combinations stay concentrated. Note that we cannot expect a proper decryption from a ciphertext if its noise is not concentrated.

2.3 TFHE Cryptosystem

The security assumption of TFHE relies on the hardness of the TLWE and TRLWE problems described in Section 1.3. TFHE relies on three types of encryption respectively called TLWE encryption, TRLWE encryption and TRGSW encryption. But first, a set of parameters must be chosen to instantiate the cryptosystem. The parameters required to define the instantiation of the TLWE and TRLWE problems are the following:

- An integer n defining the size of the TLWE secret key.
- An integer k defining the size of the TRLWE secret key.
- An integer N defining the degree of the polynomial $X^N + 1$ in the TRLWE problem. The integer N must be a power of 2.
- A standard deviation σ_{TLWE} for the gaussian distribution of the TLWE noise.
- A standard deviation σ_{TRLWE} for the gaussian distribution of the TRLWE noise.

Additional parameters are required to define specific operations. The parameters required to define the bootstrapping operation are the following:

- An integer B_g which serves as a decomposition basis.
- An integer l which bounds the size of the decomposition used in the bootstrapping procedure.

Finally, some parameters are required to define a key switching procedure:

- An integer B_{KS} which serves as a decomposition basis.
- An integer t which bounds the size of the decomposition used in the key switching procedure.

From now on, we assume that one such set of parameters is chosen to define the cryptosystem. Let us define the 3 types of encryption used in TFHE.

TLWE Encryption:

- The plaintext space is a discretized subset of $\mathbb{T}$. More specifically, it is of the form $\{0, \frac{1}{p}, \frac{2}{p}, ..., \frac{p-1}{p}\}$ for a given integer p.
- The ciphertext space is $\mathbb{T}^n \times \mathbb{T}$.
- The key space is $\mathbb{Z}^n$. In this thesis we will only consider binary keys.
- The encryption procedure for a given key $\boldsymbol{s}$ and a message m is as follows. Sample $\boldsymbol{a}$ from a uniformly random distribution over $\mathbb{T}^n$. Sample e from a centered gaussian distribution with standard deviation σ_{TLWE}. A TLWE encryption of m is defined as $\text{TLWE}_s(m) = (\boldsymbol{a}, \boldsymbol{a} \cdot \boldsymbol{s} + m + e)$.
- The phase of a TLWE ciphertext given a key $\boldsymbol{s}$ is $\phi_{\boldsymbol{s}}(\boldsymbol{a}, b) = b - \boldsymbol{a} \cdot \boldsymbol{s}$. The decryption procedure rounds the phase of a ciphertext to the nearest element of the plaintext space. This procedure outputs the right result as long as the noise is bounded by $\frac{1}{2p}$.

This can be considered as the main encryption type of TFHE as it is the only one that can be efficiently bootstrapped. In the literature, two types of bootstrapping can be found over TLWE ciphertexts, namely the "gate bootstrapping" which outputs a TLWE ciphertext and the "circuit bootstrapping" which outputs a TRGSW ciphertext. Since the circuit bootstrapping can be seen as multiple gate bootstrappings followed by the construction of a TRGSW ciphertext from multiple TLWE ciphertext, we will only consider the gate bootstrapping as a bootstrapping.

TRLWE Encryption:

- The plaintext space is a discretized subset of $\mathbb{T}[X]/(X^N + 1)$. More specifically, it is composed of polynomials with coefficients in $\{0, \frac{1}{p}, \frac{2}{p}, ..., \frac{p-1}{p}\}$ for a given integer p.
- The ciphertext space is $(\mathbb{T}[X]/(X^N + 1))^k \times \mathbb{T}[X]/(X^N + 1)$.
- The key space is $(\mathbb{Z}[X]/(X^N + 1))^k$.
- The encryption procedure for a given key $\boldsymbol{s}$ and a message m is as follows. Sample $\boldsymbol{a}$ from a uniformly random distribution over $(\mathbb{T}[X]/(X^N+1))^k$. Sample each coefficient of e from a centered gaussian distribution with standard deviation σ_{TRLWE}. A TRLWE encryption of m is defined as $\text{TRLWE}_s(m) = (\boldsymbol{a}, \boldsymbol{a} \cdot \boldsymbol{s} + m + e)$.

- The phase of a TRLWE ciphertext given a key $\boldsymbol{s}$ is $\phi_{\boldsymbol{s}}(\boldsymbol{a}, b) = b - \boldsymbol{a} \cdot \boldsymbol{s}$. The decryption procedure rounds the phase of a ciphertext to the nearest element of the plaintext space. This procedure outputs the right result as long as the coefficients of the noise are bounded by $\frac{1}{2p}$.

In the context of bootstrapping, this can be considered as an auxiliary type of encryption to compute the bootstrapping of TLWE ciphertexts.

TRGSW Encryption:

This type of encryption relies on a so called "gadget matrix" $H \in \mathcal{M}_{(k+1)\cdot l, k+1}(\mathbb{T})$ which can be defined as follows in practice:

Let us note

$$V = \begin{pmatrix} \frac{1}{B_g} \\ \frac{1}{B_g^2} \\ \vdots \\ \frac{1}{B_g^l} \end{pmatrix} \in \mathcal{M}_{l,1}(\mathbb{T}) \text{ and } \mathbf{0} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} \in \mathcal{M}_{l,1}(\mathbb{T})$$

Then

$$H = \begin{pmatrix} V & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & V & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & V \end{pmatrix}$$

- The plaintext space is $\mathbb{Z}[X]/(X^N + 1)$ modulo B_g^l.
- The ciphertext space is $\mathcal{M}_{(k+1)\cdot l, k+1}(\mathbb{T}[X]/(X^N + 1))$. This can also be seen as vectors of TRLWE ciphertexts.
- The key space is $(\mathbb{Z}[X]/(X^N + 1))^k$.
- The encryption procedure for a given key $\boldsymbol{s}$ and a message m is as follows. Sample a vector Z of $(k + 1) \cdot l$ TRLWE samples. A TRGSW encryption of m is then defined as $Z + m \cdot H$.
- The message can be decrypted like a TRLWE ciphertext by using the last line of the matrix and multiplying the result by B_g^l.

In the context of bootstrapping, this can be considered as another auxiliary type of encryption to compute the bootstrapping of TLWE ciphertexts.

2.4 Arithmetic Operations

What would be a cryptosystem without homomorphic properties? Well, not FHE. So let us dive into the homomorphic properties of TFHE.

Additions: It is a very straight forward operation for each cryptosystem. Let us note m_1 and m_2 two messages from the same plaintext space. We also note $(\boldsymbol{a}_1, b_1) \in \text{T(R)LWE}_{\boldsymbol{s}}(m_1)$ and $(\boldsymbol{a}_2, b_2) \in \text{T(R)LWE}_{\boldsymbol{s}}(m_2)$ two ciphertexts with noise e_1 and e_2, respectively. Finally, we note $C_1 = Z_1 + m_1 \cdot H$ and $C_2 = Z_2 + m_2 \cdot H$ two TRGSW ciphertexts.
Then $(\boldsymbol{a}_1 + \boldsymbol{a}_2, b_1 + b_2) = (\boldsymbol{a}_1 + \boldsymbol{a}_2, (\boldsymbol{a}_1 + \boldsymbol{a}_2) \cdot \boldsymbol{s} + (m_1 + m_2) + e_1 + e_2)$ is an encryption of $m_1 + m_2$. The ciphertext $(\boldsymbol{a}_1, b_1 + m_2) = (\boldsymbol{a}_1, \boldsymbol{a}_1 \cdot \boldsymbol{s} + (m_1 + m_2) + e_1)$ is also an encryption of $m_1 + m_2$.

Similarly, $C_1 + C_2 = (Z_1 + Z_2) + (m_1 + m_2) \cdot H$ is an encryption of $m_1 + m_2$. Indeed, the sum of two TRLWE encryption of 0 is a TRLWE encryption of 0, thus $Z = Z_1 + Z_2$ is still a vector of TRLWE encryptions of 0 and $C_1 + C_2 = Z + (m_1 + m_2) \cdot H$ is an encryption of $m_1 + m_2$. We also get that $C_1 + m_2 \cdot H$ is an encryption of $m_1 + m_2$.

In every cases, the addition of two encrypted ciphertexts leads to an additive growth of the resulting noise while the addition of an encrypted ciphertext and a clear text does not lead to any noise growth. Note that additions between ciphertexts must be done between ciphertexts of a same type.

Multiplications: For a given integer m and a ciphertext c of any type encrypting an element m_1 with noise e, it is easy to see that $m \cdot c$ encrypts $m \cdot m_1$ with noise $m \cdot e$. We can see here that the multiplication of a clear message and an encrypted message leads to a noise growth as opposed to the addition of a clear message and a ciphertext.

Let us note $C = Z + m_1 \cdot H$ a TRGSW ciphertext and $\boldsymbol{c} = (\boldsymbol{a}, b)$ a TRLWE encryption of a message m with noise e, both under the same key $\boldsymbol{s} = (s_1, ..., s_k)$. We note a_i the polynomials which are the coefficients of the vector $\boldsymbol{a}$. These polynomials as well as b can be decomposed in basis B_g in the format $a_i = \sum_{j=1}^{l} a_{i,j} B_g^{-j} + \epsilon_i$ and $b = \sum_{j=1}^{l} b_j B_g^{-j} + \epsilon_b$ where each $a_{i,j}$ and b_j are polynomials with coefficients bounded by $\frac{B_g}{2}$. Besides, each ϵ_x term is a polynomial with coefficients bounded by $\frac{1}{2B_g^l}$. We note $\tilde{a}_i = a_i - \epsilon_i$ and $\tilde{b} = b - \epsilon_b$. We define the following decomposition procedure

over TRLWE ciphertexts:

$$\mathrm{Dec}(\boldsymbol{c}) = (a_{1,1}, a_{1,2}, ..., a_{k,l}, b_1, ..., b_l)$$

which is effectively a rounded decomposition of each a_i and b in basis B_g with coefficients in $[\![-\frac{B_g}{2}, \frac{B_g}{2} - 1]\!]$. Let us note Z_i the i^{th} line of Z so that each Z_i is a TRLWE sample with noise e_i, and $S = \sum_{i=1}^{k}\sum_{j=1}^{l} a_{i,j} Z_{i\cdot l+j} + \sum_{j=1}^{l} b_j Z_{k\cdot l+j}$.

Then,

$$\begin{aligned}
\mathrm{Dec}(\boldsymbol{c}) \cdot C &= \mathrm{Dec}(\boldsymbol{c}) \cdot (Z + m_1 \cdot H) \\
&= \mathrm{Dec}(\boldsymbol{c}) \cdot Z + m_1 \cdot \mathrm{Dec}(\boldsymbol{c}) \cdot H \\
&= \sum_{i=1}^{k}\sum_{j=1}^{l} a_{i,j} Z_{i\cdot l+j} + \sum_{j=1}^{l} b_j Z_{k\cdot l+j} + m_1 \cdot (\tilde{a}_1, ..., \tilde{a}_k, \tilde{b}) \\
&= S + m_1 \cdot (\epsilon_1, ..., \epsilon_k, \epsilon_b) + m_1 \cdot (\boldsymbol{a}, b)
\end{aligned}$$

Note that S is a linear combination of TRLWE encryptions of 0, so S is also a TRLWE encryption of 0. Besides, $m_1 \cdot (\epsilon_1, ..., \epsilon_k, \epsilon_b)$ can be seen as an error term. Finally, $m_1 \cdot (\boldsymbol{a}, b)$ is an encryption of $m_1 \cdot m$. As such, $\mathrm{Dec}(\boldsymbol{c}) \cdot C$ gives an encryption of $m_1 \cdot m$. This operation is called external product as it is performed between two different types of ciphertexts.

Besides, the resulting noise of this procedure is given by:

$$\sum_{i=1}^{k}\sum_{j=1}^{l} a_{i,j} \cdot e_{i\cdot l+j} + \sum_{j=1}^{l} b_j \cdot e_{k\cdot l+j} + m_1 \cdot (\epsilon_b - \sum_{i=1}^{k} \epsilon_i \cdot s_i) + m_1 \cdot e$$

Note that the coefficients of each ϵ_x term can be considered as random terms sampled independently from a uniform distribution in $[-\frac{1}{2B_g^l}, \frac{1}{2B_g^l}]$ with standard deviation $\frac{1}{2\sqrt{3}B_g^l}$. As such, they are $\frac{1}{2\sqrt{3}B_g^l}$-subgaussian. Considering that each error term e_i follows a gaussian distribution with variance at most V_{TRGSW} and e follows a gaussian distribution with variance at most V_{TRLWE}, we get that each coefficient of the noise of the output follows a gaussian distribution with variance at most:

$$(k+1) \cdot l \cdot N \cdot \left(\frac{B_g}{2}\right)^2 \cdot V_{TRGSW} + ||m_1||_2^2 \cdot \left(\frac{1 + k \cdot N}{12 B_g^{2l}}\right) + ||m_1||_2^2 \cdot V_{TRLWE}$$

Note that this result improves slightly on the original TFHE paper [25] since they considered a variance of $\frac{1}{4B_g^{2l}}$ instead of $\frac{1}{12B_g^{2l}}$ for the ϵ_i terms.

Besides, this bound heavily rely on the independence assumption. Notably to compute the variance of each coefficient $m_1 \cdot \epsilon_i \cdot s_i$. We can first consider that each $\epsilon_i \cdot s_i$ is a random variable with independent coefficients and variances bounded by $\frac{N}{12B_g^{2l}}$, then multiply by m_1 to get a variance bounded by $||m_1||_2^2 \frac{N}{12B_g^{2l}}$. But in truth, the coefficients of $\epsilon_i \cdot s_i$ are not independent and the best bound we would find for $m_1 \cdot \epsilon_i \cdot s_i$ is $||m_1||_1^2 \frac{N}{12B_g^{2l}}$. Since this bound is identical when m_1 is a constant, the choice of $||\cdot||_1$ or $||\cdot||_2$ will not impact any results in this thesis.

Considering that a TRGSW ciphertext is actually a vector of TRLWE ciphertexts, it is possible to multiply two TRGSW ciphertexts with the same technique. This operation is then called internal multiplication.

2.5 Advanced operations

In this section, we describe the non arithmetic operation supported in the TFHE cryptosystem. Notably, we describe its efficient bootstrapping procedure and show how it can become "functional".

Keyswitching: The goal of this operation is to change a TLWE encryption of a message m under a key $\boldsymbol{s}_1$ to a T(R)LWE encryption of m under another key $\boldsymbol{s}_2$. It can additionally be tweaked to compute a linear combination of multiple ciphertexts during its computation. When the linear computation performed during the keyswitch is publicly known, it is referred as public functional keyswitch. Otherwise, it is referred as private functional keyswitch. The description of this two procedures are given in Algorithms 1 and 2. Note that the output of the keyswitch can be a TLWE ciphertext if $N = 1$. Besides, we will simply call keyswitch the public functional keyswitch used when $f : \begin{cases} \mathbb{T} & \to & \mathbb{T} \\ x & \mapsto & x \end{cases}$.

Let us prove that Algorithm 1 outputs the right value and analyse the noise of its output. To that end, we note $\tilde{e}_i = a_i - \tilde{a}_i$, $\boldsymbol{b} = (b^{(1)}, ..., b^{(p)})$, $e_{i,j}$ the noise term in each $\mathrm{KS}_{i,j}$ and e_i the noise in each $\boldsymbol{c}^{(i)}$. Then we get the following formulas:

Algorithm 1 Public Functional Key Switching

Input: p TLWE ciphertexts $\boldsymbol{c}^{(i)} = (\boldsymbol{a}^{(i)}, b^{(i)}) \in \mathrm{TLWE}_{\boldsymbol{s}}(m_i)$, a public morphism $f : \mathbb{T}^p \to \mathbb{T}[X]/(X^N+1)$, a keyswitch key $\mathrm{KS}_{i,j} \in \mathrm{T(R)LWE}_{\boldsymbol{k}}(\frac{s_i}{B_{KS}^j})$ for $i \in [\![1, n]\!]$ and $j \in [\![1, t]\!]$.

Output: A T(R)LWE ciphertext $\boldsymbol{c} \in \mathrm{T(R)LWE}_{\boldsymbol{k}}(f(m_1, ..., m_p))$

1: **for** $i \in [\![1, n]\!]$ **do**

2: $\quad a_i := f(a_i^{(1)}, ..., a_i^{(p)})$

3: $\quad$ Let $\tilde{a}_i$ be the closest multiple of B_{KS}^{-t} to a_i

4: $\quad$ Let $\displaystyle\sum_{j=1}^{t} \frac{\tilde{a}_{i,j}}{B_{KS}^j}$ be the decomposition of $\tilde{a}_i$ in basis B_{KS} with $\tilde{a}_{i,j} \in [\![-\frac{B_{KS}}{2}, \frac{B_{KS}}{2} - 1]\!]$

5: **Return** $\left(\mathbf{0}, f(b^{(1)}, ..., b^{(p)})\right) - \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j} \cdot \mathrm{KS}_{i,j}$

$$
\begin{aligned}
\phi_{\boldsymbol{k}}(\boldsymbol{c}) &= f(\boldsymbol{b}) - \sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j} \cdot \phi_{\boldsymbol{k}}\left(\mathrm{KS}_{i,j}\right) \\
&= f(\boldsymbol{b}) - \sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j} \cdot \left(\frac{s_i}{B_{KS}^j} + e_{i,j}\right) \\
&= f(\boldsymbol{b}) - \sum_{i=1}^{n} \tilde{a}_i \cdot s_i - \sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j} \cdot e_{i,j} \\
&= f(\boldsymbol{b}) - \sum_{i=1}^{n} (a_i - \tilde{e}_i) \cdot s_i - \sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j} \cdot e_{i,j} \\
&= f(\boldsymbol{b}) - \sum_{i=1}^{n} a_i \cdot s_i + \sum_{i=1}^{n} \tilde{e}_i \cdot s_i - \sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j} \cdot e_{i,j} \\
&= f(b^{(1)}, ..., b^{(p)}) - \sum_{i=1}^{n} f(a_i^{(1)}, ..., a_i^{(p)}) \cdot s_i + \sum_{i=1}^{n} \tilde{e}_i s_i - \sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j} e_{i,j} \\
&= f\left(b^{(1)} - \sum_{i=1}^{n} a_i^{(1)} s_i, ..., b^{(p)} - \sum_{i=1}^{n} a_i^{(p)} s_i\right) + \sum_{i=1}^{n} \tilde{e}_i s_i - \sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j} e_{i,j} \\
&= f\left(m_1, ..., m_p\right) + f\left(e_1, ..., e_p\right) + \sum_{i=1}^{n} \tilde{e}_i \cdot s_i - \sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j} \cdot e_{i,j}
\end{aligned}
$$

Thus, $\boldsymbol{c} \in \mathrm{T(R)LWE}_{\boldsymbol{k}}\left(f(m_1, ..., m_p)\right)$ with noise:

$$f\left(e_1, ..., e_p\right) + \sum_{i=1}^{n} \tilde{e}_i \cdot s_i - \sum_{i=1}^{n} \sum_{j=1}^{t} \tilde{a}_{i,j} \cdot e_{i,j}$$

By definition of $\tilde{a}_i$, each coefficient of $\tilde{e}_i$ is in $[-\frac{1}{2B_{KS}^t}, \frac{1}{2B_{KS}^t}]$ and as such are $\frac{1}{2\sqrt{3}B_{KS}^t}$-subgaussian. Thus, given R so that f is R-Lipschitz, we get the following bound on the noise variance of $\boldsymbol{c}$:

$$\mathrm{Var}(\boldsymbol{c}) \leq R^2 \cdot \mathrm{Var}(c^{(1)}, ..., c^{(p)}) + \frac{n}{12B_{KS}^{2t}} + n \cdot N \cdot t \cdot \left(\frac{B_{KS}}{2}\right)^2 \cdot \mathrm{Var}_{KS}$$

where Var_{KS} is the maximum variance of the ciphertexts making up the keyswitch key.

Note that Algorithm 1 and its noise formula improve on the result from the original TFHE paper [25] on multiple levels. Indeed, they only considered the binary decomposition basis, which make our result more general, and they bounded the error terms $\tilde{e}_i$ by considering a standard deviation of $\frac{1}{2B_{KS}^t}$ rather than $\frac{1}{2\sqrt{3}B_{KS}^t}$.

Algorithm 2 Private Functional Key Switching

Input: p TLWE ciphertexts $\boldsymbol{c}^{(i)} = (c_1^{(i)}, ..., c_{n+1}^{(i)}) \in \mathrm{TLWE}_{\boldsymbol{s}}(m_i)$, a keyswitch key $\mathrm{KS}_{i,j,q}^{(f)} \in \mathrm{T(R)LWE}_{\boldsymbol{k}}(f(0, ..., 0, \frac{s_i}{B_{KS}^j}, 0, ..., 0))$ for $i \in [\![1, n+1]\!]$, $j \in [\![1, t]\!]$ and $q \in [\![1, p]\!]$ with $s_{n+1} = -1$ by convention.

Output: A T(R)LWE ciphertext $\boldsymbol{c} \in \mathrm{T(R)LWE}_{\boldsymbol{k}}(f(m_1, ..., m_p))$

1: **for** $i \in [\![1, n+1]\!]$ **do**
2: **for** $q \in [\![1, p]\!]$ **do**
3: Let $\tilde{c}_i^{(q)}$ be the closest multiple of B_{KS}^{-t} to $c_i^{(q)}$
4: Let $\sum_{j=1}^{t} \frac{\tilde{c}_{i,j}^{(q)}}{B_{KS}^j}$ be the decomposition of $\tilde{c}_i^{(q)}$ in basis B_{KS} with $\tilde{c}_{i,j}^{(q)} \in [\![-\frac{B_{KS}}{2}, \frac{B_{KS}}{2} - 1]\!]$
5: **Return** $-\sum_{q=1}^{p} \sum_{i=1}^{n+1} \sum_{j=1}^{t} \tilde{c}_{i,j}^{(q)} \cdot \mathrm{KS}_{i,j,q}^{(f)}$

Let us now prove that Algorithm 2 outputs the right value and analyse the noise of its output. To that end, we note $\tilde{e}_{i,q} = c_i^{(q)} - \tilde{c}_i^{(q)}$, $e_{i,j,q}$ the noise term in each $\mathrm{KS}_{i,j,q}^{(f)}$ and e_i the noise in each $\boldsymbol{c}^{(i)}$. Finally, let us note $f_q(x) := f(0, ..., 0, x, 0, ..., 0)$ where x is in the q^{th} position. Then we get the following formulas:

$$
\begin{aligned}
\phi_{\boldsymbol{k}}(\boldsymbol{c}) &= -\sum_{q=1}^{p}\sum_{i=1}^{n+1}\sum_{j=1}^{t}\tilde{c}_{i,j}^{(q)}\cdot\phi_{\boldsymbol{k}}\left(\mathrm{KS}_{i,j,q}^{(f)}\right)\\
&= -\sum_{q=1}^{p}\sum_{i=1}^{n+1}\sum_{j=1}^{t}\tilde{c}_{i,j}^{(q)}\cdot\left(f_q(\frac{s_i}{B_{KS}^{j}})+e_{i,j,q}\right)\\
&= -\sum_{q=1}^{p}\sum_{i=1}^{n+1}\sum_{j=1}^{t}f_q\left(\tilde{c}_{i,j}^{(q)}\cdot\frac{s_i}{B_{KS}^{j}}\right)-\sum_{q=1}^{p}\sum_{i=1}^{n+1}\sum_{j=1}^{t}\tilde{c}_{i,j}^{(q)}\cdot e_{i,j,q}\\
&= -\sum_{q=1}^{p}\sum_{i=1}^{n+1}f_q\left(\tilde{c}_{i}^{(q)}\cdot s_i\right)-\sum_{q=1}^{p}\sum_{i=1}^{n+1}\sum_{j=1}^{t}\tilde{c}_{i,j}^{(q)}\cdot e_{i,j,q}\\
&= -\sum_{q=1}^{p}\sum_{i=1}^{n+1}f_q\left((c_{i}^{(q)}-\tilde{e}_{i,q})\cdot s_i\right)-\sum_{q=1}^{p}\sum_{i=1}^{n+1}\sum_{j=1}^{t}\tilde{c}_{i,j}^{(q)}\cdot e_{i,j,q}\\
&= -\sum_{q=1}^{p}\sum_{i=1}^{n+1}f_q\left(c_{i}^{(q)}\cdot s_i\right)+\sum_{q=1}^{p}\sum_{i=1}^{n+1}f_q\left(\tilde{e}_{i,q}\cdot s_i\right)-\sum_{q=1}^{p}\sum_{i=1}^{n+1}\sum_{j=1}^{t}\tilde{c}_{i,j}^{(q)}\cdot e_{i,j,q}\\
&= \sum_{q=1}^{p}f_q\left(c_{n+1}^{(q)}-\sum_{i=1}^{n}c_{i}^{(q)}\cdot s_i\right)+\sum_{q=1}^{p}\sum_{i=1}^{n+1}f_q\left(\tilde{e}_{i,q}\cdot s_i\right)-\\
&\quad\sum_{q=1}^{p}\sum_{i=1}^{n+1}\sum_{j=1}^{t}\tilde{c}_{i,j}^{(q)}\cdot e_{i,j,q}\\
&= \sum_{q=1}^{p}f_q\left(m_q+e_q\right)+\sum_{q=1}^{p}\sum_{i=1}^{n+1}f_q\left(\tilde{e}_{i,q}\cdot s_i\right)-\sum_{q=1}^{p}\sum_{i=1}^{n+1}\sum_{j=1}^{t}\tilde{c}_{i,j}^{(q)}\cdot e_{i,j,q}\\
&= f(m_1,...,m_p)+f(e_1,...,e_p)+\sum_{q=1}^{p}\sum_{i=1}^{n+1}f_q\left(\tilde{e}_{i,q}\cdot s_i\right)-\\
&\quad\sum_{q=1}^{p}\sum_{i=1}^{n+1}\sum_{j=1}^{t}\tilde{c}_{i,j}^{(q)}\cdot e_{i,j,q}
\end{aligned}
$$

Thus, $\boldsymbol{c}\in\mathrm{T(R)LWE}_{\boldsymbol{k}}\left(f(m_1,...,m_p)\right)$ with noise:

$$
f(e_1,...,e_p)+\sum_{q=1}^{p}\sum_{i=1}^{n+1}f_q\left(\tilde{e}_{i,q}\cdot s_i\right)-\sum_{q=1}^{p}\sum_{i=1}^{n+1}\sum_{j=1}^{t}\tilde{c}_{i,j}^{(q)}\cdot e_{i,j,q}
$$

By definition of $\tilde{c}_{i}^{(q)}$, each coefficient of $\tilde{e}_{i,q}$ is in $[-\frac{1}{2B_{KS}^{t}},\frac{1}{2B_{KS}^{t}}]$ and as such are $\frac{1}{2\sqrt{3}B_{KS}^{t}}$-subgaussian. Besides, $\sum_{q=1}^{p}\sum_{i=1}^{n+1}f_q\left(\tilde{e}_{i,q}\cdot s_i\right)=f\left(\sum_{i=1}^{n+1}\tilde{e}_{i,1}\cdot s_i,...,\sum_{i=1}^{n+1}\tilde{e}_{i,p}\cdot s_i\right)$.

Thus, given R so that f is R-Lipschitz, we get the following bound on the noise variance of $\boldsymbol{c}$:

$$\mathrm{Var}(\boldsymbol{c}) \leq R^2 \cdot \mathrm{Var}(c^{(1)}, ..., c^{(p)}) + R^2 \cdot \frac{n+1}{12B_{KS}^{2t}} + p \cdot (n+1) \cdot t \cdot \left(\frac{B_{KS}}{2}\right)^2 \cdot \mathrm{Var}_{KS}$$

where Var_{KS} is the maximum variance of the ciphertexts making up the keyswitch key.

This algorithm and noise formula give similar improvement to the TFHE paper as in the public keyswitch case.

Extraction: Each coefficient of a message in $\mathbb{T}[X]/(X^N+1)$ can be seen as a message in $\mathbb{T}$. Similarly, we can extract TLWE ciphertexts from a TRLWE ciphertext. Insight can be found in the evaluation of the phase of a TRLWE ciphertext. Given $\boldsymbol{c} = (a_1, ..., a_k, b) \in \mathrm{TRLWE}_{\boldsymbol{s}}(m)$ with $m = \sum_{i=0}^{N-1} m_i \cdot X^i$, we get:

$$\phi_{\boldsymbol{s}}(\boldsymbol{c}) \quad = \quad b - \sum_{i=1}^{k} a_i \cdot s_i$$

If we note $a_i = \sum_{j=0}^{N-1} a_{i,j} \cdot X^j$, $s_i = \sum_{j=0}^{N-1} s_{i,j} \cdot X^j$, $b = \sum_{j=0}^{N-1} b_j \cdot X^j$, and consider the q^{th} coefficient of $\phi_{\boldsymbol{s}}(\boldsymbol{c})$ we get:

$$\phi_{\boldsymbol{s}}(\boldsymbol{c})_q \quad = \quad b_q - \sum_{i=1}^{k} \sum_{j=0}^{q} a_{i,q-j} \cdot s_{i,j} + \sum_{i=1}^{k} \sum_{j=q+1}^{N-1} a_{i,N+q-j} \cdot s_{i,j}$$

From this, we get that for any $q \in [\![0, N-1]\!]$, we can build a TLWE encryption of m_q under the key $\boldsymbol{s}$ seen as a vector $(s_{1,0}, ..., s_{1,N-1}, ..., s_{k,N-1}) \in \mathbb{B}^{k \cdot N}$ as

$$\mathrm{TLWE}_{\boldsymbol{s}}(m_q) = (\epsilon_{1,0} \cdot a_{1,0}, ..., \epsilon_{k,N-1} \cdot a_{k,N-1}, b_q)$$

where $\epsilon_{i,j} = \begin{cases} 1 \text{ if } j \leq q \\ -1 \text{ otherwise} \end{cases}$ and with no additional noise compared to the TRLWE ciphertext it is extracted from.

CMux gate: The CMux gate is used to make a homomorphic selection between two encrypted messages given an encrypted bit. Given $\boldsymbol{C} \in \mathrm{TRGSW}_{\boldsymbol{s}}(B)$ with $B \in \mathbb{B}$, $\boldsymbol{c}_1 \in \mathrm{TRLWE}_{\boldsymbol{s}}(m_1)$, and $\boldsymbol{c}_2 \in \mathrm{TRLWE}_{\boldsymbol{s}}(m_2)$, the CMux gate verifies:

$$\mathrm{CMux}(\boldsymbol{C}, \boldsymbol{c}_2, \boldsymbol{c}_1) \in \begin{cases} \mathrm{TRLWE}_{\boldsymbol{s}}(m_1) \text{ if } B = 0 \\ \mathrm{TRLWE}_{\boldsymbol{s}}(m_2) \text{ if } B = 1 \end{cases}$$

Simply put, if $\boldsymbol{C}$ is an encryption of 0, the CMux gate outputs a new encryption of m_1, whereas if $\boldsymbol{C}$ is an encryption of 1, the CMux gate outputs a new encryption of m_2. It is computed using the external product as follows:

$$\text{CMux}(\boldsymbol{C}, \boldsymbol{c}_2, \boldsymbol{c}_1) = \text{Dec}\,(c_2 - c_1) \cdot \boldsymbol{C} + c_1$$

The variance of the noise from this operation can easily be inferred from the noise of an external multiplication:

$$\text{Var}\,(\text{CMux}(\boldsymbol{C}, \boldsymbol{c}_2, \boldsymbol{c}_1)) \leq (k+1) \cdot l \cdot N \cdot \left(\frac{B_g}{2}\right)^2 \cdot V_{TRGSW} + \left(\frac{1 + k \cdot N}{12 B_g^{2l}}\right) + V_{TRLWE}$$

where V_{TRLWE} is the maximum between the variance of c_1 and c_2.

BlindRotate: The goal of the BlindRotate is to perform a hidden rotation of the coefficients of the message encrypted in a TRLWE ciphertext. More specifically, given a ciphertext $\text{TLWE}_{\boldsymbol{s}}(m)$ and a ciphertext $\text{TRLWE}_{\boldsymbol{k}}(P)$, the BlindRotate operation aims at computing an encryption of $X^{-\lceil 2N \cdot m \rfloor} \cdot P$.

Algorithm 3 Blind Rotation

Input: a TLWE ciphertext $(a_1, ..., a_n, b) \in \text{TLWE}_{\boldsymbol{s}}(m)$, a TRLWE ciphertext $\boldsymbol{c}_{\text{in}} \in \text{TRLWE}_{\boldsymbol{k}}(P)$, and a bootstrapping key $(\text{BK}_i \in \text{TRGSW}_{\boldsymbol{k}}(s_i))_{i \in [\![1,n]\!]}$.

Output: A ciphertext $\text{ACC} \in \text{TRLWE}_{\boldsymbol{k}}(X^{-\rho} \cdot P)$ where $\rho = \lceil 2N \cdot b \rfloor - \sum_{i=1}^{n} \lceil 2N \cdot a_i \cdot s_i \rfloor$

1: $\overline{b} := \lceil 2N \cdot b \rfloor$
2: **for** $i \in [\![1, n]\!]$ **do**
3: $\quad \overline{a}_i := \lceil 2N \cdot a_i \rfloor$
4: $\text{ACC} := X^{-\overline{b}} \cdot c_{\text{in}}$
5: **for** $i \in [\![1, n]\!]$ **do**
6: $\quad \text{ACC} = \text{CMux}\,(\text{BK}_i, X^{\overline{a}_i} \cdot \text{ACC}, \text{ACC})$
7: **Return** ACC

Algorithm 3 describes how to leverage the CMux gate operation to compute a BlindRotate. We can easily see that at any time t in the for loop, ACC is set to $X^{-\rho_t} \cdot c_{\text{in}}$ where $\rho_t = \sum_{i=1}^{t} (\overline{a}_i \cdot s_i) - b$. As such, by the end of Algorithm 3, ACC is set to $X^{-\rho} \cdot c_{\text{in}} \in \text{TRLWE}_{\boldsymbol{s}}\,(X^{-\rho} \cdot P)$ as expected. Since multiplying a TRLWE ciphertext by X^x does not increase the noise of the ciphertext, the noise of the output of a Blind Rotation can be inferred easily from the noise of the CMux gate

operation. Hence, the variance of the error of a BlindRotate is bounded by:

$$\mathrm{Var}(\mathrm{ACC}) \leq n \cdot \left((k+1) \cdot l \cdot N \cdot \left(\frac{B_g}{2} \right)^2 \cdot V_{TRGSW} + \left(\frac{1 + k \cdot N}{12 B_g^{2l}} \right) \right) + V_{TRLWE}$$

Bootstrapping: Thanks to all the previous operations described in this section, we can now describe the noise management operation called bootstrapping in Algorithm 4. Note that this algorithm is tailored for a binary plaintext space.

Algorithm 4 Bootstrapping

Input: a TLWE ciphertext $\boldsymbol{c} \in \mathrm{TLWE}_{\boldsymbol{s}}(m)$ with $m \in \{0, \frac{1}{2}\}$, a bootstrapping key $\mathrm{BK} = (\mathrm{BK}_i \in \mathrm{TRGSW}_{\boldsymbol{k}}(s_i))_{i \in [\![1,n]\!]}$, a keyswitch key $\mathrm{KS} = \left(\mathrm{KS}_{i,j} \in \mathrm{TLWE}_{\boldsymbol{s}} \left(\frac{k_i}{B_{KS}^j} \right) \right)_{i \in [\![1,n]\!]; j \in [\![1,t]\!]}$.

Output: A ciphertext $\boldsymbol{c}_{\mathrm{out}} \in \mathrm{TLWE}_{\boldsymbol{s}}(m)$ with noise small and independent from $\boldsymbol{c}$.

1: $P = (\mathbf{0}, X^{\frac{N}{2}} \sum_{i=0}^{N-1} \frac{1}{4} \cdot X^i)$
2: ACC= BlindRotate$(\boldsymbol{c}, P, \mathrm{BK})$
3: Let $\boldsymbol{c}_{\mathrm{extr}}$ be the result of the extraction of the first coefficient of ACC
4: $\boldsymbol{c}_{\mathrm{out}} = \mathrm{KeySwitch}(\boldsymbol{c}_{\mathrm{extr}}, \mathrm{KS})$
5: $\boldsymbol{c}_{\mathrm{out}} = \boldsymbol{c}_{\mathrm{out}} + \left(\mathbf{0}, \frac{1}{4}\right)$
6: **Return** $\boldsymbol{c}_{\mathrm{out}}$

Let us note

$$\mathrm{Var}_{\mathrm{KS}} = \frac{N}{12 B_{KS}^{2t}} + N \cdot t \cdot \left(\frac{B_{KS}}{2} \right)^2 \cdot \mathrm{Var}_{KS}$$

the noise introduced by the Keyswitch procedure from key $\boldsymbol{k}$ to $\boldsymbol{s}$, and

$$\mathrm{Var}_{\mathrm{BR}} = n \cdot \left((k+1) \cdot l \cdot N \cdot \left(\frac{B_g}{2} \right)^2 \cdot V_{TRGSW} + \left(\frac{1 + k \cdot N}{12 B_g^{2l}} \right) \right)$$

the noise added by the BlindRotate procedure. Considering that the initial TRLWE ciphertext $(\mathbf{0}, X^{\frac{N}{2}} \sum_{i=0}^{N-1} \frac{1}{4} \cdot X^i)$ is noiseless, and that the identity function is 1-Lipschitz, we find the following bound on the variance of the output of the bootstrapping procedure:

$$\mathrm{Var}(\boldsymbol{c}_{\mathrm{out}}) \leq \mathcal{E}_{BR} + \mathcal{E}_{KS}$$

Besides, the result is an encryption of the right message only if the first coefficient of $X^{-\rho} \cdot P$, where $\rho = \lceil 2N \cdot b \rfloor - \sum_{i=1}^{n} \lceil 2N \cdot a_i \cdot s_i \rfloor$, is the same as the first coefficient of $X^{-2N \cdot m} \cdot P$ in $\mathbb{T}[X]/(X^N + 1)$. This is true only if $|\rho - 2N \cdot m| < \frac{N}{2}$ which introduces a probability of error to this algorithm.

It can be seen that the bootstrapping procedure actually is a look-up table evaluation where the look-up table is defined by P and the index of the table is defined by ρ. As such, for any TRLWE ciphertext $Q = \left(\mathbf{0}, \sum_{i=0}^{N-1} p_i \cdot X^i\right)$, a bootstrapping procedure can be defined by replacing P with Q, achieving the computation of the table

$$(p_0, ..., p_{N-1}, -p_0, ..., -p_{N-1})$$

The symmetric property of this table, called "negacyclicity", comes from the modulo $X^N + 1$ used to define our plaintext space. Given a non binary plaintext space, this allows the computation of non polynomial negacyclic functions with no overhead compared to the binary bootstrapping. This technique is then called functional bootstrapping. Note that proper redundancy must be introduced in the coefficients of the polynomial to ensure a low probability of error during this procedure. In general, given a plaintext space of 2^r elements, the result is an encryption of the right element only if $|\rho - 2N \cdot m| < \frac{N}{2^r}$.

2.6 Noise Analysis

In this section, we gather all the noise results mentioned in previous sections to make them easily accessible. We assume here that the noise variance of each input TLWE sample (respectively TRLWE sample and TRGSW sample) is bounded by V_{TLWE} (respectively V_{TRLWE}). We note V the noise variance of the output of each operation. Finally, we note V_{BS} and V_{KS} the noise variances of the bootstrapping key and keyswitch key.

Additions:

- Between a TLWE and a cleartext: $V \leq V_{\text{TLWE}}$.
- Between two TLWE: $V \leq 2 \cdot V_{\text{TLWE}}$.
- Between a TRLWE and a cleartext: $V \leq V_{\text{TRLWE}}$.
- Between two TRLWE: $V \leq 2 \cdot V_{\text{TRLWE}}$.
- Between a TRGSW and a cleartext: $V \leq V_{\text{TRLWE}}$.

- Between two TRGSW: $V \leq 2 \cdot V_{\text{TRLWE}}$.

Multiplications:

- Between a TLWE and a cleartext $m \in \mathbb{Z}$: $V \leq m^2 \cdot V_{\text{TLWE}}$.
- Between a TRLWE and a cleartext $m \in \mathbb{Z}[X]$: $V \leq ||m||_2^2 \cdot V_{\text{TRLWE}}$.
- Between a TRGSW and a cleartext $m \in \mathbb{Z}[X]$: $V \leq ||m||_2^2 \cdot V_{\text{TRLWE}}$.
- Between a TRGSW(m) and a TLWE:

$$V \leq (k+1) \cdot l \cdot N \cdot \left(\frac{B_g}{2}\right)^2 \cdot V_{\text{TRLWE}} + ||m||_2^2 \cdot \left(\frac{1 + k \cdot N}{12 B_g^{2l}}\right) + ||m||_2^2 \cdot V_{\text{TRLWE}}$$

Public Functional Keyswitch:

$$V \leq R^2 \cdot V_{\text{TLWE}} + \frac{n}{12 B_{KS}^{2t}} + n \cdot N \cdot t \cdot \left(\frac{B_{KS}}{2}\right)^2 \cdot \text{Var}_{\text{KS}}$$

Private Functional Keyswitch:

$$V \leq R^2 \cdot V_{\text{TLWE}} + R^2 \cdot \frac{n+1}{12 B_{KS}^{2t}} + p \cdot (n+1) \cdot t \cdot \left(\frac{B_{KS}}{2}\right)^2 \cdot \text{Var}_{\text{KS}}$$

Extraction:

$$V \leq V_{\text{TRLWE}}$$

CMux gate:

$$V \leq (k+1) \cdot l \cdot N \cdot \left(\frac{B_g}{2}\right)^2 \cdot V_{\text{TRLWE}} + \left(\frac{1 + k \cdot N}{12 B_g^{2l}}\right) + V_{\text{TRLWE}}$$

BlindRotate:

$$V \leq n \cdot \left((k+1) \cdot l \cdot N \cdot \left(\frac{B_g}{2}\right)^2 \cdot V_{\text{BS}} + \left(\frac{1 + k \cdot N}{12 B_g^{2l}}\right)\right) + V_{\text{TRLWE}}$$

Bootstrapping:

$$V \leq n \left((k+1) \cdot l \cdot N \left(\frac{B_g}{2}\right)^2 V_{\text{BS}} + \left(\frac{1 + kN}{12 B_g^{2l}}\right) + \frac{1}{12 B_{KS}^{2t}} + N \cdot t \left(\frac{B_{KS}}{2}\right)^2 V_{\text{KS}}\right)$$

2.7 TFHE in This Thesis

Most FHE cryptosystems can be utilized in either of two modes:

- **Binary mode:** functions are evaluated as logic circuits.
- **Larger plaintext mode:** functions are approximated by low degree polynomials over a large plaintext space. The functional bootstrapping of TFHE can also compute negacyclic functions.

In this thesis, we improve TFHE on both points as follows:

- We extend the computation of logic circuits to inputs decomposed in any basis B rather than 2.
- We build a functional bootstrapping that is not restricted to negacyclic functions.

Besides, the analysis of the noise formulas of each building blocks of TFHE allows us to efficiently estimate the noise growth and error rate of our techniques. This also gives us different metrics to compare our work to the state of the art, namely the latency, the noise growth, and the error rate.

Apart from extending the toolkit of TFHE, we apply our knowledge of TFHE to interesting tasks, such as transciphering and neural network inferences.

Chapter 3

Neural Networks

3.1 Neural Networks and FHE

An artificial neural network, or neural network for short, is a type of machine learning model which can "learn" from a dataset to infer some specific information over new data. In 1957, Rosenblatt introduced the first neural network called "perceptron" [82]. His aim was to build a system with the ability to learn and generalize from limited information, similarly to what he calls "higher organisms". The basic structure that he came up with gave birth to the concept of feedforward neural networks. Since then, feedforward neural networks have been used in diverse fields such as pattern classification, image processing, control systems and many more [84].

Deep neural networks gained a tremendous boost in popularity in 2015 when AlphaGo [89] achieved victory in the board game Go against a professional player. Only 2 years later, it successfully beat the world champion of Go, fostering the dream of conceiving an artificial intelligence (AI) smarter than humans. In 2022, deep neural networks became a hot societal topic again with the development of ChatGPT [76], a deep neural network-based conversational AI. This chatbot can hold human-like conversation on any subject but raised many concerns, notably privacy issues coming with its use [106].

But concretely, what is a neural network? We describe a basic feedforward neural network structure in Figure 3.1. Let us introduce the required terminology:

- **Neuron:** A neuron is the smallest unit in the neural network. They are represented as circle in Figure 3.1.
- **Layer:** A layer is a group of neurons all working on the same inputs. The

result of their computation is sent as the input of the next layer of neurons. They are represented as rectangles in Figure 3.1. The intermediary layers are often called hidden layer.

- **Weights:** The linear combinations computed in each layer depends on the input of the layer and the weights of the layer.
- **Activation:** The activation functions are nonlinear function computed by each neuron after the linear operation and noted as f_i on Figure 3.1. Note that if an activation function was linear, multiple layer could be merged leading to a smaller network computing the same function. The most usual activation functions are the Rectified Linear Unit $\mathrm{ReLU}(x) = \max(0, x)$, the Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$ and the hyperbolic tangent $\tanh(x) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$.
- **Hyperparameter:** The number of neurons in each layer, the number of layers, the choice of activation functions, and other parameters related to the structure of the neural network are called hyperparameters.

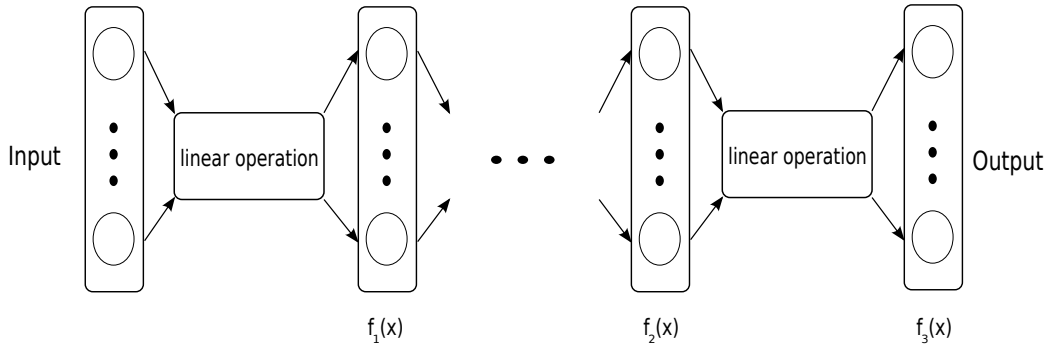


Figure 3.1 – Basic structure of a feedforward neural network.

A neural network usually lives through two phases:

- **Training:** During the training phase, the neural network uses large amount of data to find the most appropriate weights to fit a targeted application. Fine tuning of the hyperparameters can also be performed during the training process.
- **Inference:** The inference phase is the application of the neural network to new inputs.

The most usual training methods are variants of the gradient descent [75]. To that end, a loss function is defined and empirically evaluated to determine how close

the outputs of the neural network are for a given set of weights. Then, the weights of the neural network are updated by following the gradient of the loss function to improve the loss. A technique called "back propagation" allows for an efficient computation of the gradient by leveraging the structure of neural networks. This operation is iterated until convergence of the result to a local optimal set of weights. Many more efficient variants of this training technique exist, such as the stochastic gradient descent or the adaptive gradient descent [95].

Trained neural networks are often highly valuable intellectual properties. As such, service providers usually have to make computations on their end on the client's requests in order to avoid disclosing the network.

For the purpose of this thesis, we assume that the server doing the computation is honest but curious. Concretely, this means that the server can be trusted to compute the expected function while trying to extract information from clients for its own purposes. In this context, we only aim to ensure the confidentiality of the client's input data against attack from the service provider. Hence, we do not investigate ways to ensure that the server computes the intended function, such as verifiable computing [102]. In Figure 3.2, we describe a basic interaction between a client and the cloud. The client's input x represents personal data, which could be medical data, geographic position, or any other sensitive data. The cloud only has access to x and $f(x)$ between brackets, which denotes encrypted information. As such, it does not learn anything about either x or $f(x)$ as long as the cryptosystem used is secure[1] and malleable enough to compute the encryption of $f(x)$ from the encryption of x. Meanwhile, the client can decrypt the result $f(x)$ using his private key. The mentioned security and malleability requirements are met by FHE cryptosystems, which are perfect candidates for this type of application. It is important to note that thanks to FHE the client does not need to interact with the server at all during the computation of the function f.

FHE computations can theoretically be used either during the training phase or the inference phase of the neural network. However, the training phase is much more difficult to compute in the encrypted domain than the inference phase. Indeed, difficulties such as discretization leading to numerical error propagation come into play during the training process. Since the loss function and the precision of the network cannot be monitored while encrypted, the impact of numerical errors cannot be assessed without decryption. This also makes the fine tuning of hyperparameters difficult without interactions with the client. Some works ([74],[65]) try to tackle the problem of homomorphic training. However, they fail to address some of the subtleties of a complete training process, in particular in regard to numerical stability, and still require a prohibitively long training process: an ex-

[1] In the "honest but curious" model, CPA security is enough to be considered secure.

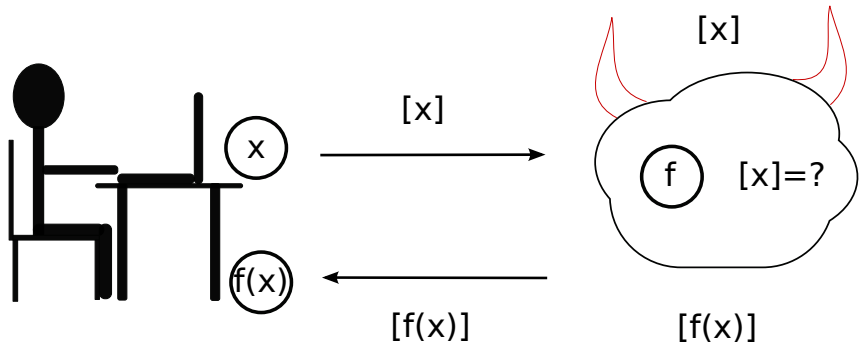


Figure 3.2 – Exchange between client and honest but curious cloud. Brackets denote encryption.

pected time of 13 years for the full training of a fairly small neural network. A more realistic approach exists in a different setting when allowing for interaction between the server and clients thanks to federated learning [92, 86]. However, in the federated learning setting, most of the actual training of the network is performed in the clear domain while FHE is used for a crucial but restricted part of the protocol. Since this thesis focus on FHE computation, it makes sense to focus first on the inference phase.

The inference of neural networks can be roughly categorized in 2 types:

- **Feedforward Neural Networks:** Essentially following the structure of Figure 3.1, feedforward neural networks are a succession of layer fixed during the training phase. Although the most efficient homomorphic evaluations of this type of neural networks rely on the LHE mode of cryptosystems such as BGV and CKKS, difficulties arise when computing the activation functions. The neural network can be purposely designed to be FHE friendly by using low degree polynomial activation functions or the usual activation functions can be approximated by polynomials. Besides, such neural networks are usually very deep in practice (VGG16 has 16 layers [90]), but the largest standard LHE parameters allow for a multiplicative depth of up to 30 [61] (which would not even allow a multiplication depth of 2 per layer for the evaluation of VGG16). In order to scale to deep and useful neural networks, it is required to switch to FHE mode.

- **Recurrent Neural Networks:** As opposed to feedforward neural networks, recurrent neural networks work on a list of inputs of arbitrary size, such as sentences. The neural network takes as input one element of the list as well as

its own output and is used recursively over each element of the list of inputs as shown on Figure 3.3. As such, the multiplicative depth of computation is unbounded due to the recurrent nature of the network, and FHE needs to be used over LHE. Furthermore, errors from polynomial approximations propagate through each layer of the neural network. It can be considered as a more prospective type of neural networks from the FHE perspective.

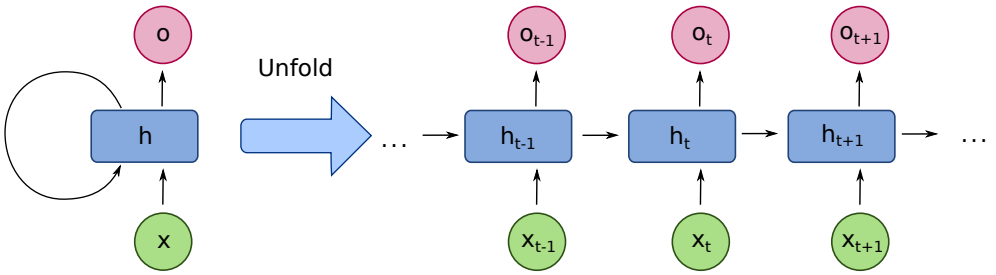


Figure 3.3 – Recurrent Neural Network Structure.
Image from https://commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg?uselang=fr

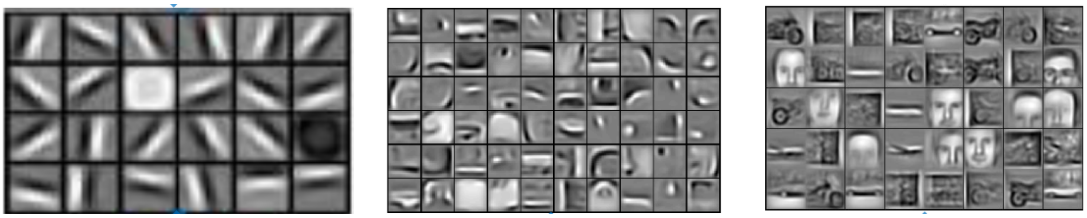
The ease of implementation of neural network in the encrypted domain is deeply correlated to the resilience of the networks to discretization and approximation. On the one hand, with BFV, BGV, and CKKS, each activation function of a given network need to be approximated as low degree polynomials, which may impact the accuracy of the results. On the other hand, TFHE is usually used with small plaintext space, which requires a heavy discretization of the network. Thus, research on discretized neural networks, whether with discretization a posteriori [52] or discretization by design [20], may give fruitful resource to build FHE friendly neural networks.

Our ultimate goal is to be able to evaluate any neural network over encrypted inputs. As such, in this work we focus on improving FHE operators for evaluating neural networks building blocks rather than designing FHE friendly networks. Indeed, some FHE friendly networks can be evaluated in less than a second, but are only efficient for extremely simple tasks. For instance, in [13], Brutzkus et al., build an FHE evaluation of a feedforward neural network which can be evaluated in half a second with 96.9% precision over the MNIST [60] dataset. The MNIST dataset is essentially the easiest dataset to fit with a neural network and 96.9% accuracy is not considered that high for this specific task. Besides, timings scale up to 12 minutes and 10 seconds to reach 74.1% accuracy over the CIFAR-10 [58] dataset which is still a fairly simple dataset to fit.

3.2 Transfer Learning to the Rescue of FHE

Nowadays, a wide variety of neural networks consist of pre-trained models. Indeed the neural network ecosystem is structured around open source publicly available libraries of training algorithms as well as building blocks, pre-trained models and full networks for solving numerous tasks. When a new model needs to be built, on top of using publicly available training algorithms, one generally uses relevant pre-trained models or subsets of existing models as starting points for easier network design and faster convergence of training. Some approaches, referred to as *transfer learning* approaches, do so by inducing no or very little modifications of the building blocks used in a pre-existing publicly known network.

Transfer learning is particularly suited for building networks for new image analysis tasks from networks already built for other such tasks [90]. Indeed, a network able to perform an image analysis task is generally structured in two parts: the first part takes raw pixels as inputs and, layer after layer, turns them into abstract "features". Essentially, at some point in the network, a high dimensional vector is built which components quantify the presence of a given concept in the input image. This idea is illustrated in Figure 3.4 over a dummy neural network where we see what kind of image activate each neuron in each layer. In this occurrence, neurons from the first layer are activated by edges from the input images while the third layer are activated by more complex concept such as vehicles. The remaining of the neural network layers then turn this features into a prediction, in accordance to actual task of the network.



First Layer: Edges | Second Layer: Parts | Third Layer: Objects

Figure 3.4 – Visualization of features recognized at each layer.
Image modified from: https://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/CVPR2012-Tutorial_lee.pdf

When building a network for a new image analysis task, one can reuse the first layers of the network almost as is and replace the few last layers by newly trained ones, specific to the new task. A most relevant example is the VGG16 model [90] which has been trained on the ImageNet database and which is able to turn any image into feature vectors of dimension ≈ 25000. Plugging a small network of a few hundred neurons after VGG16 is then enough to perform many image analysis tasks with high reliability.

Let us now consider the following scenario. A radiologist has just acquired images from the body of one of its patient and needs to interact with a remote proprietary diagnostic service to get some insight. Said service provider carefully crafted a neural network for this purpose using a lot of precious hard-earned training data and is considered critical intellectual property. It is thus unacceptable for the service provider to disclose the details of their neural network. Besides, as health-related data, the patient images and data are considered sensitive and cannot be shared without protecting their confidentiality.

One way to resolve these conflicting requirements is by bringing privacy preserving FHE calculations into the picture. The use of transfer learning can make this computation much more FHE friendly. Indeed, the network can then be split in two parts as shown in Figure 3.5:

- A preprocessing network (such as VGG16) which is publicly available and has no dependencies on the precious hard-earned training data of the service provider. This part can thus be disclosed to the radiologist's information system and run in the clear domain before encryption.
- A much smaller decisional network trained on the service provider sensitive data which turns the outputs of the preprocessing network into the highly reliable insight expected by the radiologist. This part must be kept hidden as critical intellectual property.

So rather than sending FHE-encrypted images, the radiologist only sends an FHE-encryption of the preprocessed image, which is a vector of much smaller size than high-resolution images. On the service provider side, only the smaller decisional network has to be run in the encrypted domain therefore dramatically decreasing the footprint of FHE-calculations and resulting in much better scaling properties. Since transfer learning techniques are widely applicable and applied in the neural network community we can therefore claim that performing advanced machine learning tasks over encrypted data does not require scaling encrypted-domain calculation to large networks, since, as argued above, the fact of running the preprocessing on the user side does not impact the confidentiality properties of the setup.

Since its introduction in early 2010s [36, 87], transfer learning has become a commonly used technique in machine learning, as shown in [90, 103]. In particular, many examples have been published in medical context, as described in [73, 78, 56, 3]. It offers numerous advantages compared to training a new network from scratch. First it significantly reduces the amount of computational resources needed to converge towards a satisfying statistical model during the training phase. Second it can drastically reduce the amount of data needed during the training phase. Last

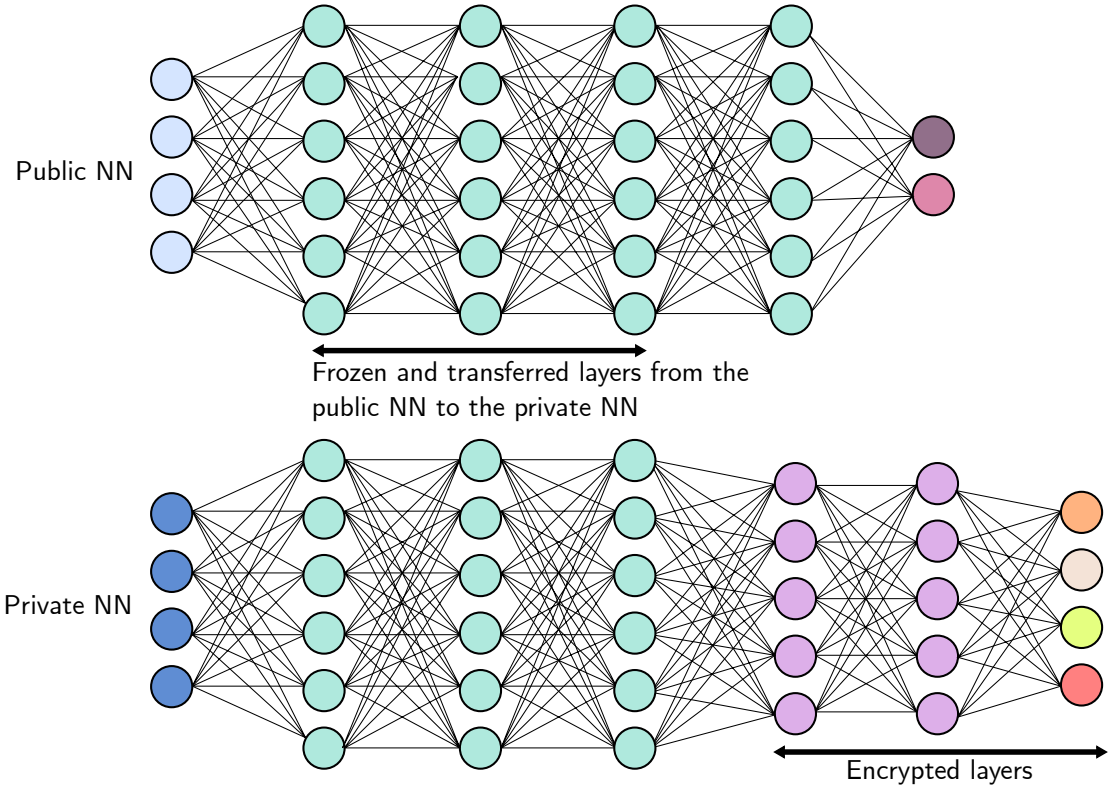


Figure 3.5 – Transfer learning set up for FHE

it can even significantly improve overall performances in the target domain. To achieve these goals, many approaches have been proposed in the literature [105] to design specific transformations able to transfer knowledge from the source domain to the target domain. In the specific context of deep learning, a wide variety of methods have been designed [94] to overcome training limitations of large scale deep learning approaches. For the purpose of FHE, we consider transfer learning in its simplest form. It consists in using deep features from a pre-trained as a pre-processing step for a target domain-specific classifier, as inspired by [88], without fine-tuning the preprocessing network. This approach assumes relative proximity between the source domain and the target domain, as noted in [101]. However, in the context of homomorphic evaluation, it allows us to consider the preprocessing step as public knowledge which can thus be performed in clear by the client, restricting the amount of homomorphic computation to be evaluated by the server. The main advantages we derive from this technique are a lower multiplicative depth for the homomorphic part of the network and a lowered latency for the overall evaluation of the network. In addition, the output of the preprocessing part is usually smaller than its input, which further ease the following homomorphic

computation.

3.3 Homomorphic Neural Network Evaluation

The rise of machine learning as a service (MLaaS) has grown concerns regarding the privacy of the users' data. This led a part of the research community to look into privacy enhancing techniques that could fit well with these services. As such, multiple works study the use of FHE as a way to enable a private evaluation of an outsourced neural network. The first results come from CryptoNets [45] in 2016 which succeeded in computing a small neural network with 2 hidden layers in 4 minutes and 10 seconds, achieving an accuracy of 98.95% over the MNIST dataset. However, this first milestone is far from enough: the activation function are restricted to square functions to be FHE friendly and the latency of 4 minutes is huge considering how small the network is. Besides, an LHE cryptosystem is used. As such, the parameters of the cryptosystem, and consequently the running time of each operation, grow with the depth of the network, thwarting any attempt to infer the running time of a deeper neural network from this result.

In 2017, a new approach relying on heavy discretization is proposed in [9]. They apply their strategy over the MNIST dataset and reach an accuracy of 93.7% for a latency of $0.5s$ or an accuracy of 96.3% for a latency of $1.6s$. To that end, they discretize the inputs to the binary level while keeping the weights in $[\![-10, 10]\!]$, while using the sign function for the activations thanks to TFHE's functional bootstrapping. This strategy shows the possibility to evaluate a neural network in the encrypted domain with a low latency, and more specifically, below the threshold of one second. Furthermore, this approach relies on a cryptosystem in FHE mode, making it easier to infer the running time of deeper neural networks. From this point, the question that remains is whether deep and accurate neural networks can be evaluated while using more complex activation functions and keeping a reasonable latency.

The problem of evaluating more usual activation function is explored in [32] using polynomial approximations. In this work, each activation is approximated by a degree two polynomial. The idea of quantization and pruning of the neural network are also addressed to make homomorphic computation efficient. They achieve an accuracy of 98.7% over the MNIST dataset with a latency of 39s on a network of similar structure to the first CryptoNets. Furthermore, they extend their experiments to a more complex dataset: CIFAR-10. To fit this dataset properly, they require a much deeper network, with a latency close to 30 minutes and only achieving 76% accuracy. This work highlights well the gap left to bridge in order to evaluate practical neural networks in the span of a second.

So far, no work made use of batching techniques to reduce the latency when possible. Indeed, both CryptoNet [45] and its improvement [32] use batching to achieve an amortized running time of $0.06s$. However, it is doubtful that a user would request for the inference of a neural network over thousands of inputs at once. As such, it makes sense to use batching to achieve lower latency rather than lower amortized computation time. In [13], efforts are put in batching inputs in the most efficient manner to reach the best latency possible. They compare themselves to both CryptoNet [32] and DiNN [9]. First, they use the CryptoNet structure and reach a latency of $2.2s$ to reach 98.95% accuracy on the MNIST dataset. Then, they build a smaller network reaching an accuracy of 96.92% for a latency of $0.29s$. With this, they top all other approaches of the state of the art on the MNIST dataset. However, they acknowledge that this result is not enough to evaluate efficiently a deep neural network. As such, they suggest the use of "deep representation" to scale their work to deep neural network. This deep representation is, in fact, the transfer learning approach discussed in Section 3.2.

In [66], more work is done to study binary discretization for homomorphic computation, reaching an accuracy of 99.77%, the highest accuracy so far on MNIST with an FHE implementation with a latency of 2 minutes, and an accuracy of 99.54% with a latency of 9 seconds. They also achieve a better scaling to CIFAR-10 than other methods by achieving an accuracy of 94.62% with a latency of 3 hours and 40 minutes or an accuracy of 92.54% with a latency of 38 minutes.

So far, all these works study the implementation of simple feedforward neural network. However, different types of neural networks exist. In [53], Izabachène et al., show that one update of a Hopfield neural network can be computed in less than a second in the homomorphic domain. However, the literature is limited regarding the evaluation of other types of recurrent neural networks.

Besides, in [59], Lam et al., show that using transfer learning, we can achieve state of the art accuracy (100% accuracy) on practical tasks such as face recognition with a latency lower than a second. However, a degradation of the accuracy occur for tasks such as voice recognition when aiming for low latency: from 97% in the clear down to 91% accuracy even with a latency of 10 seconds. This work confirms that the conjunction of transfer learning and FHE is an excellent candidate to make a privacy preserving outsourced neural network. This fact is further validated in [107] where transfer learning is used along FHE to implement an efficient homomorphic speaker recognition system.

The main roadblocks are now the computation of precise activation functions, the latency of the evaluation for deep neural network, and the evaluation of recurrent neural networks in the homomorphic domain. In this context, one of our aims is to

improve the toolbox of FHE operators so that activation functions become easier to compute in the homomorphic domain.

Part II

Contributions

Chapter 4

ComBo: a novel functional bootstrapping method for efficient evaluation of nonlinear functions in the encrypted domain

This chapter is a reproduction of our paper ComBo [33] accepted in AfricaCrypt 2023 and co-written with Aymen Boudguiga and Renaud Sirdey.

Abstract. The application of Fully Homomorphic Encryption (FHE) to privacy issues arising in inference or training of neural networks has been actively researched over the last few years. Yet, although practical performances have been demonstrated on certain classes of neural networks, the inherent high computational cost of FHE operators has prevented the scaling capabilities of FHE-based encrypted domain inference to the large and deep networks used to deliver advanced classification functions such as image interpretation tasks. To achieve this goal, a new hope is coming from TFHE functional bootstrapping which, rather than being just used for refreshing ciphertexts (i.e., reducing their noise level), can be used to evaluate operators which are difficult to express as low complexity arithmetic circuits, at no additional cost. In this work, we first propose ComBo (Composition of Bootstrappings) a new full domain functional bootstrapping method with TFHE for evaluating any function of domain and codomain the real torus $\mathbb{T}$ by using a small number of bootstrappings. This result improves on previous approaches: like them, we allow for evaluating any functions, but with error rates reduced by a factor of up to 2^{80}. This claim is supported by a theoretical analysis of the error rate of other functional bootstrapping methods

from the literature. The paper is concluded by extensive experimental results demonstrating that our method achieves better performances in terms of both time and precision, in particular for the Rectified Linear Unit (ReLU) function, a nonlinear activation function commonly used in neural networks. As such, this work provides a fundamental building-block towards scaling the homomorphic evaluation of neural networks over encrypted data.

Keywords: FHE; TFHE; functional bootstrapping; ReLU; ComBo

4.1 Introduction

Machine learning application to the analysis of private data, such as health or genomic data, has encouraged the use of homomorphic encryption for private inference or prediction with classification or regression algorithms where the ML models and/or their inputs are encrypted homomorphically [99, 18, 17, 9, 107, 53, 108]. Even training machine learning models with privacy guarantees on the training data has been investigated in the centralized [54, 24, 74, 65] and collaborative [86, 69] settings. In practice, machine learning algorithms and especially neural networks require the computation of non-linear activation functions such as the sign, ReLU or sigmoid functions. Still, computing non-linear functions homomorphically remains challenging. For levelled homomorphic schemes such as BFV [11, 39] or CKKS [23], non-linear functions have to be approximated by polynomials. However, the precision of these approximations differs with respect to the considered plaintext space (i.e., input range), approximation polynomial degree and its coefficients size, and has a direct impact on the multiplicative depth and parameters of the cryptosystem. The more precise is the approximation, the larger are the cryptosystem parameters and the slower is the computation. On the other hand, homomorphic encryption schemes having an efficient bootstrapping, such as TFHE [25, 28] or FHEW [37], can be tweaked to encode functions via look-up table (LUT) evaluations within their bootstrapping procedure. Hence, rather than being just used for refreshing ciphertexts (i.e., reducing their noise level), the bootstrapping becomes *functional* [10] or *programmable* [30] by allowing the evaluation of arbitrary functions as a bonus. These capabilities result in promising new approaches for improving the overall performances of homomorphic calculations, making the FHE "API" better suited to the evaluation of mathematical operators which are difficult to express as low complexity arithmetic circuits. It is also important to note that FHE cryptosystems can be hybridized, for example BFV ciphertexts can be efficiently (and homomorphically) turned into TFHE ones [7, 107]. As such, the building blocks discussed in this paper are of relevance also in the setting where the desired encrypted-domain calculation can be split into a preprocessing step more efficiently done using BFV (e.g. several inner product or

distance computations) followed by a nonlinear postprocessing step (such as an activation function or an argmin) which can then be more conveniently performed by exploiting TFHE functional bootstrapping. In this work, we thus systematize and further investigate the capabilities of TFHE functional bootstrapping.

Contributions – The main contribution of this paper is a novel functional bootstrapping algorithm. It is a *full domain* functional bootstrapping algorithm in the sense that it does not require to add a bit of padding to the encoding of the messages (as described clearly in [30]). There are several other such methods in the literature. We show that ours is the best option to date for single-digit operations on the full torus (where a message is encoded into a single ciphertext).
There are several other contributions in this paper. We present them succinctly here:

- Our *novel functional bootstrapping algorithm* (ComBo) is built by composing several bootstrapping operations. It is based on the idea to separate any function in a even and odd part and then compute both in parallel. We present several versions to increase its efficiency and show that our method is the most accurate among state-of-the art full domain bootstrapping algorithms.

- We *implement and test* our algorithms by evaluating several functions homomorphically. Among them, the Rectified Linear Unit (ReLU) function is of particular interest for private neural network applications. This allows us to compare the computational overhead of our algorithm with other existing methods.

- In order to compare the error rate of the different existing methods (which this work aims to reduce), *we develop an error analysis methodology* and describe it in detail. This shows that *our algorithm improves on previous approaches*, most of the time by a significant margin. This methodology, we argue, is the most appropriate way to compare similar algorithms and can be reused for further research on the subject to improve comparability.

- As a bonus, in order to compare our algorithm fairly to other previous solutions from the community, *we introduce consistent notations for describing all existing solutions and their error probabilities in a unified way.* We also fully implemented and tested all of them. We consider that this strengthens the present paper and can be considered, in and of itself, a worthy contribution to the development of the field.

Related works – In 2016, the TFHE paper made a breakthrough by proposing an efficient bootstrapping for homomorphic gate computation. Then, Bourse

et al., [9] and Izabachene et al., [53] used the same bootstrapping algorithm for extracting the (encrypted) sign of an encrypted input. Boura et al., [8] showed later that TFHE bootstrapping could be extended to support a wider class of functionalities. Indeed, TFHE bootstrapping naturally allows to encode function evaluation via their representation as look-up tables (LUTs). Recently, different approaches have been investigated for functional bootstrapping improvement. In particular, Kluczniak and Schild [57], Liu et al., [64] and Yang et al., [100] proposed methods that take into consideration the negacyclicity of the cyclotomic polynomial used within the bootstrapping, for encoding look-up tables over the full real torus $\mathbb{T}$. Meanwhile, Guimarães et al., [48] extended the ideas in Bourse et al., [10] to support the evaluation of certain activation functions such as the sigmoid. One last method (WoP-PBS), presented in Chillotti et al., [31] achieves a functional bootstrapping over the full torus using a BFV type multiplication, which was designed for and only applicable to parameter sets much larger than standard TFHE parameters. Besides, since the probabilistic behavior of decryption also appears during the bootstrapping procedure, the error rate analysis of homomorphic computation are becoming of interest when using TFHE as shown in [48] and [5].

Paper organization – The remainder of this paper is organized as follows. Section 4.2 reviews TFHE building blocks. Section 4.3 describes the functional bootstrapping idea coming from the TFHE gate bootstrapping. Sections 4.4 presents our new functional bootstrapping method ComBo in full detail. It also describes, under a unified formalism, the other available methods for single digit functional bootstrapping. Finally, Section 4.6 provides experimental results for ComBo and compares it to the other methods which we also implemented. These results are provided for both generic LUT evaluations over encrypted data as well as the ReLU neural network activation function.

4.2 TFHE

4.2.1 Notations

In the upcoming sections, we denote vectors by bold letters and so, each vector $\boldsymbol{x}$ of n elements is described as: $\boldsymbol{x} = (x_1, \ldots, x_n)$. $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ is the inner product of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. We denote matrices by capital letters, and the set of matrices with m rows and n columns with entries sampled in $\mathbb{K}$ by $\mathcal{M}_{m,n}(\mathbb{K})$.

We refer to the real torus $\mathbb{R}/\mathbb{Z}$ as $\mathbb{T}$. $\mathbb{T}_N[X]$ denotes the $\mathbb{Z}$-module $\mathbb{R}[X]/(X^N + 1) \mod [1]$ of torus polynomials, where N is a power of 2. $\mathcal{R}$ is the ring $\mathbb{Z}[X]/(X^N + 1)$ and its subring of polynomials with binary coefficients is $\mathbb{B}_N[X] = \mathbb{B}[X]/(X^N +$

1) ($\mathbb{B} = \{0, 1\}$). Finally, we denote respectively by $[x]_{\mathbb{T}}$, $[x]_{\mathbb{T}_N[X]}$ and $[x]_{\mathcal{R}}$ the encryption of x over $\mathbb{T}$, $\mathbb{T}_N[X]$ or $\mathcal{R}$.

$x \xleftarrow{\$} \mathbb{K}$ denotes sampling x uniformly from $\mathbb{K}$, while $x \xleftarrow{\mathcal{N}(\mu,\sigma^2)} \mathbb{K}$ refers to sampling x from $\mathbb{K}$ following a Gaussian distribution of mean μ and variance σ^2. Given $x \xleftarrow{\mathcal{N}(\mu,\sigma^2)} \mathbb{R}$, the probability $P(a \leq x \leq b)$ is equal to $\frac{1}{2}(erf(\frac{b-\mu}{\sqrt{2}\sigma}) - erf(\frac{a-\mu}{\sqrt{2}\sigma}))$, where erf is Gauss error function; $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}$. If $\mu = 0$, we will denote $P(-a \leq x \leq a) = erf(\frac{a}{\sqrt{2}\sigma})$ by $\mathcal{P}(a, \sigma^2)$. The same result and notation apply for $x \xleftarrow{\mathcal{N}(0,\sigma^2)} \mathbb{T}$ as long as the distribution is concentrated as described in [28].

Given a function $f : \mathbb{T} \to \mathbb{T}$ and an integer k, we define $\mathrm{LUT}_k(f)$ to be the Look-Up Table defined by the set of k pairs $\left(i, f\left(\frac{i}{k}\right)\right)$ for $i \in [\![0, k-1]\!]$. We will write $\mathrm{LUT}(f)$ when the value of k is tacit.

Given a function $f : \mathbb{T} \to \mathbb{T}$ and an integer $k \leq N$, we define a polynomial $P_{f,k} \in \mathbb{T}_N[X]$ of degree N as: $P_{f,k} = \sum_{i=0}^{N-1} f\left(\frac{\lfloor \frac{k \cdot i}{2N} \rfloor}{k}\right) \cdot X^i$. If k is a divisor of $2N$, $P_{f,k}$ can be written as $P_{f,k} = \sum_{i=0}^{\frac{k}{2}-1} \sum_{j=0}^{\frac{2N}{k}-1} f(\frac{i}{k}) \cdot X^{\frac{2N}{k} \cdot i + j}$. For simplicity sake, we will write P_f instead of $P_{f,k}$ when the value k is tacit.

4.2.2 TFHE Structures

The TFHE encryption scheme was proposed in 2016 [25]. It improves the FHEW cryptosystem [37] and introduces the TLWE problem as an adaptation of the LWE problem to $\mathbb{T}$. It was updated later in [26] and both works were recently unified in [28]. The TFHE scheme is implemented in the TFHE library [27]. TFHE relies on three structures to encrypt plaintexts defined over $\mathbb{T}$, $\mathbb{T}_N[X]$ or $\mathcal{R}$:

- **TLWE Sample:** $(\boldsymbol{a}, b)$ is a valid TLWE sample if $\boldsymbol{a} \xleftarrow{\$} \mathbb{T}^n$ and $b \in \mathbb{T}$ verifies $b = \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e$, where $\boldsymbol{s} \xleftarrow{\$} \mathbb{B}^n$ is the secret key, and $e \xleftarrow{\mathcal{N}(0,\sigma^2)} \mathbb{T}$. Then, $(\boldsymbol{a}, b)$ is a fresh TLWE encryption of 0.
- **TRLWE Sample:** a pair $(\boldsymbol{a}, b) \in \mathbb{T}_N[X]^k \times \mathbb{T}_N[X]$ is a valid TRLWE sample if $\boldsymbol{a} \xleftarrow{\$} \mathbb{T}_N[X]^k$, and $b = \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e$, where $\boldsymbol{s} \xleftarrow{\$} \mathbb{B}_N[X]^k$ is a TRLWE secret key and $e \xleftarrow{\mathcal{N}(0,\sigma^2)} \mathbb{T}_N[X]$ is a noise polynomial. In this case, $(\boldsymbol{a}, b)$ is a fresh TRLWE encryption of 0.

  The TRLWE decision problem consists of distinguishing TRLWE samples

from random samples in $\mathbb{T}_N[X]^k \times \mathbb{T}_N[X]$. Meanwhile, the TRLWE search problem consists in finding the private polynomial $\boldsymbol{s}$ given arbitrarily many TRLWE samples. When $N = 1$ and k is large, the TRLWE decision and search problems become the TLWE decision and search problems, respectively.

Let $\mathcal{M} \subset \mathbb{T}_N[X]$ (or $\mathcal{M} \subset \mathbb{T}$) be the discrete message space[1]. To encrypt a message $m \in \mathcal{M}$, we add $(\mathbf{0}, m) \in \{0\}^k \times \mathcal{M}$ to a TRLWE sample (or to a TLWE sample if $\mathcal{M} \subset \mathbb{T}$). In the following, we refer to an encryption of m with the secret key $\boldsymbol{s}$ as a T(R)LWE ciphertext noted $\boldsymbol{c} \in \text{T(R)LWE}_{\boldsymbol{s}}(m)$.

To decrypt a ciphertext $\boldsymbol{c} \in \text{T(R)LWE}_{\boldsymbol{s}}(m)$, we compute its *phase* $\phi(\boldsymbol{c}) = b - \langle \boldsymbol{a}, \boldsymbol{s} \rangle = m + e$. Then, we round it to the nearest element of $\mathcal{M}$. Therefore, if the error e was chosen to be small enough (yet high enough to ensure security), the decryption will be accurate.

- **TRGSW Sample:** a valid TRGSW sample is a vector of TRLWE samples. To encrypt a message $m \in \mathcal{R}$, we add $m \cdot H$ to a TRGSW sample, where H is a gadget matrix[2] using an integer B_g as a base for its decomposition. Chilotti et al., [28] defines an external product between a TRGSW sample A encrypting $m_a \in \mathcal{R}$ and a TRLWE sample $\boldsymbol{b}$ encrypting $m_b \in \mathbb{T}_N[X]$. This external product consists in multiplying A by the approximate decomposition of $\boldsymbol{b}$ with respect to H (Definition 3.12 in [28]). It yields an encryption of $m_a \cdot m_b$ i.e., a TRLWE sample $\boldsymbol{c} \in \text{TRLWE}_{\boldsymbol{s}}(m_a \cdot m_b)$. Otherwise, the external product allows also to compute a controlled MUX gate (CMUX) where the selector is $C_b \in \text{TRGSW}_{\boldsymbol{s}}(b), b \in \{0, 1\}$, and the inputs are $\boldsymbol{c}_0 \in \text{TRLWE}_{\boldsymbol{s}}(m_0)$ and $\boldsymbol{c}_1 \in \text{TRLWE}_{\boldsymbol{s}}(m_1)$.

4.2.3 TFHE Bootstrapping

TFHE bootstrapping relies mainly on three building blocks:

- **Blind Rotate:** rotates a plaintext polynomial encrypted as a TRLWE ciphertext by an encrypted position. It takes as inputs: a TRLWE ciphertext $\boldsymbol{c} \in \text{TRLWE}_{\boldsymbol{k}}(m)$, a vector $(a_1, \ldots, a_n, a_{n+1} = b)$ where $\forall i,\ a_i \in \mathbb{Z}_{2N}$, and n TRGSW ciphertexts encrypting $(s_1, \ldots, s_n)$ where $\forall i,\ s_i \in \mathbb{B}$. It returns a TRLWE ciphertext $\boldsymbol{c'} \in \text{TRLWE}_{\boldsymbol{k}}(X^{\langle \boldsymbol{a}, \boldsymbol{s} \rangle - b} \cdot m)$. In this paper, we will refer

[1] In practice, we discretize the Torus with respect to our plaintext modulus. For example, the usual encryption of a message $m \in \mathbb{Z}_4 = \{0, 1, 2, 3\}$ would be one of the following value $\{0, 0.25, 0.5, 0.75\}$.

[2] Refer to Definition 3.6 and Lemma 3.7 in TFHE paper [28] for more information about the gadget matrix H.

to this algorithm as BlindRotate. With respect to independence heuristic[3] stated in [28], the variance $\mathcal{V}_{BR}$ of the resulting noise after a BlindRotate satisfies the formula:

$$\mathcal{V}_{BR} < V_c + n\left((k+1)\ell N\left(\frac{B_g}{2}\right)^2 \vartheta_{BK} + \frac{(1+kN)}{12 \cdot B_g^{2l}}\right)$$

where V_c is the variance of the noise of the input ciphertext c, and ϑ_{BK} is the variance of the error of the bootstrapping key. In the following, we define:

$$\mathcal{E}_{BR} = n\left((k+1)\ell N\left(\frac{B_g}{2}\right)^2 \vartheta_{BK} + \frac{(1+kN)}{12 \cdot B_g^{2l}}\right)$$

- **TLWE Sample Extract:** takes as inputs both a position $p \in [\![0, N[\![$ and a ciphertext $\boldsymbol{c} \in \mathrm{TRLWE}_{\boldsymbol{k}}(m)$, and returns a TLWE ciphertext $\boldsymbol{c'} \in \mathrm{TLWE}_{\boldsymbol{k}}(m_p)$ where m_p is the p^{th} coefficient of the polynomial m. In this paper, we will refer to this algorithm as SampleExtract. This algorithm does not add any noise to the ciphertext.
- **Public Functional Keyswitching:** transforms a set of p ciphertexts $\boldsymbol{c}_i \in \mathrm{TLWE}_{\boldsymbol{k}}(m_i)$ into the resulting ciphertext $\boldsymbol{c'} \in \mathrm{T(R)LWE}_{\boldsymbol{s}}(f(m_1, \ldots, m_p))$, where $f()$ is a public linear morphism from $\mathbb{T}^p$ to $\mathbb{T}_{\overline{N}}[X]$. This algorithm uses 2 specific parameters, namely B_{KS} which is used as a base to decompose some coefficients, and t which gives the precision of the decomposition. Note that functional keyswitching serves at changing encryption keys and parameters. In this paper, we will refer to this algorithm as KeySwitch. As stated in [28, 48], the variance $\mathcal{V}_{KS}$ of the resulting noise after KeySwitch follows the formula[4]:

$$\mathcal{V}_{KS} < R^2 \cdot V_c + n\left(t\overline{N}\vartheta_{KS} + \frac{B_{KS}^{-2t}}{12}\right)$$

where V_c is the variance of the noise of the input ciphertext c, R is the Lipschitz constant of f and ϑ_{KS} the variance of the error of the keyswitching key. Note that n is a parameter of the input ciphertext, while $\overline{N}$ is a parameter of the output ciphertext. Thus, $\overline{N} = 1$ if the output is a TLWE ciphertext. In this paper and in most cases, $R = 1$. In the following, we define:

$$\mathcal{E}_{KS}^{n,\overline{N}} = n\left(t\overline{N}\vartheta_{KS} + \frac{B_{KS}^{-2t}}{12}\right)$$

[3]The independence heuristic ensures that all the coefficients of the errors of TLWE, TRLWE or TRGSW samples are independent and concentrated. More precisely, they are σ-subgaussian where σ is their standard deviation.

[4]Note that there is a discrepancy in the original TFHE papers [25, 26, 28] between the theorem and the proof.

TFHE comes with two bootstrapping algorithms. The first one is the gate bootstrapping. It aims at reducing the noise level of a TLWE sample that encrypts the result of a boolean gate evaluation on two ciphertexts, each of them encrypting a binary input. The binary nature of inputs/outputs of this algorithm is not due to inherent limitations of the TFHE scheme but rather to the fact that the authors of the paper were building a bitwise set of operators for which this bootstrapping operation was perfectly fitted.

TFHE gate bootstrapping steps are summarized in Algorithm 5. Note that $\{0,1\}$ is encoded as $\{0,\frac{1}{2}\}$. Step 1 consists in selecting a value $\mu \in \mathbb{T}$ which will serve later for setting the coefficients of the test polynomial $testv$ (in step 3). Step 2 rescales the components of the input ciphertext $\boldsymbol{c}$ as elements of $\mathbb{Z}_{2N}$. Step 3 defines the test polynomial $testv$. Note that for all $p \in [\![0, 2N[\![$, the constant term of $testv \cdot X^p$ is μ if $p \in]\!]\frac{N}{2}, \frac{3N}{2}[\![$ and $-\mu$ otherwise. Step 4 returns an accumulator $ACC \in \mathrm{TRLWE}_{\boldsymbol{s'}}(testv \cdot X^{\langle \bar{\boldsymbol{a}}, \boldsymbol{s} \rangle - \bar{b}})$. Indeed, the constant term of ACC is $-\mu$ if $\boldsymbol{c}$ encrypts 0, or μ if $\boldsymbol{c}$ encrypts $\frac{1}{2}$ as long as the noise of the ciphertext is small enough. Then, step 5 creates a new ciphertext $\overline{\boldsymbol{c}}$ by extracting the constant term of ACC and adding to it $(\mathbf{0}, \mu)$. That is, $\overline{\boldsymbol{c}}$ either encrypts 0 if $\boldsymbol{c}$ encrypts 0, or m if $\boldsymbol{c}$ encrypts $\frac{1}{2}$ (By choosing $m = \frac{1}{2}$, we get a fresh encryption of $\boldsymbol{c}$). Since a bootstrapping operation can be summarized as a $\mathsf{BlindRotate}$ over a noiseless TRLWE followed by a $\mathsf{KeySwitch}$, the bootstrapping noise ($\mathcal{V}_{BS}$) satisfies: $\mathcal{V}_{BS} < \mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1}$.

Algorithm 5 TFHE gate bootstrapping [28]

Input: a constant $m \in \mathbb{T}$, a TLWE sample $\boldsymbol{c} = (\boldsymbol{a}, b) \in \mathrm{TLWE}_{\boldsymbol{s}}(x \cdot \frac{1}{2})$ with $x \in \mathbb{B}$, a bootstrapping key $BK_{\boldsymbol{s} \to \boldsymbol{s'}} = (BK_i \in \mathrm{TRGSW}_{S'}(s_i))_{i \in [\![1,n]\!]}$ where S' is the TRLWE interpretation of a secret key $\boldsymbol{s'}$

Output: a TLWE sample $\overline{\boldsymbol{c}} \in \mathrm{TLWE}_{\boldsymbol{s}}(x.m)$

1: Let $\mu = \frac{1}{2}m \in \mathbb{T}$ (pick one of the two possible values)
2: Let $\bar{b} = \lfloor 2Nb \rceil$ and $\bar{a}_i = \lfloor 2Na_i \rceil \in \mathbb{Z},\ \forall i \in [\![1, n]\!]$
3: Let $testv := (1 + X + \cdots + X^{N-1}) \cdot X^{\frac{N}{2}} \cdot \mu \in \mathbb{T}_N[X]$
4: $ACC \leftarrow \mathsf{BlindRotate}((\mathbf{0}, testv), (\bar{a}_1, \ldots, \bar{a}_n, \bar{b}), (BK_1, \ldots, BK_n))$
5: $\overline{\boldsymbol{c}} = (\mathbf{0}, \mu) + \mathsf{SampleExtract}(ACC)$
6: return $\mathsf{KeySwitch}_{\boldsymbol{s'} \to \boldsymbol{s}}(\overline{\boldsymbol{c}})$

TFHE specifies a second type of bootstrapping called *circuit bootstrapping*. It converts TLWE samples into TRGSW samples and serves mainly for TFHE used in a leveled manner. This additional type of bootstrapping will not be discussed further in this paper.

4.3 TFHE Functional Bootstrapping

4.3.1 Encoding and Decoding

Our goal is to build a homomorphic LUT for any function $f : \mathbb{Z}_p \to \mathbb{Z}_p$ for any integer p. As we are using TFHE, every message from $\mathbb{Z}_p$ has to be encoded in $\mathbb{T}$. To that end, we use the encoding function:

$$E_p : \begin{array}{ccc} \mathbb{Z}_p & \to & \mathbb{T} \\ k & \mapsto & \frac{k}{p} \end{array}$$

and its corresponding decoding function:

$$D_p : \begin{array}{ccc} \mathbb{T} & \to & \mathbb{Z}_p \\ x & \mapsto & \lfloor x \cdot p \rceil \end{array}$$

Finally, we specify a torus-to-torus function $f_{\mathbb{T}}$ to get $f = D_p \circ f_{\mathbb{T}} \circ E_p$.

$$\begin{array}{ccc} \mathbb{Z}_p & \xrightarrow{f = D_p \circ f_{\mathbb{T}} \circ E_p} & \mathbb{Z}_p \\ E_p \downarrow & & \uparrow D_p \\ \mathbb{T} & \underset{f_{\mathbb{T}}}{\longrightarrow} & \mathbb{T} \end{array}$$

Since the function $f_{\mathbb{T}} = E_p \circ f \circ D_p$ makes the diagram commutative, we consider this function as the encoding of f over $\mathbb{T}$.

We use $m^{(p)}$ to refer to a message in $\mathbb{Z}_p$, and m to refer to $E_p(m^{(p)})$. That is, m is the representation of $m^{(p)}$ in $\mathbb{T}$ after discretization.

4.3.2 Functional Bootstrapping Idea

The original bootstrapping algorithm from [25] had already all the tools to implement a LUT of any negacyclic function[5]. In particular, TFHE is well-suited for $\frac{1}{2}$-antiperiodic function, as the plaintext space for TFHE is $\mathbb{T}$, where $[0, \frac{1}{2}[$ corresponds to positive values and $[\frac{1}{2}, 1[$ to negative ones, and the bootstrapping step 2 of the Algorithm 5 encodes elements from $\mathbb{T}$ into powers of X modulo $(X^N + 1)$, where $\forall \alpha \in [\![0, N[\![,\ X^{\alpha+N} \equiv -X^{\alpha} \bmod [X^N + 1]$.

Boura et al., [8] were the first to use the term *functional bootstrapping* for TFHE. They describe how TFHE bootstrapping computes a sign function. In addition,

[5]Negacyclic functions are antiperiodic functions over $\mathbb{T}$ with period $\frac{1}{2}$, i.e., verifying $f(x) = -f(x + \frac{1}{2})$.

they use bootstrapping to build a Rectified Linear Unit (ReLU). However, they do not delve into the details of how to implement the ReLU in practice[6].

Algorithm 6 describes a sign computation with the TFHE bootstrapping. It returns μ if m is positive (i.e., $m \in [0, \frac{1}{2}[$), and $-\mu$ if m is negative.

Algorithm 6 Sign extraction with bootstrapping

Input: a constant $\mu \in \mathbb{T}$, a TLWE sample $\boldsymbol{c} = (\boldsymbol{a}, b) \in \mathrm{TLWE}_{\boldsymbol{s}}(m)$ with $m \in \mathbb{T}$, a bootstrapping key $BK_{\boldsymbol{s}\to\boldsymbol{s'}} = (BK_i \in \mathrm{TRGSW}_{S'}(s_i))_{i\in[\![1,n]\!]}$ where S' is the TRLWE interpretation of a secret key $\boldsymbol{s'}$

Output: a TLWE sample $\overline{\boldsymbol{c}} \in \mathrm{TLWE}_{\boldsymbol{s}}(\mu.sign(m))$

1: Let $\bar{b} = \lfloor 2Nb \rceil$ and $\bar{a}_i = \lfloor 2Na_i \rceil \in \mathbb{Z}, \forall i \in [\![1, n]\!]$
2: Let $testv := (1 + X + \cdots + X^{N-1}) \cdot \mu \in \mathbb{T}_N[X]$
3: $ACC \leftarrow \mathsf{BlindRotate}((\mathbf{0}, testv), (\bar{a}_1, \ldots, \bar{a}_n, \bar{b}), (BK_1, \ldots, BK_n))$
4: $\overline{\boldsymbol{c}} = \mathsf{SampleExtract}(ACC)$
5: return $\mathsf{KeySwitch}_{\boldsymbol{s'}\to\boldsymbol{s}}(\overline{\boldsymbol{c}})$

When we look at the building blocks of Algorithm 6, we notice that we can build more complex functions just by changing the coefficients of the test polynomial $testv$. Indeed, if we consider $t = \sum_{i=0}^{N-1} t_i \cdot X^i$ where $t_i \in \mathbb{T}$ and $t^*(x)$ is the function:

$$t^* : \begin{array}{ccc} [\![-N, N-1]\!] & \to & \mathbb{T} \\ i & \mapsto & \begin{cases} t_i & \text{if } i \in [\![0, N[\![\\ -t_{i+N} & \text{if } i \in [\![-N, 0[\![\end{cases} \end{array},$$

the output of the bootstrapping of a TLWE ciphertext $[x]_{\mathbb{T}} = (\boldsymbol{a}, b)$ with the test polynomial $testv = t$ is $[t^*(\phi(\bar{\boldsymbol{a}}, \bar{b}))]_{\mathbb{T}}$, where $(\bar{\boldsymbol{a}}, \bar{b})$ is the rescaled version of $(\boldsymbol{a}, b)$ in $\mathbb{Z}_{2N}$ (line 1 of Algorithm 6).

Indeed, we first remind that for any positive integer i s.t. $0 \leq i < N$, we have:

$$testv.X^{-i} = t_i + \cdots - t_0 X^{N-i} - \cdots - t_{i-1} X^{N-1} \bmod [X^N + 1] \tag{4.1}$$

Then, we notice that BlindRotate (line 3 of Algorithm 6) computes $testv \cdot X^{-\phi(\bar{\boldsymbol{a}}, \bar{b})}$. Therefore, we get using equation (4.1) the following results:

- if $\phi(\bar{\boldsymbol{a}}, \bar{b}) \in [\![0, N[\![$, the constant term of $testv \cdot X^{-\phi(\bar{\boldsymbol{a}}, \bar{b})}$ is $t_{\phi(\bar{\boldsymbol{a}}, \bar{b})}$.
- if $\phi(\bar{\boldsymbol{a}}, \bar{b}) \in [\![-N, 0[\![$, we have:

[6] They build the function 2×ReLU from an absolute value function, but do not explain how to divide by two to get the ReLU result.

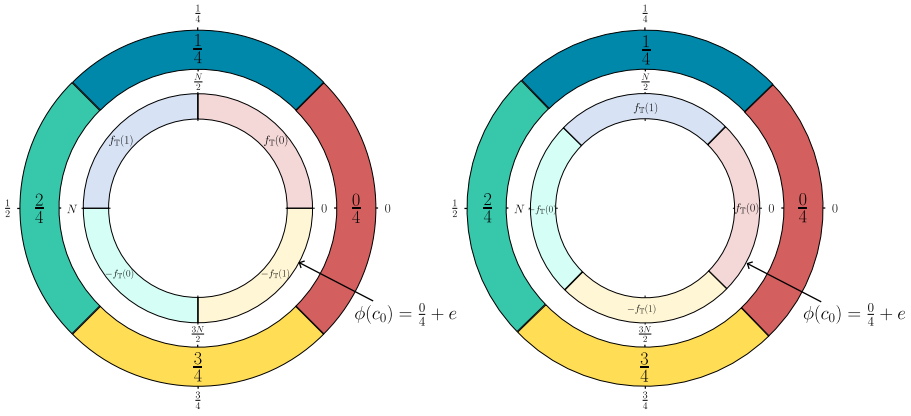


Figure 4.1 – Functional bootstrapping outputs with $\mathbb{Z}_4$ as plaintext space.

$$testv \cdot X^{-\phi(\bar{\boldsymbol{a}},\bar{b})} = -testv \cdot X^{-\phi(\bar{\boldsymbol{a}},\bar{b})-N} \bmod [X^N + 1]$$

with $(\phi(\bar{\boldsymbol{a}},\bar{b}) + N) \in [\![0, N[\![$. So, the constant term of $testv \cdot X^{-\phi(\bar{\boldsymbol{a}},\bar{b})}$ is $-t_{\phi(\bar{\boldsymbol{a}},\bar{b})+N}$.

All that remains for the bootstrapping algorithm is extracting the previous constant term (in line 4) and keyswitching (in line 5) to get the TLWE sample $[t^*(\phi(\bar{\boldsymbol{a}},\bar{b}))]_{\mathbb{T}}$.

Now, we can tweak the previous idea to evaluate discretized functions. Let $f : \mathbb{Z}_p \to \mathbb{Z}_p$ be any negacyclic function over $\mathbb{Z}_p$ and $f_{\mathbb{T}} = E_p \circ f \circ D_p$. We call $\tilde{f}$ the well-defined function $f_{\mathbb{T}} \circ E_{2N}$ that satisfies:

$$\begin{array}{rccl} \tilde{f} : & [\![-N, N-1]\!] & \to & \mathbb{T} \\ & x & \mapsto & \begin{cases} f_{\mathbb{T}}(\frac{x}{2N}) & \text{if } x \in [\![0, N[\![\\ -f_{\mathbb{T}}(\frac{x+N}{2N}) & \text{if } x \in [\![-N, 0[\![\end{cases} \end{array} \tag{4.2}$$

Let P_f be the polynomial $P_f = \sum_{i=0}^{N-1} \tilde{f}(i) \cdot X^i$. Now, if we apply the bootstrapping Algorithm 6 to a TLWE ciphertext $[m]_{\mathbb{T}} = (\boldsymbol{a}, b)$ with $m^{(p)} \in \mathbb{Z}_p$ and $testv = P_f$, it outputs $[\tilde{f}(\phi(\bar{\boldsymbol{a}},\bar{b}))]_{\mathbb{T}}$. That is, Algorithm 6 allows the encoding of the function f as long as $\frac{\phi(\bar{\boldsymbol{a}},\bar{b})}{2N} = m + e'$, for some e' small enough. Further details on the variance of e' and the error probability of the bootstrapping are given in Section 4.5.

4.3.3 Example of Functional Bootstrapping in $\mathbb{Z}_4$

As an example, let us consider the plaintext space $\mathbb{Z}_4$ and a negacyclic function f. We represent $\mathbb{Z}_4$ in $\mathbb{T}$ by the set $\{\frac{0}{4}, \frac{1}{4}, \frac{2}{4}, \frac{3}{4}\}$. We denote by $f_{\mathbb{T}}$ a function over $\mathbb{T}$

that satisfies: $f_{\mathbb{T}}(\frac{i}{4}) = \frac{f(i)}{4}$ for all $i \in [\![0, 3]\!]$. We consider a ciphertext $\boldsymbol{c}_0$ encrypting the value 0. We present in Algorithm 7 the functional bootstrapping algorithm that computes $LUT(f)$. We use the notation $P_{f,k}$ from Section 4.2.1.

Algorithm 7 TFHE functional bootstrapping example

Input: a TLWE sample $\boldsymbol{c} = (\boldsymbol{a}, b) \in \mathrm{TLWE}_{\boldsymbol{s}}(m)$ with $x \in \{\frac{0}{4}, \frac{1}{4}, \frac{2}{4}, \frac{3}{4}\}$, a bootstrapping key $BK_{\boldsymbol{s}\to\boldsymbol{s'}} = (BK_i \in \mathrm{TRGSW}_{S'}(s_i))_{i\in[\![1,n]\!]}$ where S' is the TRLWE interpretation of a secret key $\boldsymbol{s'}$

Output: a TLWE sample $\overline{\boldsymbol{c}} \in \mathrm{TLWE}_{\boldsymbol{s}}(f_{\mathbb{T}}(m))$

1: Let $\bar{b} = \lfloor 2Nb \rceil$ and $\bar{a}_i = \lfloor 2Na_i \rceil \in \mathbb{Z}, \forall i \in [\![1, n]\!]$
2: Let $testv := P_{f_{\mathbb{T}},4} \cdot X^{-\frac{N}{4}} \in \mathbb{T}_N[X]$
3: $ACC \leftarrow \mathsf{BlindRotate}((\mathbf{0}, testv), (\bar{a}_1, \ldots, \bar{a}_n, \bar{b}), (BK_1, \ldots, BK_n))$
4: $\overline{\boldsymbol{c}} = \mathsf{SampleExtract}(ACC)$
5: return $\mathsf{KeySwitch}_{\boldsymbol{s'}\to\boldsymbol{s}}(\overline{\boldsymbol{c}})$

In step 2 of Algorithm 7, we set the test polynomial $testv = P_{f_{\mathbb{T}},4} \cdot X^{-\frac{N}{4}}$, where $P_{f_{\mathbb{T}},4}$ encodes a look up table corresponding to $f_{\mathbb{T}}$, and $X^{-\frac{N}{4}}$ is an offset term.

In Figure 4.1, we describe the action of the offset $X^{-\frac{N}{4}}$ on $P_{f_{\mathbb{T}},4}$. We represent in the outer circle the possible phases associated to each entry from our plaintext space $\mathbb{Z}_4$. Meanwhile, we represent in the inner circle the returned coefficients after a bootstrapping. In the left part of Figure 4.1, we consider the result of the bootstrapping algorithm without the offset. We note that the red part of the inner and outer circles do not overlap. So, whenever the error term e in the phase is negative (even for small values of e), the considered functional bootstrapping outputs an incorrect value. In our example, the bootstrapping returns $f_{\mathbb{T}}(\frac{3}{4}) = -f_{\mathbb{T}}(\frac{1}{4})$ instead of $f_{\mathbb{T}}(0)$. Meanwhile, in the right part of the Figure 4.1, we consider the bootstrapping algorithm with the offset. Now, the red part of the inner and outer circles overlap, and so, the functional bootstrapping returns the right value as long as the error term remains small enough.

For a given plaintext space $\mathbb{Z}_p$, the offset is $X^{-\lfloor \frac{N}{p} \rceil}$. We assume from now on that p divides N to ease notations and formulas.

4.3.4 Multi-Value Functional Bootstrapping

Carpov et al., [16] introduced a nice method for evaluating multiple LUTs with one bootstrapping. They factor the test polynomial P_{f_i} associated to the function f_i into a product of two polynomials v_0 and v_i, where v_0 is a common factor to all

P_{f_i}. Indeed, they notice that:

$$(1 + X + \cdots + X^{N-1}) \cdot (1 - X) = 2 \bmod [X^N + 1] \tag{4.3}$$

Let $P_{f_i} = \sum_{j=0}^{N-1} \alpha_{i,j} X^j$ with $\alpha_{i,j} \in \mathbb{T}$, and $q \in \mathbb{N}^*$ the smallest integer so that: $\forall i$, $q \cdot (1 - X) \cdot P_{f_i} \in \mathbb{Z}[X]$ (q is a divisor of p). We get using equation (4.3):

$$\begin{aligned} P_{f_i} &= \frac{1}{2q} \cdot (1 + \cdots + X^{N-1}) \cdot (q \cdot (1 - X) \cdot P_{f_i}) \mod [X^N + 1] \\ &= v_0 \cdot v_i \bmod [X^N + 1] \end{aligned}$$

where:

$$\begin{aligned} v_0 &= \frac{1}{2q} \cdot (1 + \cdots + X^{N-1}) \\ v_i &= q \cdot (\alpha_{i,0} + \alpha_{i,N-1} + \sum_{j=1}^{N-1} (\alpha_{i,j} - \alpha_{i,j-1}) \cdot X^j) \end{aligned}$$

Thanks to this factorization, it becomes possible to compute many LUTs with one bootstrapping. Indeed, we just have to set the initial test polynomial to $testv = v_0$ during the bootstrapping. Then, after the BlindRotate, we multiply the obtained ACC by each v_i corresponding to LUT(f_i) to obtain ACC_i.

Algorithm 8 Multi-value bootstrapping

Input: a TLWE sample $\boldsymbol{c} = (\boldsymbol{a}, b) \in \text{TLWE}_{\boldsymbol{s}}(m)$ with $m \in \mathbb{T}$, a bootstrapping key $BK_{\boldsymbol{s} \to \boldsymbol{s'}} = (BK_i \in \text{TRGSW}_{S'}(s_i))_{i \in [\![1,n]\!]}$ where S' is the TRLWE interpretation of a secret key $\boldsymbol{s'}$, k LUTs s.t. $\text{LUT}(f_i) = v_0.v_i, \forall i \in [\![1, k]\!]$

Output: a list of k TLWE ciphertexts $\overline{\boldsymbol{c}_{\boldsymbol{i}}} \in \text{TLWE}_{\boldsymbol{s}}(f_i(\frac{\phi(\bar{\boldsymbol{a}}, \bar{b})}{2N}))$

1: Let $\bar{b} = \lfloor 2Nb \rceil$ and $\bar{a}_i = \lfloor 2Na_i \rceil \in \mathbb{Z}, \forall i \in [\![1, n]\!]$
2: Let $testv := v_0$
3: $\text{ACC} \leftarrow \mathsf{BlindRotate}((\mathbf{0}, testv), (\bar{a}_1, \ldots, \bar{a}_n, \bar{b}), (BK_1, \ldots, BK_n))$
4: **for** $i \leftarrow 1$ to k **do**
5: $\quad \text{ACC}_i := \text{ACC} \cdot v_i$
6: $\quad \overline{\boldsymbol{c}_{\boldsymbol{i}}} = \mathsf{SampleExtract}(\text{ACC}_i)$
7: $\quad$ return $\mathsf{KeySwitch}_{\mathsf{s'} \to \mathsf{s}}(\overline{\boldsymbol{c}_{\boldsymbol{i}}})$

4.4 Look-Up-Tables over a Single Ciphertext

In Section 4.3.2, we demonstrated that functional bootstrapping can serve to compute LUT(f) for any negacyclic function f. In this section, we describe 4 different

ways to specify homomorphic LUTs for *any* function (i.e., not necessarily negacyclic ones). We present 3 solutions from the state of the art [30, 57, 100] in Sections 4.4.1, 4.4.2 and 4.4.3, and our novel method ComBo in Section 4.4.4. In addition, we discuss a solution to reduce the noise of the functional bootstrapping from [57] in Section 4.4.2.

As in Section 4.3.1, we call $f_{\mathbb{T}} : \mathbb{T} \to \mathbb{T}$ the function that specifies our homomorphic LUT, and $f : \mathbb{Z}_p \to \mathbb{Z}_p$ its corresponding function over the input and output space $\mathbb{Z}_p$.

4.4.1 Partial Domain Functional Bootstrapping – Half-Torus

The Half-Torus method gets around the negacyclic restriction of functional bootstrapping by encoding values only on $[0, \frac{1}{2}[$ (i.e., half of the torus). Let's consider the test polynomial P_h for a given negacyclic function h. Recall Equation 4.2 that defines the output of the bootstrapping operation as:

$$\tilde{h} : \begin{array}{ccc} [\![-N, N-1]\!] & \to & \mathbb{T} \\ x & \mapsto & \begin{cases} h(\frac{x}{2N}) & \text{if } x \in [\![0, N[\![\\ -h(\frac{x+N}{2N}) & \text{if } x \in [\![-N, 0[\![\end{cases} \end{array}$$

As we restrict the encoding space to $[0, \frac{1}{2}[$, we also restrict $\tilde{h}$ domain to $[\![0, N[\![$, where h has no negacyclic property. That is, we get a method to evaluate a LUT with a *single* bootstrapping.

4.4.2 Full Domain Functional Bootstrapping – FDFB

Kluczniak and Schild [57] specified FDFB to evaluate encrypted LUTs of domain the full torus $\mathbb{T}$. Let's consider a TLWE ciphertext $[m]_{\mathbb{T}}$ given a message $m^{(p)} \in \mathbb{Z}_p$. We denote by g the function:

$$g : \begin{array}{ccc} \mathbb{T} & \to & \mathbb{T} \\ x & \mapsto & -f_{\mathbb{T}}(x + \frac{1}{2}) \end{array}$$

We denote by $q \in \mathbb{N}^*$ the smallest integer such that $q \cdot (P_f - P_g)$ is a polynomial with coefficients in $\mathbb{Z}$. Then, we define $P_1 = q \cdot P_f$ and $P_2 = q \cdot P_g$. We note that the coefficients of $P_f - P_g$ are multiples of $\frac{1}{p}$ in $\mathbb{T}$, where $\mathbb{T}$ corresponds to $[-\frac{1}{2}, \frac{1}{2}[$. We note that q is a divisor of p and $P_2 - P_1$ has coefficients of norm lower or equal to $\frac{q}{2}$.

We define the Heaviside function H as:

$$H : x \mapsto \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

We can express H by using the sign function as follows: $H(x) = \frac{\text{sign}(x)+1}{2}$.

In order to evaluate a LUT, we first compute $[E_q(H(m))]_{\mathbb{T}}$ with one bootstrapping (using Algorithm 6) and deduce $[E_q((1-H)(m))]_{\mathbb{T}} = (\mathbf{0}, \frac{1}{q}) - [E_q(H(m))]_{\mathbb{T}}$. Then, we make a keyswitch to transform the TLWE sample $[E_q((1-H)(m))]_{\mathbb{T}}$ into a TRLWE sample $[E_q((1-H)(m))]_{\mathbb{T}_N[X]}$. Finally, we define:

$$\boldsymbol{c}_{\text{LUT}} = (P_2 - P_1) \cdot [E_q((1-H)(m))]_{\mathbb{T}_N[X]} + (\mathbf{0}, P_f)$$

such that:

$$\boldsymbol{c}_{\text{LUT}} = \begin{cases} [P_f]_{\mathbb{T}_N[X]} & \text{if } m \geq 0 \\ [P_g]_{\mathbb{T}_N[X]} & \text{if } m < 0 \end{cases}$$

We note that depending on the sign of m, $\boldsymbol{c}_{\text{LUT}}$ is a TRLWE encryption of P_f or P_g, the test polynomials of f or g, respectively. As such, we obtain $[f_{\mathbb{T}}(m)]_{\mathbb{T}}$ after a second bootstrapping with $[m]_{\mathbb{T}}$ as input and $\boldsymbol{c}_{\text{LUT}}$ as a test polynomial.

We can reduce the noise of $\boldsymbol{c}_{\text{LUT}}$ by applying to P_f and P_g the factorization described in Section 4.3.4. First, we replace the polynomials P_f and P_g by $v_f = (1-X) \cdot P_f$ and $v_g = (1-X) \cdot P_g$, respectively. Thanks to the redundancy of the coefficients of P_f and P_g, v_f and v_g have at most $\frac{p}{2}$ non null coefficients. We denote by $q' \in \mathbb{N}^*$ the smallest integer such that $q' \cdot (v_f - v_g)$ is a polynomial with coefficients in $\mathbb{Z}$. We ensure that $q' \leq q$ as $q \cdot (1-X) \cdot (P_f - P_g) = (1-X) \cdot (q \cdot (P_f - P_g))$ has coefficients in $\mathbb{Z}$. Then, we define $v_1 = q' \cdot v_f$ and $v_2 = q' \cdot v_g$. We get that $v_2 - v_1$ has coefficients in $\mathbb{Z}$ of norm lower or equal to q'. Finally, we compute a TRLWE encryption of $\sum_{i=0}^{N-1} X^i \cdot E_{2 \cdot q'}((1-H)(m))$ from the TLWE sample $[E_{2 \cdot q'}((1-H)(m))]_{\mathbb{T}}$, by applying a KeySwitch. We get:

$$\boldsymbol{c}_{\text{LUT}} = (v_2 - v_1) \cdot [\sum_{i=0}^{N-1} X^i \cdot E_{2 \cdot q'}((1-H)(m))]_{\mathbb{T}_N[X]} + (\mathbf{0}, P_f)$$

such that:

$$\boldsymbol{c}_{\text{LUT}} = \begin{cases} [P_f]_{\mathbb{T}_N[X]} & \text{if } m \geq 0 \\ [P_g]_{\mathbb{T}_N[X]} & \text{if } m < 0 \end{cases}$$

4.4.3 Full Domain Functional Bootstrapping – TOTA

Both Liu et al., [64] and Yan et al., [100] independently proposed the same approach[7] to evaluate arbitrary functions over the torus using a functional bootstrapping. As such, we refer to both methods in this paper with the name TOTA (as

[7] Although both papers use different notations, both methods rescale the message space into the first half of the torus before applying a half torus functional bootstrapping. In both cases, a sign evaluation is performed to compute that rescaling.

proposed by Yan et al.). Let's consider a ciphertext $[m_1]_{\mathbb{T}} = (\boldsymbol{a}, b = \langle \boldsymbol{a}, \boldsymbol{s} \rangle + m_1 + e)$. Then, by dividing each coefficient of this ciphertext by 2, we get a ciphertext $[m_2]_{\mathbb{T}} = (\frac{\boldsymbol{a}}{2}, \langle \frac{\boldsymbol{a}}{2}, \boldsymbol{s} \rangle + m_2 + \frac{e}{2})$ where $m_2 = \frac{m_1}{2} + \frac{k}{2}$ with $k \in \{0, 1\}$ and $\frac{m_1}{2} \in [0, \frac{1}{2}[$. Using the original bootstrapping algorithm, we compute $[\frac{\text{sign}(m_2)}{4}]_{\mathbb{T}}$ an encryption of $\frac{\text{sign}(m_2)}{4} = \begin{cases} \frac{1}{4} & \text{if } k = 0 \\ -\frac{1}{4} & \text{if } k = 1 \end{cases}$. Then, we get an encryption of $\frac{m_1}{2}$ by computing: $[m_2]_{\mathbb{T}} - [\frac{\text{sign}(m_2)}{4}]_{\mathbb{T}} + (\mathbf{0}, \frac{1}{4})$.

For any function $f_{\mathbb{T}}$, let's define $f_{(2)}$ such that $f_{(2)}(x) = f_{\mathbb{T}}(2x)$. Since $\frac{m_1}{2} \in [0, \frac{1}{2}[$, we can compute $f_{(2)}(\frac{m_1}{2})$ with a single bootstrapping using the partial domain approach from 4.4.1, and $f_{(2)}(\frac{m_1}{2}) = f_{\mathbb{T}}(m_1)$.

4.4.4 Full Domain Functional Bootstrapping with Composition - ComBo

In this section, we present ComBo, a novel method to compute any function using the full (discretized) torus as plaintext space. We will assume that p is even and fixed[8].

Pseudo odd functions: We call pseudo odd function a function f that satisfies: $\forall x \in \mathbb{Z}_p,\ f(-x-1) = -f(x)$.

Let f be a pseudo odd function over $\mathbb{Z}_p$. We define the following negacyclic functions:

$$f_{\text{neg}} : \begin{array}{ccc} [\![0, p-1]\!] & \to & \mathbb{Z}_p \\ x & \mapsto & \begin{cases} f(x) & \text{if } x \in [\![0, \frac{p}{2} - 1]\!] \\ -f(x - \frac{p}{2}) & \text{if } x \in [\![\frac{p}{2}, p-1]\!] \end{cases} \end{array}$$

and

$$\text{Id}_{\text{neg}} : \begin{array}{ccc} [\![0, p-1]\!] & \to & \mathbb{R}_p \\ x & \mapsto & \begin{cases} x + \frac{1}{2} & \text{if } x \in [\![0, \frac{p}{2} - 1]\!] \\ \frac{p}{2} - x - \frac{1}{2} & \text{if } x \in [\![\frac{p}{2}, p-1]\!] \end{cases} \end{array}$$

Since these 2 functions are negacyclic, they can be computed with the usual negacyclic functional bootstrapping (presented in section 4.3.2).

Note that $(\text{Id}_{\text{neg}} - \frac{1}{2})$ is a bijection of $\mathbb{Z}_p$ that satisfies the equality $(\text{Id}_{\text{neg}} - \frac{1}{2})(x) = x$, for all $x \in [\![0, \frac{p}{2} - 1]\!]$. Otherwise, for all $x \in [\![\frac{p}{2}, p-1]\!]$, $(\text{Id}_{\text{neg}} - \frac{1}{2})(x) = \frac{p}{2} - x - 1$. In $\mathbb{Z}_p$, $\forall x \in [\![\frac{p}{2}, p-1]\!]$, we have $(\frac{p}{2} - x - 1) \in [\![\frac{p}{2}, p-1]\!]$.

Now, we compose it with f_{neg} to obtain: $f_{\text{neg}} \circ (\text{Id}_{\text{neg}} - \frac{1}{2})(x) = f_{\text{neg}}(x) = f(x)$ if $x \in [\![0, \frac{p}{2} - 1]\!]$. If $x \in [\![\frac{p}{2}, p-1]\!]$, $f_{\text{neg}} \circ (\text{Id}_{\text{neg}} - \frac{1}{2})(x) = f_{\text{neg}}(\frac{p}{2} - x - 1) = -f(-x-1)$. Since f is pseudo odd, we have: $-f(-x-1) = f(x)$.

[8] If p is odd, we set $p := p + 1$ to get back to the assumption that p is even.

Pseudo even functions: We call pseudo even function a function f that satisfies: $\forall x \in \mathbb{Z}_p,\ f(-x-1) = f(x)$.

Let f be a pseudo even function over $\mathbb{Z}_p$. We define the following negacyclic functions:

$$f_{\text{neg}} : \begin{array}{ccc} [\![0, p-1]\!] & \to & \mathbb{Z}_p \\ x & \mapsto & \begin{cases} f(x) & \text{if } x \in [\![0, \frac{p}{2}-1]\!] \\ -f(x-\frac{p}{2}) & \text{if } x \in [\![\frac{p}{2}, p-1]\!] \end{cases} \end{array}$$

and

$$\text{abs}_{\text{neg}} : \begin{array}{ccc} [\![0, p-1]\!] & \to & \mathbb{R}_p \\ x & \mapsto & \begin{cases} x + \frac{p}{4} + \frac{1}{2} & \text{if } x \in [\![0, \frac{p}{2}-1]\!] \\ \frac{p}{4} - x - \frac{1}{2} & \text{if } x \in [\![\frac{p}{2}, p-1]\!] \end{cases} \end{array}$$

Since these 2 functions are also negacyclic, they can similarly be computed with the usual negacyclic functional bootstrapping (presented in section 4.3.2).

Note that $(\text{abs}_{\text{neg}} - \frac{p}{4} - \frac{1}{2})$ satisfies the equality $(\text{abs}_{\text{neg}} - \frac{p}{4} - \frac{1}{2})(x) = x$ for all $x \in [\![0, \frac{p}{2}-1]\!]$. However, if $x \in [\![\frac{p}{2}, p-1]\!]$, $(\text{abs}_{\text{neg}} - \frac{p}{4} - \frac{1}{2})(x) = -x-1 \in [\![0, \frac{p}{2}-1]\!]$. As such, we ensure that the function $(\text{abs}_{\text{neg}} - \frac{p}{4} - \frac{1}{2})$ behaves similarly to the absolute value function.

It follows that $f_{\text{neg}} \circ (\text{abs}_{\text{neg}} - \frac{p}{4} - \frac{1}{2})(x) = f_{\text{neg}}(x) = f(x)$ if $x \in [\![0, \frac{p}{2}-1]\!]$. If $x \in [\![\frac{p}{2}, p-1]\!]$, $f_{\text{neg}} \circ (\text{abs}_{\text{neg}} - \frac{p}{4} - \frac{1}{2})(x) = f_{\text{neg}}(-x-1) = f(-x-1)$. Since f is pseudo even, we have $f(-x-1) = f(x)$.

Any function: We write any function $f \in \mathbb{Z}_p$ as a sum of a pseudo even function and a pseudo odd function: $f(x) = f_{\text{even}}(x) + f_{\text{odd}}(x)$, where $f_{\text{even}}(x) = \frac{f(x)+f(-x-1)}{2}$ and $f_{\text{odd}}(x) = \frac{f(x)-f(-x-1)}{2}$. Besides, we build any pseudo odd or pseudo even function with at most 2 bootstrappings. So, we can build any function with at most 4 bootstrappings.

We describe in Algorithm 9 the overall algorithm for running ComBo. We denote by $\mathsf{FB}[f]((\boldsymbol{a}, b))$ the application of the negacyclic functional bootstrapping procedure using the test vector $P_{\frac{f}{p}}$ (as defined in Section 4.2.1) and applied to a ciphertext $(\boldsymbol{a}, b)$ given a function $f : \mathbb{Z}_p \to \mathbb{R}_p$.

Correctness: If we assume that the negacyclic functional bootstrapping (FB) is correct, we obtain by Algorithm 9 a ciphertext $[\frac{f(m)}{p}]_{\mathbb{T}}$ where $m \in \mathbb{Z}_p$ is the input of the algorithm and $f : \mathbb{Z}_p \to \mathbb{Z}_p$ is the target function. Indeed, Step 1 computes an encryption of $\frac{\text{Id}_{\text{neg}}(m)}{p}$ since Id_{neg} is a negacyclic function. Step 2 computes an encryption of $\frac{\text{Id}_{\text{neg}}(m)-\frac{1}{2}}{p} = \frac{(\text{Id}_{\text{neg}}-\frac{1}{2})(m)}{p}$. Let us refer by f_{neg} to the negacyclic function corresponding to f_{odd} over $[\![0, \frac{p}{2}-1]\!]$. Then Step 3 computes an encryption

Algorithm 9 ComBo

Input: a TLWE sample $[\frac{m}{p}]_{\mathbb{T}} \in \mathrm{TLWE}_{\boldsymbol{s}}(\frac{m}{p})$ with $m \in \mathbb{Z}_p$, a bootstrapping key $BK_{\boldsymbol{s}\to\boldsymbol{s'}} = (BK_i \in \mathrm{TRGSW}_{S'}(s_i))_{i\in[\![1,n]\!]}$ where S' is the TRLWE interpretation of a secret key $\boldsymbol{s'}$, a target function $f : \mathbb{Z}_p \to \mathbb{Z}_p$, and the two functions $f_{\mathrm{odd}} : \begin{array}{ccc} \mathbb{Z}_p & \to & \mathbb{R}_p \\ x & \mapsto & \frac{f(x)-f(-x-1)}{2} \end{array}$ and $f_{\mathrm{even}} : \begin{array}{ccc} \mathbb{Z}_p & \to & \mathbb{R}_p \\ x & \mapsto & \frac{f(x)+f(-x-1)}{2} \end{array}$

Output: a TLWE ciphertext $(\boldsymbol{a'}, b') = [\frac{f(m)}{p}]_{\mathbb{T}} \in \mathrm{TLWE}_{\boldsymbol{s}}(\frac{f(m)}{p})$

1: $(\boldsymbol{a}, b) = \mathsf{FB}[\mathrm{Id}_{\mathrm{neg}}]([\frac{m}{p}]_{\mathbb{T}})$ ▷ Start of pseudo odd computation
2: $(\boldsymbol{a}, b) = (\boldsymbol{a}, b - \frac{1}{2p})$
3: $(\boldsymbol{a}_{\mathrm{odd}}, b_{\mathrm{odd}}) = \mathsf{FB}[f_{\mathrm{odd}}]((\boldsymbol{a}, b))$ ▷ End of pseudo odd computation
4: $(\boldsymbol{a}, b) = \mathsf{FB}[\mathrm{abs}_{\mathrm{neg}}]([\frac{m}{p}]_{\mathbb{T}})$ ▷ Start of pseudo even computation
5: $(\boldsymbol{a}, b) = (\boldsymbol{a}, b - \frac{1}{2p} - \frac{1}{4})$
6: $(\boldsymbol{a}_{\mathrm{even}}, b_{\mathrm{even}}) = \mathsf{FB}[f_{\mathrm{even}}]((\boldsymbol{a}, b))$ ▷ End of pseudo even computation
7: $(\boldsymbol{a'}, b') = (\boldsymbol{a}_{\mathrm{odd}}, b_{\mathrm{odd}}) + (\boldsymbol{a}_{\mathrm{even}}, b_{\mathrm{even}})$

of $\frac{f_{\mathrm{neg}}\circ(\mathrm{Id}_{\mathrm{neg}}-\frac{1}{2})(m)}{p}$: the encoding of $f_{\mathrm{odd}}(m)$ over $\mathbb{T}$ (as discussed in the paragraph about pseudo odd functions in Section 4.4.4). Similarly, Steps 4 to 6 compute an encryption of the encoding of $f_{\mathrm{even}}(m)$ over $\mathbb{T}$. Finally, Step 7 computes the sum of the pseudo odd and pseudo even outputs which results in an encryption of $\frac{f(m)}{p}$: the encoding of $f(m)$ over $\mathbb{T}$.

In practice, we can reduce the (single-shot) computation time by using parallelism (e.g. multithreading or SIMD) for evaluating the pseudo odd and pseudo even functions simultaneously. So, we end-up with a computation time of 2 bootstrappings. We can alternatively reduce the number of bootstrappings to 3 thanks to the multi-value functional bootstrapping (see Section 4.3.4).

From now on, we call ComBoMV the ComBo method when used with the multi-value bootstrapping, and ComboP with parallelism.

Examples: We describe how to build the functions Id, and ReLU with ComBo.

For Id, the decomposition in pseudo even and pseudo odd functions gives $\mathsf{Id}(x) = (-\frac{1}{2}) + (x + \frac{1}{2})$. The pseudo even function $\mathsf{Id}_{\mathrm{even}} = -\frac{1}{2}$ is a constant and does not require any bootstrapping. We only have to compute the pseudo odd function $\mathsf{Id}_{\mathrm{odd}} = x + \frac{1}{2}$. In this case, we have no need for multithreading or multi-value bootstrapping.

For ReLU, the decomposition gives $\mathsf{ReLU}(x) = \mathsf{ReLU}_{\mathrm{even}}(x) + \mathsf{ReLU}_{\mathrm{odd}}(x)$ where:

$$\mathsf{ReLU}_{\mathrm{even}} : x \mapsto \begin{cases} \frac{x}{2} & \text{if } x \in [\![0, \frac{p}{2} - 1]\!] \\ -\frac{x}{2} - \frac{1}{2} & \text{otherwise} \end{cases}$$

$$\mathsf{ReLU}_{\text{odd}} : x \mapsto \begin{cases} \frac{x}{2} & \text{if } x \in [\![0, \frac{p}{2} - 1]\!] \\ \frac{x}{2} + \frac{1}{2} & \text{otherwise} \end{cases}$$

Applying ComBo naively results in 4 bootstrappings. However, we can actually compute $\mathsf{ReLU}_{\text{even}}$ with only 1 bootstrapping as for abs_{neg}. This specific improvement is useful for ComBo, as it reduces the number of consecutive bootstrappings to 3.

4.5 Error rate and noise variance

In this section, we analyze the noise variance and error rate for the aforementioned functional bootstrapping methods. We refer to each bootstrapping method by its acronym as defined in Section 4.4.

4.5.1 Noise variance

The noise variance of a bootstrapped ciphertext depends on the operations applied to the input ciphertext during the bootstrapping. Table 4.1 gives the theoretical variance of each of these operations. These formulas are taken from [28].

Operation	**Variance**
$\boldsymbol{c}_i + \boldsymbol{c}_j$	$V_i + V_j$
$\boldsymbol{C}_i + \boldsymbol{C}_j$	$V_i + V_j$
$P \cdot \boldsymbol{C}_i$	$\|\|P\|\|_2^2 \cdot V_i$
$\mathsf{Keyswitch}(\boldsymbol{c}_i)$	$V_i + \mathcal{E}_{KS}^{n,\overline{N}}$
$\mathsf{BlindRotate}(\boldsymbol{C}_i, v)$	$V_i + \mathcal{E}_{BR}$
$\mathsf{Bootstrap}(\boldsymbol{c}_i)$	$\mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1}$

Table 4.1 – Obtained noise variances when applying basic operations to independent inputs: $\boldsymbol{c}_i$ is a TLWE ciphertext of variance V_i, $\boldsymbol{C}_i$ is a TRLWE ciphertext of variance V_i, P is a plaintext polynomial and $v \in \mathbb{Z}_{2N}^{n+1}$.

Each of the bootstrapping methods of Section 4.4 relies on a composition of the operations from Table 4.1. So, we compute their resulting variances in Table 4.2 by simply composing the formulas from Table 4.1.

Bootstrapping	Variance
Half-Torus	$\mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1}$
FDFB	$\|\|v_2 - v_1\|\|_2^2 \cdot (\mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1} + \mathcal{E}_{KS}^{n,N}) + \mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1}$
TOTA	$\mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1}$
ComBo & ComBoP	$2 \cdot (\mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1})$
ComBoMV	$(\|\|v_1\|\|_2^2 + \|\|v_2\|\|_2^2) \cdot \mathcal{E}_{BR} + 2 \cdot \mathcal{E}_{KS}^{N,1}$

Table 4.2 – Output noise variance of the aforementioned functional bootstrapping methods

We identify in Table 4.2 two kinds of functional bootstrapping algorithms. On the one hand, we have functional bootstrapping algorithms that do not use any intermediary polynomial multiplication and end-up with a similar noise growth to a gate bootstrapping. On the other hand, we have functional bootstrapping algorithms that have a quadratic growth of the output noise variance with respect to the norm of the used test polynomial. For this second category, we can reduce the output noise by using the factorization technique described in Section 4.3.4.

4.5.2 Probability of Error

We discuss in this section the probabilities of error of all the functional bootstrapping methods from Section 4.4. Similar approaches to compute the probability of error of functional bootstrapping can be found in [5] and [48].

We first consider a single BlindRotate operation given a message $m^{(p)} \in \mathbb{Z}_p$, a TLWE ciphertext $(\boldsymbol{a}, b)$ where $b = (\langle \boldsymbol{a}, s \rangle + m + e)$, and a TRLWE ciphertext $(\mathbf{0}, t)$, where t is the test polynomial. Following the notation from Section 4.3.1, we have $m = E_p(m^{(p)})$.

As mentioned in Section 4.3.2, applying a BlindRotate and extracting the first coefficient outputs $[t^*(\phi(\bar{\boldsymbol{a}}, \bar{b}))]_{\mathbb{T}}$. Hence, we need the equality $[t^*(\phi(\bar{\boldsymbol{a}}, \bar{b}))] = [f\,(m)]$ to hold true for any message $m^{(p)}$ in order to compute $\mathrm{LUT}_p(f)$ for a given negacyclic function f. To that end, we consider $t = P_{f,p} \cdot X^{-\frac{N}{p}}$ assuming that p divides N (we motivated this choice in Figure 4.1 and Section 4.3.3). Note that $\phi(\bar{\boldsymbol{a}}, \bar{b}) = 2N \cdot (m + e + r) \bmod [2N]$ where r is an error introduced when scaling and rounding the coefficients of $(\boldsymbol{a}, b)$ from $\mathbb{T}$ to $\mathbb{Z}_{2N}$. Thus, we have:

$$[t^*(\phi(\bar{\boldsymbol{a}}, \bar{b}))] = \left[f\left(\frac{\left\lfloor \frac{p \cdot (\phi(\bar{\boldsymbol{a}}, \bar{b}) + \frac{N}{p})}{2N} \right\rfloor}{p} \right) \right] = \left[f\left(\frac{\lfloor p \cdot (m + e + r) + \frac{1}{2} \rfloor}{p} \right) \right]$$

It follows that $[t^*(\phi(\bar{\boldsymbol{a}}, \bar{b}))] = [f(m)]$ as long as $|e + r| < \frac{1}{2p}$. The error r follows a translated Irwin-Hall distribution with variance $\frac{n+1}{48 \cdot N^2}$ that, as is well known, can be closely approximated by a centered Gaussian distribution. With the assumptions that e and r are independent random variables, the probability that $|e + r| < \frac{1}{2p}$ is $\mathcal{P}(\frac{1}{2p}, V_c + V_r)$, where V_c and V_r are respectively the variances of the ciphertext and r, and $\mathcal{P}$ is the notation introduced in Section 4.2.1. The probability of error is then $1 - \mathcal{P}(\frac{1}{2p}, V_c + V_r)$.

When multiple BlindRotate operations occur during a functional bootstrapping, each of them must succeed to ensure a correct computation. We can use the well known formulas of probabilities for independent or correlated events to find the overall probability of error of a functional bootstrapping method.

The probabilities of success of the functional bootstrapping methods from Section 4.4 are summarized in Table 4.3. We denote by:

$$V = \mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1}$$

the variance of a simple gate bootstrapping, and by:

$$V_i = ||v_i||_2^2 \cdot \mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1}$$

the variance of a bootstrapping using an intermediary polynomial multiplication.

Bootstrapping	**Probability of success**
Half-Torus	$\mathcal{P}(\frac{1}{4 \cdot p}, V_c + V_r)$
FDFB	$\mathcal{P}(\frac{1}{2 \cdot p}, V_c + V_r)$
TOTA	$\mathcal{P}(\frac{1}{4}, V_c + V_r) \cdot \mathcal{P}(\frac{1}{4 \cdot p}, \frac{V_c}{4} + V_r + V)$
ComBo & ComBoP	$\mathcal{P}(\frac{1}{2 \cdot p}, V_c + V_r) \cdot \mathcal{P}(\frac{1}{2 \cdot p}, V + V_r)^2$
ComBoMV	$\mathcal{P}(\frac{1}{2 \cdot p}, V_c + V_r) \cdot \prod_{i=0}^{1} \mathcal{P}(\frac{1}{2 \cdot p}, V_i + V_r)$

Table 4.3 – Probability of success for each functional bootstrapping method with plaintext size p

The variances and the value of p given as inputs to the formulas of Table 4.3 have a high impact on the error rate. Indeed, $1 - \mathcal{P}(a, V)$ gets exponentially closer to 0

when a increases or when V decreases. For example, for a given p and V, the error rate of the Half-Torus method (i.e., $(1 - \mathcal{P}(\frac{1}{4p}, V))$) is higher than the probability of error of FDFB $(1 - \mathcal{P}(\frac{1}{2p}, V))$.

4.6 Experimental Results

In this section, we compare the computation time and the error rate for the functional bootstrapping methods of Section 4.4. We wrap up this section with a time-error trade-off analysis. All experiments[9] were implemented on an Intel Core i5-8250U CPU @ 1.60GHz by building on the TFHE open source library[10].

4.6.1 Parameters

We present in Table 4.4 the parameter sets used for our tests. We generate these parameters by following the guidelines below:

- We fix the security level λ to 128 bits, which is the lowest security level considered as secure by present day standard.
- For efficiency, we want N to be a small power of 2. We notice that for $N = 512$, the noise level required for ensuring security is too large to compute properly a functional bootstrapping. Thus, we choose $N = 1024$, which is the default value for the degree of the cyclotomic polynomial with TFHE.
- We note $\sigma_{\mathbb{T}_N[X]}$ the standard deviation used for the noise of the bootstrapping key and the keyswitch key from TLWE to TRLWE. We use the lattice-estimator [2] to set $\sigma_{\mathbb{T}_N[X]}$ as low as possible with respect to the security level λ. Thus, $\sigma_{\mathbb{T}_N[X]} = 5.6 \cdot 10^{-8}$.
- For efficiency, we choose values of n lower than N. As such, we generate sets of parameters for all n between 700 and 1024 by step of 100.
- We note $\sigma_{\mathbb{T}}$ the standard deviation used for the noise of the keyswitch key from TLWE to TLWE and fresh ciphertexts. For each n, we use the lattice-estimator to set $\sigma_{\mathbb{T}}$ as low as possible with respect to the security level λ.

The remaining parameters, present in Table 4.4, are unrelated to the security level of the cryptosystem. We choose them using the following guidelines:

[9] Code available at: https://github.com/CEA-LIST/Cingulata/experiments/tfhe-funcbootstrap-experiments.zip.

[10] https://github.com/tfhe/tfhe

- We consider the Half-Torus method as the baseline for the error rate of each method. As such, we tailor sets of parameters to reach an error rate close to 2^{-30} using the Half-Torus method for a plaintext space of $p = 8$.
- For faster bootstrapping operations, we need to have l as low as possible. We still need to select l high enough to reach the target error rate.
- For given l, n, N, and $\sigma_{\mathbb{T}_N[X]}$, we choose B_g to minimize the noise of the BlindRotate.
- For lower noise, we need B_{KS} to be as high as possible. Since the size of the keys grows with the basis, we set it to 1024 to avoid memory issues.
- For faster keyswitching operations, we need to have t as low as possible. We still need to select t high enough to reach the target error rate. Given the choice of B_{KS}, we find that $t = 2$ is the optimal choice.

Set	**n**	**l**	$\boldsymbol{B_g}$	$\boldsymbol{\sigma_{\mathbb{T}}}$	$\boldsymbol{\sigma_{\mathbb{T}_N[X]}}$
1	1024	5	16	$5.6e^{-08}$	$5.6e^{-08}$
2	1024	4	32	$5.6e^{-08}$	$5.6e^{-08}$
3	900	4	32	$5.1e^{-07}$	$5.6e^{-08}$
4	900	3	64	$5.1e^{-07}$	$5.6e^{-08}$
5	800	4	32	$3.1e^{-06}$	$5.6e^{-08}$
6	800	3	64	$3.1e^{-06}$	$5.6e^{-08}$
7	700	4	32	$1.9e^{-05}$	$5.6e^{-08}$
8	700	3	64	$1.9e^{-05}$	$5.6e^{-08}$

Table 4.4 – Selected parameter sets with $p = 8$, $N = 1024$, $B_{KS} = 1024$, $t = 2$, and $\lambda = 128$, following the guidelines of Section 4.6.1

4.6.2 Error Rate

In this section, we compute the probability of error for the functional bootstrapping methods of Section 4.4 with respect to every set of parameters described in Table 4.4.

In order to have a fair evaluation of the ability to consecutively bootstrap with the same method, we assume that the input to each method immediately follows a bootstrapping with the same method. We present in Table 4.5 the obtained error rates with respect to each method.

We note that the error rate of each method does not depend on the function computed during the bootstrapping except for FDFB and ComBoMV. Thus, we define a dedicated analysis methodology for these methods:

- For FDFB, we evaluate the error rate for the functions Id and ReLU as well as the worst case that maximizes the output noise. Since we use the multi-value bootstrapping factorization (described in Section 4.3.4), the worst case test polynomial $v_2 - v_1$ has $\frac{p}{2}$ non-zero values each equal to p. If we apply the FDFB error variance formula from Table 4.2, we obtain the worst case noise bound for the output ciphertext: $\frac{p^3}{2} \cdot (\mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1} + \mathcal{E}_{KS}^{n,N}) + \mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1}$.
- For ComBoMV, we follow the decomposition $f_{\text{odd,neg}} \circ \text{Id}_{\text{neg}} + f_{\text{even,neg}} \circ \text{abs}_{\text{neg}}$ given in Section 4.4.4, and use a multi-value bootstrapping to compute Id_{neg} and abs_{neg} at the same time. As such, the error rate becomes independent from the computed function.

Set		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Half-Torus		34	28	32	20	36	23	39	25
TOTA		33	27	30	18	34	20	36	22
FDFB	**Worst**	7	3	3	1	3	1	3	1
	Id	55	27	31	11	34	13	35	14
	ReLU	55	27	31	11	34	13	35	14
ComBo		116	85	97	50	108	56	116	61
ComBoP		116	85	97	50	108	56	116	61
ComBoMV		46	21	23	8	26	9	29	10

Table 4.5 – $-\log_2$ of error rate for $p = 8$

In Table 4.5, we show that for any given set of parameters, the probability of error is almost identical between TOTA and Half-Torus, or slightly in favor of the latter. Meanwhile, ComBo and ComBoP outperform the other methods in every case by at least 30 orders of magnitude.

We notice that FDFB and ComBoMV do not behave in the same fashion as the other methods with respect to changes in parameters:

- They favorably compare to the others when the noise of the input ciphertext is small compared to V_r, as in set 1 where ComBoMV reaches an error rate of 2^{-46} while the Half-Torus method reaches an error rate of 2^{-34}. In these cases, the overhead of the noise created by the intermediary polynomial multiplication is absorbed by V_r.
- They unfavorably compare to the other methods when V_r is small compared to the noise of the input ciphertext, as in set 8 where ComBoMV reaches an error rate of 2^{-10} while the Half-Torus method reaches an error rate of 2^{-25}.

In addition, for FDFB, the specific values of the polynomial ($P_2 - P_1$ from Section4.4.2) also have to be taken into account when trying to gauge whether the

parameters are favorable or not towards FDFB use. Indeed, in simple cases such as the ReLU and Id functions, we can see a huge improvement (from 2^{-7} to 2^{-55} for the set 1) compared to the worst case approximation for FDFB.

4.6.3 Time Performance

The Half-Torus method is the fastest as it requires one BlindRotate. Then, TOTA is slightly faster than FDFB as it requires less KeySwitch operations. It is also on par with ComBoP as the parallelism overhead is negligible. As far as the ComBo method is concerned, the number of BlindRotate depends on the evaluated function. For a simple function such as the absolute value, its speed is identical to the Half-Torus method. Meanwhile, more complex functions need up to 4 bootstrappings. So, a sequential execution of ComBo becomes twice slower than TOTA and FDFB. Note however that these latter methods are intrinsically sequential. As such, they cannot outperform ComBoP.

As a bonus, we obtain a rule of thumb to get the computation time of each functional bootstrapping method. Indeed, multiplying the computation time of one bootstrapping with the number of consecutive BlindRotate gives accurate estimations of the result from Table 4.6. We remind that the computation time of one bootstrapping is almost equal to the time required to run to 1 BlindRotate plus 1 KeySwitch.

Set		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Half-Torus		135.0	126.1	101.4	94.6	97.4	84.5	85.5	72.0
TOTA		274.7	252.4	209.3	189.3	194.9	169.1	174.3	147.9
FDFB		287.0	268.1	220.5	203.2	207.4	181.2	182.8	157.8
ComBo	**abs**	136.5	126.0	104.9	94.6	97.5	84.5	87.0	74.2
	gen	551.5	503.6	417.7	378.0	389.6	337.5	341.4	296.5
ComBoP		273.6	258.8	211.1	200.1	205.3	182.1	183.3	153.5
ComBoMV		419.0	386.2	319.7	290.9	299.0	260.1	262.0	224.6

Table 4.6 – Computation time in ms. **gen** stands for a generic function.

Another way of showing ComBoP advantages is to compute the time performance of each method given their *own* optimized parameter set with respect to the same target error rate and plaintext space of size p. When doing so, we get the following example results with a target error rate of 2^{-32}:

- **p=4:** We achieve a speed up of x1.04 versus TOTA, x1.1 versus FDFB (ReLU) and x2 versus FDFB (worst case).

- **p=8:** We achieve a speed up of x1.09 versus TOTA, x1.12 versus FDFB (ReLU) and x4 versus FDFB (worst case).
- **p=16:** We achieve a speed up of x1.12 versus TOTA, x1.4 versus FDFB (ReLU) and x2 versus FDFB (worst case).

Besides, ComBo, ComBoP and ComBoMV are the only method allowing for parameters using $N = 1024$ when $p = 16$. This lead to ciphertexts twice smaller in this specific case, which is another important metric for FHE computations.

4.6.4 Wrapping-up: Time-Error trade-offs

We summarize the trade-offs between the computation time and the error rate for each method in Figure 4.2 and Figure 4.3. We separate the sets defined in Table 4.4 in order to have better readability of the figures.

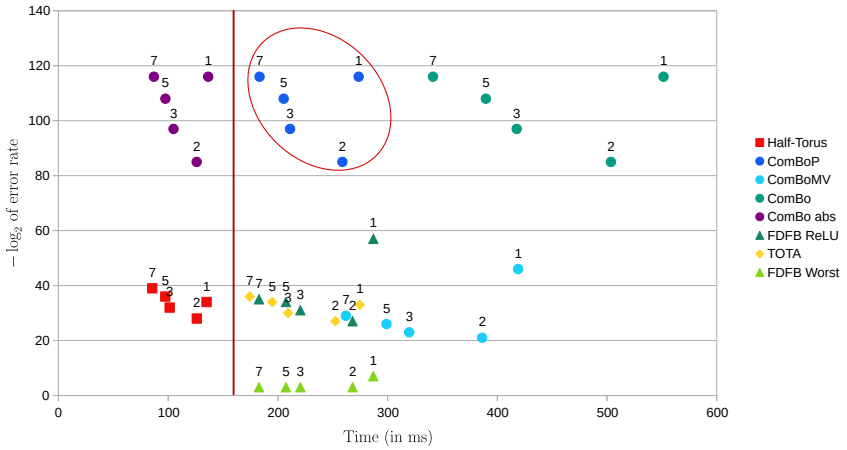


Figure 4.2 – Time-Error trade-off for parameters $1, 2, 3, 5$ and 7

For FDFB, we represent both the worst case and the ReLU, which is the best case among the functions we considered. For ComBo, ComBoMV and ComBoP methods, the best case is represented with the absolute value function and noted ComBo abs. The ComBo, ComBoMV and ComBoP points are all relative to a generic function following the pseudo even and pseudo odd decomposition from Section 4.4.4.

Fast operations will result in having points closer to the left. Meanwhile, a low error rate corresponds to points close to the upper parts of the graphs from Figures 4.2 and 4.3. With those two considerations in mind, we notice that the only methods on the left of the red line are the Half-Torus method and ComBo in the best case scenario. In this specific scenario, the ComBo method is the best in all regards. For functions requiring more bootstrappings, a compromise between speed and

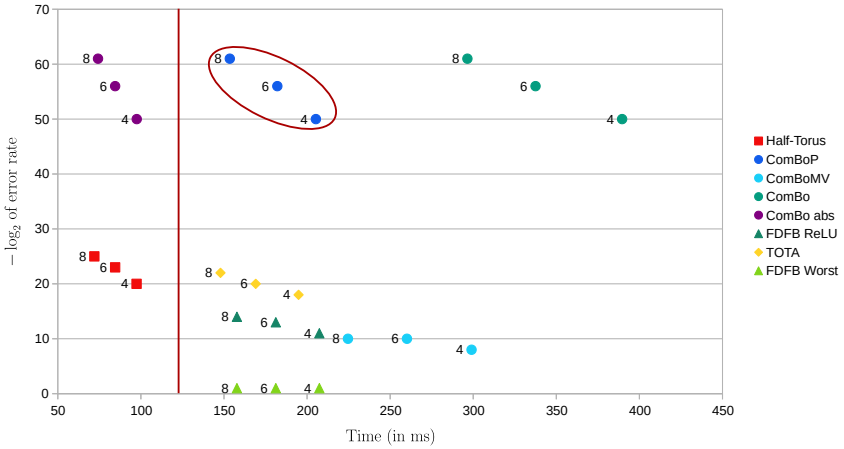


Figure 4.3 – Time-Error trade-off for parameters $4, 6$ and 8

error rate must be made. In the red circle lies the points relative to the ComBoP method. We can clearly see that it is both more accurate and faster than all the other methods except for the Half-Torus one. Thus, it is the best alternative to the Half-Torus method among the suggested functional bootstrapping.

4.7 Conclusion

Through the use of several bootstrappings and, most of the time, additional operations, every full domain method adds some output noise when compared to the partial domain method (Section 4.4.1). So the bottom line is: does a larger initial plaintext space make up for the added noise and computation time?
Table 4.5 and Table 4.6 confirm that the Yan et al., [100] (TOTA) method is both less accurate and twice as time-consuming than the partial domain method. Both Kluczniak and Schild's [57] (FDFB) and ComBoP methods provide a better accuracy than the partial domain method for well chosen parameters with varying additional computational costs.

Among the above full-domain methods, ComBoP achieves the best performance and accuracy. Furthermore, it outperforms the partial domain method in the following cases:

- The parameters of the cryptosystem are limited due to application constraints and the error rate of the Half-Torus is too large.
- Intermediate operations such as additions and multiplications push messages out of the Half-Torus space.

- Modular arithmetic is needed (which is impossible with the partial domain method).

When none of the above applies, however, the Half-Torus bootstrapping method still achieves better performances. This illustrates the fact, that there is no universal best method for functional bootstrapping and that one should carefully choose the most appropriate one depending on his or her application constraints. This paper's methodology and unified analysis gives a complete set of tools for making these choices.

ComBo (Section 4.4.4) has a smaller error rate than any other method available in the literature. In addition, as it allows to perform two bootstrappings in parallel, it may come without additional computational cost compared to the other full domain methods which are intrinsically serial. As such, ComBoP appears especially well adapted to benefit from the SIMD instruction sets available in modern processors. Furthermore, ComBo is particularly suited to homomorphic evaluation of functions such as ReLU, one of the key building-blocks for enabling advanced deep learning functions over encrypted data at larger scale.

Chapter 5

Chocobo: Creating Homomorphic Circuit Operating with Functional Bootstrapping in basis B

This chapter is a reproduction of our paper Chocobo co-written with Aymen Boudguiga and Renaud Sirdey and currently in reviewing process.

Abstract. The TFHE cryptosystem only supports small plaintext space, up to 5 bits with usual parameters. However, one solution to circumvent this limitation is to decompose input messages into a basis B over multiple ciphertexts. In this work, we introduce B-gates, an extension of logic gates to non binary bases, to compute base B logic circuit. The flexibility introduced by our approach improves the speed performance over previous approaches such as the so called tree-based method which requires an exponential amount of operations in the number of inputs. As an additional result, we introduce a keyswitching key specific to packing TLWE ciphertexts into TRLWE ciphertexts with redundancy, which is of interest in many functional bootstrapping scenarios.

keywords: FHE; TFHE; functional bootstrapping

5.1 Introduction

Homomorphic encryption schemes having an efficient bootstrapping, such as TFHE [28], can be tweaked to evaluate look-up tables within their bootstrapping procedure. Hence, rather than being just used for refreshing ciphertexts (i.e., reducing their noise level), the bootstrapping becomes *functional* [10] or *programmable* [30]

by allowing the evaluation of arbitrary functions as a bonus. These capabilities result in promising new approaches for improving the overall performances of homomorphic calculations, making the FHE "API" better suited to the evaluation of mathematical operators which are difficult to express as low complexity arithmetic circuits.

TFHE made a first breakthrough by proposing an efficient bootstrapping for homomorphic gate computation. Then, Bourse et al., [9] used the same bootstrapping algorithm for extracting the (encrypted) sign of an encrypted input. It was later used by Izabachene et al., [53] to evaluate a Hopfield network in the encrypted domain. Boura et al., [8] showed in 2019 that TFHE bootstrapping naturally allows to encode function evaluation via their representation as look-up tables (LUTs). Recently, different approaches have been investigated for functional bootstrapping improvement. Guimarães et al., [48] extended the ideas in Bourse et al., [10] to support the evaluation of functions over multiple inputs via LUTs. In this work, we build on the work of Guimarães et al., and extend it to provide a new evaluation technique for *arbitrary* circuits.

Related works – After Bourse et al., [9] described how the functional bootstrapping of TFHE computes a sign function, researchers investigated the implementation of LUT(f) for any function f with domain either half of the torus [30] or the entire torus [57, 100, 31, 35]. Encoding plaintext values only on $[0, \frac{1}{2}[$ (i.e., half of the torus) avoids the restriction of managing negacyclic functions during the bootstrapping. However, it reduces the size of the plaintext space as it is encoded on a smaller portion of the torus $\mathbb{T}$. Meanwhile, other methods [57, 100, 31, 35] support $\mathbb{T}$ as a plaintext space at the cost of adding more bootstrappings. Subsequently, Guimarães et al., have proposed the tree-based and chaining-based methods to evaluate functions of multiple ciphertexts by means of several functional bootstrapping. However, the efficiency of their tree-based method is limited by its exponential complexity relatively to the number of inputs. Meanwhile, the chaining-method is only well suited for computing carry-like functions [48].

Contributions – In this work, we first revisit the noise variances and success probabilities of the tree-based and chain-based method of Guimarães et al. We also describe and compare in detail multiple methods to compute B-gates which are logic gates extended to bases B greater than 2, and serve as building blocks for the computation of base B logic circuits. We show that the evaluation of circuits with our novel building block favorably compares to the tree-based method in terms of time performance. As an application, we show that our technique can be implemented efficiently in practice by taking as example a sorting algorithm.

Paper organization – The remainder of this paper is organized as follows. Sec-

tion 5.2 reviews TFHE building blocks. Section 5.5 describes techniques from the literature to evaluate look up tables with multiple inputs. Section 5.6 describes our method to compute multi-inputs functions using functional bootstrapping. Finally, Section 5.8 highlights the benefits of our method through the evaluation of a sorting algorithm.

5.2 Background

5.2.1 Notations

We refer to the real torus by $\mathbb{T} = \mathbb{R}/\mathbb{Z}$. $\mathbb{T}$ is the additive group of real numbers modulo 1 ($\mathbb{R} \mod [1]$) and it is a $\mathbb{Z}$-module. $\mathbb{T}_N[X]$ denotes the $\mathbb{Z}$-module $\mathbb{R}[X]/(X^N+1) \mod [1]$ of torus polynomials, where N is a power of 2. $\mathcal{R}$ is the ring $\mathbb{Z}[X]/(X^N+1)$ and its subring of polynomials with binary coefficients is $\mathbb{B}_N[X] = \mathbb{B}[X]/(X^N+1)$ ($\mathbb{B} = \{0,1\}$). We denote by $\mathbb{Z}_n$ the ring $\mathbb{Z}/n\mathbb{Z}$. Finally, we denote respectively by $[x]_{\mathbb{T}}$, $[x]_{\mathbb{T}_N[X]}$ and $[x]_{\mathcal{R}}$ the encryption of x over $\mathbb{T}$, $\mathbb{T}_N[X]$ or $\mathcal{R}$.

We refer to vectors by bold letters. $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ is the inner product of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. We denote matrices by capital letters, and the set of matrices with m rows and n columns with entries sampled in $\mathbb{K}$ by $\mathcal{M}_{m,n}(\mathbb{K})$. $x \xleftarrow{\$} \mathbb{K}$ denotes sampling x uniformly from $\mathbb{K}$, while $x \xleftarrow{\mathcal{N}(\mu,\sigma^2)} \mathbb{K}$ refers to sampling x from $\mathbb{K}$ following a Gaussian distribution of mean μ and variance σ^2.

Given $x \xleftarrow{\mathcal{N}(\mu,\sigma^2)} \mathbb{R}$, the probability $P(a \leq x \leq b)$ is equal to $\frac{1}{2}(\mathrm{erf}(\frac{b-\mu}{\sqrt{2}\sigma}) - \mathrm{erf}(\frac{a-\mu}{\sqrt{2}\sigma}))$, where erf is the Gauss error function: $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}$. The same result apply to $x \xleftarrow{\mathcal{N}(0,\sigma^2)} \mathbb{T}$ as long as the distribution is concentrated as described in [28].

5.2.2 TFHE Structures

The TFHE encryption scheme was proposed in 2016 [25] and updated in [28]. It introduces the TLWE problem as an adaptation of the LWE problem to $\mathbb{T}$. TFHE relies on three structures to encrypt plaintexts defined over $\mathbb{T}$, $\mathbb{T}_N[X]$ or $\mathcal{R}$:

- **TLWE Sample:** $(\boldsymbol{a}, b)$ is a valid TLWE sample if $\boldsymbol{a} \xleftarrow{\$} \mathbb{T}^n$ and $b \in \mathbb{T}$ verifies $b = \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e$, where $\boldsymbol{s} \xleftarrow{\$} \mathbb{B}^n$ is the secret key, and $e \xleftarrow{\mathcal{N}(0,\sigma^2)} \mathbb{T}$.
- **TRLWE Sample:** a pair $(\boldsymbol{a}, b) \in \mathbb{T}_N[X]^k \times \mathbb{T}_N[X]$ is a valid TRLWE sample if $\boldsymbol{a} \xleftarrow{\$} \mathbb{T}_N[X]^k$, and $b = \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e$, where $\boldsymbol{s} \xleftarrow{\$} \mathbb{B}_N[X]^k$ is a TRLWE secret

key and $e \xleftarrow{\mathcal{N}(0,\sigma^2)} \mathbb{T}_N[X]$ is a noise polynomial.

Let $\mathcal{M} \subset \mathbb{T}_N[X]$ (or $\mathcal{M} \subset \mathbb{T}$) be the discrete message space[1]. To encrypt a message $m \in \mathcal{M}$, we add $(\mathbf{0}, m)$ to a fresh T(R)LWE sample. In the following, we refer to an encryption of m with the secret key $\boldsymbol{s}$ as a T(R)LWE ciphertext noted $\boldsymbol{c} \in \text{T(R)LWE}_{\boldsymbol{s}}(m)$.

To decrypt a sample $\boldsymbol{c} \in \text{T(R)LWE}_{\boldsymbol{s}}(m)$, we compute its *phase* $\phi(\boldsymbol{c}) = b - \langle \boldsymbol{a}, \boldsymbol{s} \rangle = m+e$. Then, we round to it to the nearest element of $\mathcal{M}$. Therefore, if the error e was chosen to be small enough while ensuring security, the decryption will be accurate.

- **TRGSW Sample:** a TRGSW sample is a vector of TRLWE samples. To encrypt a message $m \in \mathcal{R}$, we add $m \cdot H$ to a TRGSW sample, where H is a gadget matrix[2] using an integer B_g as a basis for its decomposition. Chilotti et al., [28] defines an external product between a TRGSW sample A encrypting $m_a \in \mathcal{R}$ and a TRLWE sample $\boldsymbol{b}$ encrypting $m_b \in \mathbb{T}_N[X]$. This external product consists in multiplying A by the approximate decomposition of $\boldsymbol{b}$ with respect to H (Definition 3.12 in [28]). It yields an encryption of $m_a \cdot m_b$ i.e., a TRLWE sample $\boldsymbol{c} \in \text{TRLWE}_{\boldsymbol{s}}(m_a \cdot m_b)$. Otherwise, the external product allows also to compute a controlled MUX gate (CMUX) where the selector is $C_b \in \text{TRGSW}_{\boldsymbol{s}}(b)$, $b \in \{0, 1\}$, and the inputs are $\boldsymbol{c}_0 \in \text{TRLWE}_{\boldsymbol{s}}(m_0)$ and $\boldsymbol{c}_1 \in \text{TRLWE}_{\boldsymbol{s}}(m_1)$.

5.2.3 TFHE Bootstrapping

TFHE bootstrapping relies mainly on three building blocks:

- **Blind Rotate:** rotates a plaintext polynomial encrypted as $\boldsymbol{c} \in \text{TRLWE}_{\boldsymbol{k}}(m)$ by a position encrypted as $\boldsymbol{c}_p \in \text{TLWE}_{\boldsymbol{s}}(p)$. It takes as inputs: the TRLWE ciphertext $\boldsymbol{c} \in \text{TRLWE}_{\boldsymbol{k}}(m)$, a rescaled and rounded vector of $\boldsymbol{c}_p$ represented by $(a_1, \ldots, a_n, a_{n+1} = b)$ where $\forall i$, $a_i \in \mathbb{Z}_{2N}$, and n TRGSW ciphertexts encrypting $(s_1, \ldots, s_n)$ where $\forall i$, $s_i \in \mathbb{B}$. It returns a TRLWE ciphertext $\boldsymbol{c'} \in \text{TRLWE}_{\boldsymbol{k}}(X^{\langle \boldsymbol{a}, \boldsymbol{s} \rangle - b} \cdot m)$. In this paper, we will refer to this algorithm by $\mathsf{BlindRotate}$. With respect to the independence heuristic[3] stated in [28], the

[1] In practice, we discretize the Torus with respect to our plaintext modulus. For example, if we want to encrypt $m \in \mathbb{Z}_4 = \{0, 1, 2, 3\}$, we encode it in $\mathbb{T}$ as one of the following value $\{0, 0.25, 0.5, 0.75\}$.

[2] Refer to Definition 3.6 and Lemma 3.7 in TFHE paper [28] for more information about the gadget matrix H.

[3] The independence heuristic ensures that all the coefficients of the errors of TLWE, TRLWE or TRGSW samples are independent and concentrated. More precisely, they are σ-subgaussian where σ is the square-root of their variance.

variance $\mathcal{V}_{BR}$ of the resulting noise after a BlindRotate satisfies the formula:

$$\mathcal{V}_{BR} < V_c + \mathcal{E}_{BR}, \text{ where } \mathcal{E}_{BR} = n\left((k+1)\ell N\left(\frac{B_g}{2}\right)^2 \vartheta_{BK} + \frac{(1+kN)}{12 \cdot B_g^{2l}}\right) \tag{5.1}$$

V_c is the variance of the noise of the input ciphertext $\boldsymbol{c}$, and ϑ_{BK} is the variance of the error of the bootstrapping key. l and Bg are the parameters defining the gadget matrix as in [28]. Note that the noise of the BlindRotate is independent from the noise of the encrypted position $\boldsymbol{c}_p$.

- **TLWE Sample Extract:** takes as inputs both a position $p \in [\![0, N[\![$ and a ciphertext $\boldsymbol{c} \in \text{TRLWE}_{\boldsymbol{k}}(m)$, and returns a TLWE ciphertext $\boldsymbol{c'} \in \text{TLWE}_{\boldsymbol{k}}(m_p)$ where m_p is the p^{th} coefficient of the polynomial m. In this paper, we will refer to this algorithm by SampleExtract. This algorithm does not add any noise to the ciphertext.

- **Public Functional Keyswitching:** transforms a set of p ciphertexts $\boldsymbol{c}_i \in \text{TLWE}_{\boldsymbol{k}}(m_i)$ into the resulting ciphertext $\boldsymbol{c'} \in \text{T(R)LWE}_{\boldsymbol{s}}(f(m_1, \ldots, m_p))$, where $f()$ is a public linear morphism from $\mathbb{T}^p$ to $\mathbb{T}_N[X]$. Note that $N = 1$ when keyswitching to a TLWE ciphertext. This algorithm requires 2 parameters: the decomposition basis B_{KS} and the precision of the decomposition t. In this paper, we will refer to this algorithm by KeySwitch. As stated in [28], the variance $\mathcal{V}_{KS}$ of the resulting noise after a KeySwitch with $B_{KS} = 2$ follows the formula[4]:

$$\mathcal{V}_{KS} < R^2 \cdot V_c + n\left(tN\vartheta_{KS} + 2^{-2(t+1)}\right)$$

where V_c is the variance of the noise of the input ciphertext c, R is the Lipschitz constant of f and ϑ_{KS} the variance of the error of the keyswitching key. In this paper and in most cases, $R = 1$.

TFHE specifies a gate bootstrapping to reduce the noise level of a TLWE sample that encrypts the result of a boolean gate evaluation on two ciphertexts, each of them encrypting a binary input. TFHE gate bootstrapping steps are summarized in Algorithm 10. The step 1 consists in selecting a value $\hat{m} \in \mathbb{T}$ which will serve later for setting the coefficients of the test polynomial *testv* (in step 3). The step 2 rescales the components of the input ciphertext $\boldsymbol{c}$ as elements of $\mathbb{Z}_{2N}$. The step 3 defines the test polynomial *testv*. Note that for all $p \in [\![0, 2N[\![$, the constant term of $testv \cdot X^p$ is $\hat{m}$ if $p \in]\!]\frac{N}{2}, \frac{3N}{2}[\![$ and $-\hat{m}$ otherwise. The step 4 returns an accumulator $ACC \in \text{TRLWE}_{\boldsymbol{s'}}(testv \cdot X^{\langle \bar{\boldsymbol{a}}, \boldsymbol{s} \rangle - \bar{b}})$. Indeed, the constant term

[4]Note that as of now, the formula from [28] has a discrepancy between Theorem4.1 and its proof. The formula from the proof should be followed.

of ACC is $-\hat{m}$ if $\boldsymbol{c}$ encrypts 0, or $\hat{m}$ if $\boldsymbol{c}$ encrypts 1 as long as the noise of the ciphertext is small enough[5]. Then, step 5 creates a new ciphertext $\overline{\boldsymbol{c}}$ by extracting the constant term of ACC and adding to it $(\mathbf{0}, \hat{m})$. That is, $\overline{\boldsymbol{c}}$ either encrypts 0 if $\boldsymbol{c}$ encrypts 0, or m if $\boldsymbol{c}$ encrypts 1 (By choosing $m = \frac{1}{2}$, we get a fresh encryption of 1).

Since a bootstrapping operation is a BlindRotate over a noiseless TRLWE followed by a Keyswitch, the bootstrapping noise ($\mathcal{V}_{BS}$) satisfies:

$$\mathcal{V}_{BS} < \mathcal{E}_{BR} + n\left(t\vartheta_{KS} + 2^{-2(t+1)}\right)$$

Algorithm 10 TFHE gate bootstrapping [28]

Input: a constant $m \in \mathbb{T}$, a TLWE sample $\boldsymbol{c} = (\boldsymbol{a}, b) \in \mathrm{TLWE}_{\boldsymbol{s}}(x \cdot \frac{1}{2})$ with $x \in \mathbb{B}$, a bootstrapping key $BK_{\boldsymbol{s}\to\boldsymbol{s'}} = (BK_i \in \mathrm{TRGSW}_{S'}(s_i))_{i\in[\![1,n]\!]}$ where S' is the TRLWE interpretation of a secret key $\boldsymbol{s'}$

Output: a TLWE sample $\overline{\boldsymbol{c}} \in \mathrm{TLWE}_{\boldsymbol{s}}(x.m)$

1: Let $\hat{m} = \frac{1}{2}m \in \mathbb{T}$ (pick one of the two possible values)
2: Let $\bar{b} = \lfloor 2Nb \rceil$ and $\bar{a}_i = \lfloor 2Na_i \rceil \in \mathbb{Z}, \forall i \in [\![1, n]\!]$
3: Let $testv := (1 + X + \cdots + X^{N-1}) \cdot X^{\frac{N}{2}} \cdot \hat{m} \in \mathbb{T}_N[X]$
4: $ACC \leftarrow \mathsf{BlindRotate}((\mathbf{0}, testv), (\bar{a}_1, \ldots, \bar{a}_n, \bar{b}), (BK_1, \ldots, BK_n))$
5: $\overline{\boldsymbol{c}} = (\mathbf{0}, \hat{m}) + \mathsf{SampleExtract}(ACC)$
6: return $\mathsf{KeySwitch}_{\boldsymbol{s'}\to\boldsymbol{s}}(\overline{\boldsymbol{c}})$

5.2.4 TFHE Functional Bootstrapping

Functional bootstrapping [30, 57, 100, 31, 35] refers to TFHE's ability of evaluating a Look-Up Table (LUT) of any single input function during the bootstrapping. In particular, TFHE is well-suited for negacyclic function[6], as the plaintext space for TFHE is $\mathbb{T}$, where $[0, \frac{1}{2}[$ corresponds to positive values and $[\frac{1}{2}, 1[$ to negative ones, and the bootstrapping step 2 of the Algorithm 10 encodes elements from $\mathbb{T}$ into powers of X modulo $(X^N + 1)$, and $X^{\alpha+N} \equiv -X^{\alpha} \mod [X^N + 1]$.

In section 5.5, we will discuss methods for increasing the plaintext precision during a functional bootstrapping. We will focus on Guimarães et al., [48] ideas for combining several bootstrappings with many digits as input. These digits come from the decomposition of a plaintext in a basis B.

[5]Further details on the proper bound of the noise are given in Section 5.4.
[6]Negacyclic functions are antiperiodic functions over $\mathbb{T}$ with period $\frac{1}{2}$, i.e., $f(x) = -f(x + \frac{1}{2})$.

5.3 KeySwitch with decomposition basis greater than 2

In this section, we detail the KeySwitch operation using any decomposition basis B_{KS}, usually a power of 2.

We give in Algorithm 11 an adaptation of the KeySwitch from [28] for a decomposition basis greater than 2. This algorithm leads to the following noise formula:

$$\mathcal{V}_{KS} < R^2 \cdot V_c + \mathcal{E}_{KS}^{n,N} \text{ where } \mathcal{E}_{KS}^{n,N} = n\left(tN\vartheta_{KS} \cdot \left(\frac{B_{KS}}{2}\right)^2 + \frac{B_{KS}^{-2t}}{12}\right) \tag{5.2}$$

We call this KeySwitch *packing* when $f(m^{(1)}, ..., m^{(p)}) = \sum_{i=1}^{p} m^{(i)} X^{i-1}$ and *packing with redundancy* when $f(m^{(1)}, ..., m^{(p)}) = \sum_{i=1}^{p}\left(m^{(i)} \sum_{j=0}^{r-1} X^{r\cdot(i-1)+j}\right)$ where r is the redundancy term satisfying $r \cdot p \leq N$.

Algorithm 11 general TFHE KeySwitch

Input: p TLWE ciphertexts $\boldsymbol{c}^{(i)} = (\boldsymbol{a}^{(i)}, b^{(i)}) \in \text{TLWE}_s(m^{(i)})$ for $i \in [\![1, p]\!]$, a public R-Lipschitz morphism $f : \mathbb{T}^p \to \mathbb{T}_N[X]$, and $\text{KS}_{i,j} \in \text{T(R)LWE}_K(\frac{s_i}{B_{KS}^j})$ for $i \in [\![1, n]\!]$, $j \in [\![1, t]\!]$.

Output: A T(R)LWE sample $\boldsymbol{c} \in \text{T(R)LWE}_K(f(m^{(1)}, ..., m^{(p)}))$.

1: **for** $i \in [\![1, n]\!]$ **do**

2: Let $a_i = f(a_i^{(1)}, ..., a_i^{(p)})$

3: Let $\tilde{a}_i = \frac{\lceil B_{KS}^t \cdot a_i \rfloor}{B_{KS}^t}$

4: Let $\tilde{a}_{i,j} \in \mathbb{Z}_N[X]$ with coefficients in $[\![-\frac{B_{KS}}{2}, \frac{B_{KS}}{2} - 1]\!]$ so that $\sum_{j=1}^{t} \tilde{a}_{i,j} \cdot B_{KS}^{-j} = \tilde{a}_i$

return $(0, f(b^{(1)}, ..., b^{(p)})) - \sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j} \cdot \text{KS}_{i,j}$

When applying a TLWE to TLWE KeySwitch the analysis from [48] regarding the variance of the rounding part of the algorithm applies. Since $N = 1$ in this case, the noise formula then drops to:

$$\mathcal{V}_{KS} < R^2 \cdot V_c + \mathcal{E}_{KS}^{n,1}$$

Algorithm 12 further improves the noise bound for KeySwitch between TLWE ciphertexts by introducing a larger key. More specifically, the size of the key grows linearly with the chosen decomposition basis. The noise formula for this algorithm is:

$$\mathcal{V}_{KS} < R^2 \cdot V_c + n\left(t\vartheta_{KS} + \frac{B_{KS}^{-2t}}{12}\right)$$

Algorithm 12 TFHE KeySwitch between TLWE ciphertexts

Input: p TLWE ciphertexts $\boldsymbol{c}^{(i)} = (\boldsymbol{a}^{(i)}, b^{(i)}) \in \mathrm{TLWE}_s(m^{(i)})$ for $i \in [\![1, p]\!]$, a public R-Lipschitz morphism $f : \mathbb{T}^p \to \mathbb{T}$, and $\mathrm{KS}_{i,j,k} \in \mathrm{TLWE}_K(k \cdot \frac{s_i}{B_{KS}^j})$ for $i \in [\![1, n]\!]$, $j \in [\![1, t]\!]$, $k \in [\![0, B_{KS} - 1]\!]$.

Output: A TLWE sample $\boldsymbol{c} \in \mathrm{TLWE}_K(f(m^{(1)}, ..., m^{(p)}))$.

1: **for** $i \in [\![1, n]\!]$ **do**

2: Let $a_i = f(a_i^{(1)}, ..., a_i^{(p)})$

3: Let $\tilde{a}_i = \frac{\lceil B_{KS}^t \cdot a_i \rfloor}{B_{KS}^t}$

4: Let $\tilde{a}_{i,j} \in [\![0, B_{KS} - 1]\!]$ so that $\sum_{j=1}^{t} \tilde{a}_{i,j} \cdot B_{KS}^{-j} = \tilde{a}_i$

return $(0, f(b^{(1)}, ..., b^{(p)})) - \sum_{i=1}^{n}\sum_{j=1}^{t} \mathrm{KS}_{i,j,\tilde{a}_{i,j}}$

Finally, we show in Algorithm 13 how to compute a packing with redundancy with less noise compared to Algorithm 11 thanks to a specific keyswitch key. Algorithm 13 improves on the base aware Keyswitch of [48] by avoiding the multiplicative increase in size of the key and correcting the algorithm. This algorithm leads to the following noise formula:

$$\mathcal{V}_{KS} < V_c + \mathcal{E}_{KS}^{n,p} \tag{5.3}$$

5.4 Time complexity, noise variance and success probability

5.4.1 Time complexity

The algorithms we present in this paper use primarily BlindRotate and Keyswitch operations. Those 2 operations are the most time consuming FHE operations.

Algorithm 13 TFHE packing with redundancy

Input: p TLWE ciphertexts $\boldsymbol{c}^{(i)} = (\boldsymbol{a}^{(i)}, b^{(i)}) \in \mathrm{TLWE}_s(m^{(i)})$ for $i \in [\![1, p]\!]$, and $\mathrm{KS}_{i,j} \in \mathrm{TRLWE}_K(\frac{s_i}{B_{KS}^j} \sum_{k=0}^{r-1} X^k)$ for $i \in [\![1, n]\!]$, $j \in [\![1, t]\!]$.

Output: A TRLWE sample $\boldsymbol{c} \in \mathrm{TRLWE}_K\left(\sum_{i=1}^{p}\left(m^{(i)} \sum_{j=0}^{r-1} X^{r\cdot(i-1)+j}\right)\right)$.

1: **for** $i \in [\![1, n]\!]$ **do**

2: Let $a_i = \sum_{j=1}^{p} a_i^{(j)} X^{r\cdot(j-1)}$

3: Let $\tilde{a}_i = \frac{\lceil B_{KS}^t \cdot a_i \rfloor}{B_{KS}^t}$

4: Let $\tilde{a}_{i,j} \in \mathbb{Z}_N[X]$ with coefficients in $[\![-\frac{B_{KS}}{2}, \frac{B_{KS}}{2} - 1]\!]$ so that $\sum_{j=1}^{t} \tilde{a}_{i,j} \cdot B_{KS}^{-j} = \tilde{a}_i$

return $\left(0, \sum_{i=1}^{p} b^{(i)} \sum_{j=0}^{r-1} X^{r\cdot(i-1)+j}\right) - \sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j} \cdot \mathrm{KS}_{i,j}$

Note that a BlindRotate is $2 \cdot 10^4$ times slower than additions and 10^3 times slower than polynomial multiplications with default parameters in TFHElib[7].

The following formula gives a good approximation of the time needed for each algorithm : $n \cdot t_{\mathrm{BR}} + m \cdot t_{\mathrm{KS}} + p \cdot t_{\mathrm{KSR}}$, where n is the number of BlindRotate, m is the number of KeySwitch between TLWE ciphertexts, p is the number of KeySwitch from TLWE to TRLWE ciphertexts and the indexed t coefficients correspond to the time computation of each operation.

5.4.2 Noise variance

The noise variances resulting from homomorphic computations with bootstrappings are calculated from the noise variances of the input ciphertexts, and the noise bounds of BlindRotate and KeySwitch (Equations 5.1 and 5.2). We summarize in Table 5.1 the noise variances for basic operations such as the addition of two ciphertexts or the multiplication of a ciphertext by a plaintext. These formulas can be used as building-blocks to compute the resulting noise variance of operations in the remainder of this paper.

[7] `https://tfhe.github.io/tfhe/`

Operation	Variance
$\boldsymbol{c}_i + \boldsymbol{c}_j$	$V_i + V_j$
$\boldsymbol{C}_i + \boldsymbol{C}_j$	$V_i + V_j$
$P \cdot \boldsymbol{C}_i$	$\|\|P\|\|_2^2 \cdot V_i$
$\mathsf{Keyswitch}(\boldsymbol{c}_i)$	$V_i + \mathcal{E}_{KS}^{n,N}$
$\mathsf{BlindRotate}(\boldsymbol{C}_i)$	$V_i + \mathcal{E}_{BR}$
$\mathsf{Bootstrap}(\boldsymbol{c}_i)$	$\mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1}$

Table 5.1 – Obtained noise variances when applying basic operations to independent inputs: $\boldsymbol{c}_i$ is a TLWE ciphertext of variance V_i, $\boldsymbol{C}_i$ is a TRLWE ciphertext of variance V_i and P is a plaintext polynomial.

5.4.3 Success probability

The success probability of a BlindRotate is expressed with the Gauss error function. It depends on the noise of the input ciphertext, the size of the plaintext space and its encoding over the torus [35, 48]. Indeed, Clet et al., [33] showed that for a small plaintext space, functional bootstrapping is more efficient when the plaintexts are encoded over half of the torus.

As in the upcoming section, we consider the decomposition of plaintexts within a small basis B, we only remind the formula for the probability of success of a BlindRotate when the plaintext space is encoded over half of the torus:

$$\mathrm{erf}\left(\frac{1}{4B\sqrt{2(V_{\boldsymbol{c}} + V_r)}}\right)$$

where erf is the Gauss error function, $V_{\boldsymbol{c}}$ is the noise variance of the input ciphertext, and $V_r = \frac{n+1}{48N^2}$ is the rounding variance (introduced by the step 2 of Algorithm 10). From now on, we use the notation:

$$\mathcal{P}_B(V_{\boldsymbol{c}}) = \mathrm{erf}(\frac{1}{4B\sqrt{2(V_{\boldsymbol{c}} + V_r)}}) \tag{5.4}$$

Let's consider an algorithm requiring to compute n BlindRotate on independent ciphertexts $(\boldsymbol{c}_i)_{i\in[\![0,m-1]\!]}$. Each $\boldsymbol{c}_i$ is used, at least once, as input to a BlindRotate. Then the probability of success of the algorithm is the probability that each BlindRotate succeeds. It is equal to:

$$\prod_{i=0}^{m-1} \mathcal{P}_B(V_{\boldsymbol{c}_i})$$

We note $\mathcal{F}_B = 1 - \mathcal{P}_B$ so that $\mathcal{F}$ represents the error rate of a BlindRotate. Then, if each $\mathcal{F}_B(V_{\boldsymbol{c}_i})$ is small enough, we get the approximation:

$$1 - \prod_{i=0}^{m-1} \mathcal{P}_B(V_{\boldsymbol{c}_i}) \simeq \sum_{i=0}^{m-1} \mathcal{F}_B(V_{\boldsymbol{c}_i}) \tag{5.5}$$

Let's analyse how precise this approximation can be.
We get thanks to Boole's inequality that

$$1 - \prod_{i=0}^{m-1} \mathcal{P}_B(V_{\boldsymbol{c}_i}) \leq \sum_{i=0}^{m-1} \mathcal{F}_B(V_{\boldsymbol{c}_i}) \tag{5.6}$$

Besides, if we note $\alpha = \sum_{i=0}^{m-1} \mathcal{F}_B(V_{\boldsymbol{c}_i})$ ($\ll 1$ since each $\mathcal{F}_B(V_{\boldsymbol{c}_i})$ is small), we get

$$\begin{aligned} \prod_{i=0}^{m-1} \mathcal{P}_B(V_{\boldsymbol{c}_i}) &= \prod_{i=0}^{m-1} (1 - \mathcal{F}_B(V_{\boldsymbol{c}_i})) \\ &\leq \left(1 - \tfrac{\alpha}{m}\right)^m = 1 - \alpha + \tfrac{m-1}{2m}\alpha^2 + \sum_{k=3}^{m} \binom{m}{k} (-\frac{\alpha}{m})^k \end{aligned}$$

Note that $|\sum_{k=3}^{m} \binom{m}{k} (-\frac{\alpha}{m})^k| \leq \sum_{k=3}^{m} \frac{\alpha^k}{k!} \leq \alpha^3 \cdot (e - 2,5) \leq 0.22 \cdot \alpha^3$.
Thus for any $\epsilon > 0$ we can ensure that

$$\begin{aligned} 1 - \prod_{i=0}^{m-1} \mathcal{P}_B(V_{\boldsymbol{c}_i}) &\geq \sum_{i=0}^{m-1} \mathcal{F}_B(V_{\boldsymbol{c}_i})(1 - \alpha(\frac{m-1}{2m} + 0.22\alpha)) \\ &\geq \sum_{i=0}^{m-1} \mathcal{F}_B(V_{\boldsymbol{c}_i})(1 - \epsilon) \end{aligned} \tag{5.7}$$

as long as $\alpha(\frac{m-1}{2m} + 0.22\alpha) \leq \epsilon$. To satisfy this inequality, it is enough that $0.72\alpha \leq \min(\epsilon, 1)$.

We combine Equations 5.6 and 5.7 to get

$$\sum_{i=0}^{m-1} \mathcal{F}_B(V_{\boldsymbol{c}_i})(1 - \epsilon) \leq 1 - \prod_{i=0}^{m-1} \mathcal{P}_B(V_{\boldsymbol{c}_i}) \leq \sum_{i=0}^{m-1} \mathcal{F}_B(V_{\boldsymbol{c}_i})$$

which shows that the approximation is tight when the error rate is low.

We use the same independence heuristic as in [25] to avoid overly complex formula when the correlation between ciphertexts becomes too intricate. From now on, we use Boole's inequality given in Equation 5.6 as a tight approximation of the error rate of our computations under this assumption.

5.5 LUTs with Multiple Encrypted Inputs

The aforementioned functional bootstrapping methods ([30, 57, 100, 31, 35]) are univariate and have a limited plaintext precision. They evaluate look-up tables with a size bounded by the degree N of the used cyclotomic polynomial. The size of the plaintext space gets even smaller when taking noise into account [35]. In addition, these methods are not suited for computing a LUT for a multivariate function f with two or more encrypted inputs. In order to overcome these issues, Guimarães et al., [48] proposed two methods for homomorphic computation with digits: a tree-based approach and a chaining approach. In this section, we discuss these two methods and give bounds for their noise variances and error rates.

5.5.1 Tree-based Method

We consider d TLWE ciphertexts $(\boldsymbol{c}_0, \dots, \boldsymbol{c}_{d-1})$ encrypting $(m_0, \dots, m_{d-1}) \in \mathbb{Z}_B^d$ over half of the torus for some $B \in \mathbb{N}$. That is, each ciphertext $\boldsymbol{c}_i$ corresponds to an encryption of $m_i \in [\![0, B-1]\!]$. $(m_0, \dots, m_{d-1})$ can represent a message decomposed in a basis B, via the bijection:

$$g_d : \begin{array}{ccc} [\![0, B-1]\!]^d & \to & [\![0, B^d-1]\!] \\ (m_0, \dots, m_{d-1}) & \mapsto & \sum_{i=0}^{d-1} m_i \cdot B^i \end{array} \tag{5.8}$$

or simply d uncorrelated messages.

In the following, we describe a tree-like structure to build a LUT for the function $f : [\![0, B-1]\!]^d \to [\![0, B-1]\!]$. An example is described in Figure 5.1 of a tree of depth $d = 2$

First, we encode the LUT for f in B^{d-1} TRLWE ciphertexts. Each ciphertext encrypts a polynomial P_i where:

$$P_i(X) = \sum_{j=0}^{B-1} \sum_{k=0}^{\frac{N}{B}-1} f \circ g_d^{-1}(j \cdot B^{d-1} + i) \cdot X^{j \cdot \frac{N}{B} + k}, \quad i \in [\![0, B^{d-1} - 1]\!]$$

Then, we apply the BlindRotate algorithm to $\boldsymbol{c}_{d-1}$ and $\mathsf{TRLWE}(P_i), \forall i \in [\![0, B^{d-1} - 1]\!]$. That is, we rotate each $\mathsf{TRLWE}(P_i)$ by the encrypted position in $\boldsymbol{c}_{d-1}$. Finally, we run the SampleExtract algorithm on each of the rotated $\mathsf{TRLWE}(P_i)$. We

end up with B^{d-1} TLWE ciphertexts, each encrypting $f \circ g_d^{-1}(m_{d-1} \cdot B^{d-1} + i)$ for $i \in [\![0, B^{d-1}-1]\!]$. Then, we apply B^{d-2} KeySwitch to pack these B^{d-1} TLWE ciphertexts into B^{d-2} TRLWE ciphertexts, that correspond to the LUT of h where:

$$h : \begin{array}{ccc} [\![0, B-1]\!]^{d-1} & \to & [\![0, B-1]\!] \\ (a_0, \ldots, a_{d-2}) & \mapsto & f(a_0, \ldots, a_{d-2}, m_{d-1}) \end{array}$$

We iterate this operation until getting only one TLWE ciphertext encrypting $f(m_0, \ldots, m_{d-1})$, at the cost of running $\sum_{i=0}^{d-1} B^i = \frac{B^d-1}{B-1}$ BlindRotate, $\sum_{i=0}^{d-2} B^i = \frac{B^{d-1}-1}{B-1}$ KeySwitch from TLWE ciphertexts to TRLWE ciphertext and one KeySwitch between TLWE to go back to the initial parameters. Intermediary KeySwitch between TLWE can be performed before keyswitching to TRLWE ciphertexts which reduce the overall noise as long as $\mathcal{E}_{KS}^{N,1} + \mathcal{E}_{KS}^{n,B} \leq \mathcal{E}_{KS}^{N,B}$ at the cost of $\frac{B^d-B}{B-1}$ additional KeySwitch.

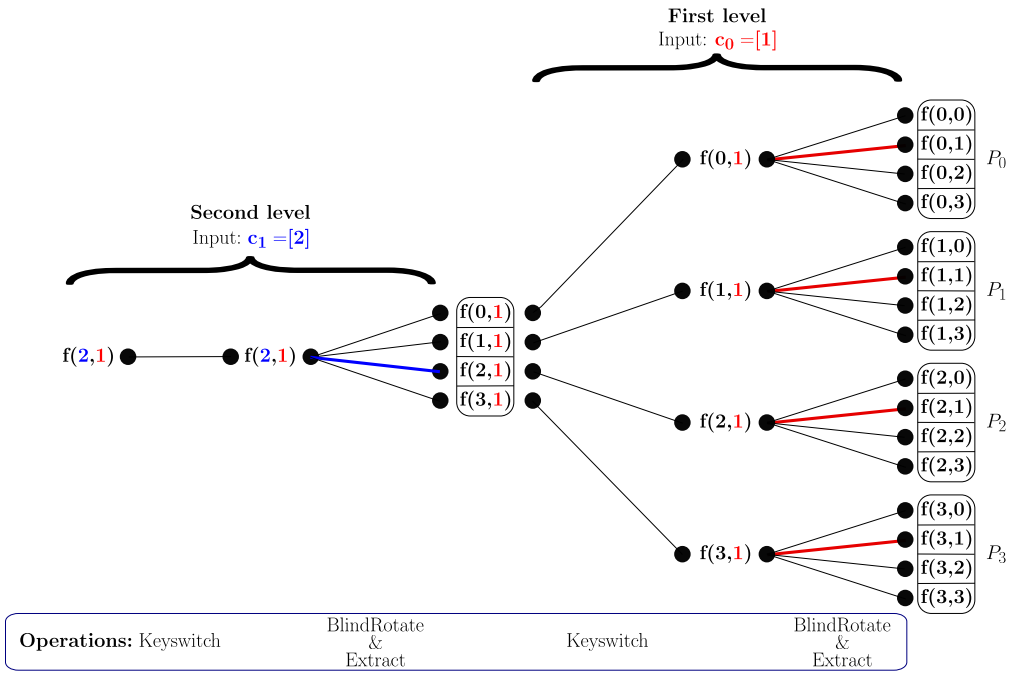


Figure 5.1 – Tree of depth 2 in basis 4 with inputs c_0 and c_1 encrypting 1 and 2 respectively.

We can accelerate the tree evaluation by encoding the first LUTs in plaintext polynomials rather than TRLWE ciphertexts. Then, we use the multi-value bootstrapping from [16] to compute only one BlindRotate instead of B^{d-1} at the first level of the tree. We call *selector* the result of the BlindRotate of the first layer. Thus, we now compute: $1 + \sum_{i=0}^{d-2} B^i = 1 + \frac{B^{d-1}-1}{B-1}$ BlindRotate.

Note that we can build any function from $[\![0, B-1]\!]^d$ to $[\![0, B-1]\!]^k$ given any $k, d \in \mathbb{N}^*$ with the tree-based method. Indeed, any function from $[\![0, B-1]\!]^d$ to $[\![0, B-1]\!]^k$ can be decomposed as k functions from $[\![0, B-1]\!]^d$ to $[\![0, B-1]\!]$.

Noise variance

The noise variances of the Blindrotate and Keyswitch are additive. Thus, the noise variance of the tree-based method when applied to d inputs is less than $d \cdot \mathcal{E}_{BR} + (d-1)\mathcal{E}_{KS}^{N,B} + \mathcal{E}_{KS}^{N,1}$ or $d \cdot (\mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1}) + (d-1)\mathcal{E}_{KS}^{n,B}$ with the intermediary Keyswitch operations. In order to simplify the noise analysis, we only consider the tree-based method without the intermediary Keyswitch operations from now on. Note that our bound rely on our specific packing with redundancy technique introduced in Section 5.3.

If we implement the multi-value bootstrapping [16] for the evaluation of the first level of the tree with the polynomials $(P_i)_{i \in [\![0,d-1]\!]}$, the noise bound increases to $(d - 1 + \max(||P_i||_2^2)) \cdot \mathcal{E}_{BR} + (d-1) \cdot \mathcal{E}_{KS}^{N,B} + \mathcal{E}_{KS}^{N,1}$. Note that in the worst case $\max_i(||P_i||_2^2) \leq (B+3) \cdot (B-1)^2$ which leads to a worst case noise variance of $(d - 1 + (B+3) \cdot (B-1)^2) \cdot \mathcal{E}_{BR} + (d-1) \cdot \mathcal{E}_{KS}^{N,B} + \mathcal{E}_{KS}^{N,1}$.

Error rate

We consider that the input ciphertexts $(\boldsymbol{c}_i)_{i \in [\![0,d-1]\!]}$ encrypting the set of messages $(m_i)_{i \in [\![0,d-1]\!]}$ are mutually independent. We refer by $(V_{\boldsymbol{c}_i})_{i \in [\![0,d-1]\!]}$ to the noise variances of the input ciphertexts $(\boldsymbol{c}_i)_{i \in [\![0,d-1]\!]}$.

Each BlindRotate takes as input one of the ciphertext $\boldsymbol{c}_i$. As such, we can apply the Equation 5.6 from Section 5.4 to find the bound $\mathcal{F}_{\text{TM}} \leq \sum_{i=0}^{d-1} \mathcal{F}_B(V_{\boldsymbol{c}_i})$, where $\mathcal{F}_{\text{TM}}$ is the error rate of the tree based method. This bound holds true whether we use the multi-value bootstrapping or not.

5.5.2 Chaining Method

The chaining method has a much lower complexity and a lower error growth than the tree-based method. However, as stated in [48], it works only for a restricted set of functions, i.e., functions with carry-like logic.

We consider d TLWE ciphertexts $(\boldsymbol{c}_0, \ldots, \boldsymbol{c}_{d-1})$ respectively encrypting the messages $(m_0, \ldots, m_{d-1})$. We denote by $LC(a, b)$ *any* linear combination of a and b. Given a set of functions $(f_i)_{i \in [\![0,d-1]\!]}$ such that $f_i : [\![0, B-1]\!] \to [\![0, B-1]\!]$, we build with Algorithm 14 a function $f : [\![0, B-1]\!]^d \to [\![0, B-1]\!]$.

Algorithm 14 Chaining method

Input: A vector $(\boldsymbol{c}_0, \ldots, \boldsymbol{c}_{d-1})$ of TLWE ciphertexts encrypting the vector of messages $(m_0, \ldots, m_{d-1})$.

Output: A ciphertext encrypting $f(m_0, \ldots, m_{d-1})$. f is defined by the different linear combinations and the univariate functions f_i.

1: $\overline{\boldsymbol{c}}_0 \leftarrow f_0(\boldsymbol{c}_0)$
2: **for** $i \in [\![0, d-2]\!]$ **do**
3: $\quad \overline{\boldsymbol{c}}_{i+1} \leftarrow f_{i+1}(LC(\overline{\boldsymbol{c}}_i, \boldsymbol{c}_{i+1}))$
return $\overline{\boldsymbol{c}}_{d-1}$

The functions $(f_i)_{i \in [\![0,d-1]\!]}$ can be implemented using any method from the state of the art for univariate functional bootstrapping. We remind that in this paper, we choose the usual encoding method where the plaintext space is restricted to half of the torus [30].

Noise variance

Since Algorithm 14 ends with a functional bootstrapping, the resulting noise variance of its output ciphertext is bounded by $\mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1}$.

Error rate

The inputs of each BlindRotate are linear combinations of the independent ciphertexts $(\boldsymbol{c}_i)_{i \in [\![0,d-1]\!]}$. Thus, we bound the error rate by:

$$\mathcal{F}_{\text{CM}} \leq \mathcal{F}_B(V_{\boldsymbol{c}_0}) + \sum_{i=1}^{d-1} \mathcal{F}_B(V_{LC_i(c_{\mathsf{boot}}, \boldsymbol{c}_i)})$$

where LC_i is the linear combination used at the i^{th} step of Algorithm 14, c_{boot} is a freshly bootstrapped ciphertext and $\mathcal{F}_{\text{CM}}$ is the error rate of the chaining method.

5.5.3 Performances Comparison

Table 5.2 summarizes the noise variances and probabilities of success for the tree-based and chaining methods. We refer by TBM to the tree-based method without using the multi-value bootstrapping trick, by TMV to the tree-based method when using the multi-value bootstrapping, and by CM to the chaining method.

Table 5.3 summarizes the time complexity of both methods. As mentioned in Section 5.4, the time complexity is given as the number of BlindRotate and KeySwitch.

Method	Noise variance	Bound on error rate
TBM	$d \cdot \mathcal{E}_{BR} + (d-1)\mathcal{E}_{KS}^{N,B} + \mathcal{E}_{KS}^{N,1}$	$\sum_{i=0}^{d-1} \mathcal{F}_B(V_{\mathbf{c}_i})$
TMV	$(d-1+\max(\|\|P_i\|\|_2^2)) \cdot \mathcal{E}_{BR}$ $+(d-1) \cdot \mathcal{E}_{KS}^{N,B} + \mathcal{E}_{KS}^{N,1}$	$\sum_{i=0}^{d-1} \mathcal{F}_B(V_{\mathbf{c}_i})$
CM	$\mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1}$	$\mathcal{F}_B(V_{\mathbf{c}_0})$ $+\sum_{i=1}^{d-1} \mathcal{F}_B(V_{LC_i(c_{\mathbf{boot}},\mathbf{c}_i)})$

Table 5.2 – Noise variance and success rate in basis B for d inputs

Method	Blindrotate	KeySwitch	
		to TLWE	to TRLWE
TBM	$\frac{B^d-1}{B-1}$	1	$\frac{B^{d-1}-1}{B-1}$
TMV	$1+\frac{B^{d-1}-1}{B-1}$	1	$\frac{B^{d-1}-1}{B-1}$
CM	d	d	0

Table 5.3 – Time complexity in basis B for d inputs

It is straightforward to see from Table 5.2 and 5.3 that CM leads to a lower noise variance with more efficient computation. These benefits come with limitations since CM is not a method that generalize well to every function. Besides, the error rate can be much higher with CM depending on the linear combination involved in Algorithm 14. Note that $\mathcal{F}_B(x) = \text{erfc}(\frac{1}{4B\sqrt{2\cdot(x+V_r)}})$. We use the bound $\text{erfc}(X) \lesssim \frac{e^{-X^2}}{X\sqrt{\pi}}$ which has the same order of magnitude as erfc as long as $X > 1$ to get that

$$\mathcal{F}_B(x) \lesssim 4B \cdot e^{-\frac{1}{32B^2(x+V_r)}} \sqrt{\frac{2(x+V_r)}{\pi}} \tag{5.9}$$

From this formula we get that the growth of the error rate induced by the growth of x heavily depends on the relative size of x and V_r. Indeed, when $x \gg V_r$, $\mathcal{F}_B(x) \simeq 4B \cdot e^{-\frac{1}{32B^2\cdot x}} \sqrt{\frac{2x}{\pi}}$. Thus a growth of x has an exponential impact on the error rate. Whereas $x \ll V_r$ implies that $\mathcal{F}_B(x) \simeq 4B \cdot e^{-\frac{1}{32B^2\cdot V_r}} \sqrt{\frac{2V_r}{\pi}}$. Thus the growth of x is absorbed by the term V_r. Since $V_r = \frac{n+1}{48N^2}$, this means that

depending on the encryption parameters, the drawback on the error rate of CM can be mitigated.

5.6 Circuit Method

Both of the previous methods come with restrictions. On the one hand, the tree-based method requires an exponential number of BlindRotate relatively to the depth d of the tree. On the other hand, the chaining method can only be applied to some specific functions.

In the following sections, we propose an alternative method relying on logic circuits with encrypted inputs (in section 5.6.1). The plaintext space for these inputs is $[\![0, B-1]\!]$ for a given integer $B = 2^k$.

5.6.1 Extended Lupanov Bound

A *logic circuit* of a function f takes as input a set of d digits $\{d_0, \ldots, d_{d-1}\}$ in $[\![0, B-1]\!]$ and feeds them to a circuit of *B-gates* to evaluate $f(d_0, \ldots, d_{d-1})$. A B-gate is any function that takes at most 2 digits and outputs 1 digit. As such, a logic circuit as we mean it can be seen as an extension of binary circuits to non binary basis.

The Lupanov bound [67] states that any function $f : \mathbb{B}^d \to \mathbb{B}$ can be computed using a circuit with at most $\frac{2^d}{d}(1 + \circ(1))$ binary gates. This result can easily be extended to any base B logic circuit as follows. For any basis $B = 2^k$, a function $f : [\![0, B-1]\!]^d \to [\![0, B-1]\!]$ can be seen as a vector of k functions $f_i : [\![0, B-1]\!]^d \to \mathbb{B}$. Similarly, each function f_i can be seen as a function $g_i : \mathbb{B}^{k \cdot d} \to \mathbb{B}$. Applying the Lupanov bound to the functions g_i, we get that each of them can be computed with a circuit of size at most $\frac{2^{kd}}{kd}(1 + \circ(1))$. Thus the function f can be computed using a circuit of size $k \cdot \frac{2^{kd}}{kd}(1 + \circ(1)) = \frac{B^d}{d}(1 + \circ(1))$ which extends the Lupanov bound to functions with inputs in other bases.

This result shows that even for functions with large circuit representation, it is possible to improve on the purely exponential bound of the tree-based method. However, the circuit method truly shines when the computed functions are well structured. For instance, a list of encrypted messages can be sorted in quadratic time using a homomorphic circuit, which cannot be achieved using the tree-based method naively.

5.6.2 Computing B-gates

In this section, we describe in detail how to build B-gates. Each B-gate can be computed either with the chaining method CM, the tree-based method TBM, or the tree-based method with multi-value bootstrapping TMV.

Using CM as a building block

We can combine two digits x and y with the bijection:

$$g_2 : \begin{array}{ccc} [\![0, B-1]\!]^2 & \to & [\![0, B^2-1]\!] \\ (x, y) & \mapsto & x + B \cdot y \end{array} \tag{5.10}$$

That is, we compute any B-gate with encrypted digits $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ by applying one functional bootstrapping to $g_2(\boldsymbol{c}_1, \boldsymbol{c}_2)$. To that end, we need to use a plaintext size of B^2 instead of B to encrypt each digit. We summarize in Table 5.4 and Table 5.5 the noise variance, the error rate and the time complexity of a generic gate using CM found following the formulas from Table 5.2. From now on, we call CM-gate gates computed with the CM method.

Noise variance	**Error rate**
$\mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1}$	$\mathcal{F}_B(V_{\boldsymbol{c}_0} + B^2 \cdot V_{\boldsymbol{c}_1})$

Table 5.4 – CM-gate error rate and noise variance. $V_{\boldsymbol{c}_0}$ and $V_{\boldsymbol{c}_1}$ respectively represent the noise variance of the first and second inputs of the B-gate.

Blindrotate	Keyswitch	
	to TLWE	**to TRLWE**
1	1	0

Table 5.5 – CM-gate time complexity

Using TBM as a building block

B-gates can be computed as trees of depth 2. We summarize in Table 5.6 and Table 5.7 the noise variance, the error rate and the time complexity of a generic gate using TBM found following the formulas from Table 5.2 and Table 5.3. From now on, we call TBM-gate gates computed with the TBM method.

Noise variance	Error rate
$2 \cdot \mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,B} + \mathcal{E}_{KS}^{N,1}$	$\mathcal{F}_B(V_{\boldsymbol{c}_0}) + \mathcal{F}_B(V_{\boldsymbol{c}_1})$

Table 5.6 – TBM-gate error rate and noise variance. $V_{\boldsymbol{c}_0}$ and $V_{\boldsymbol{c}_1}$ respectively represent the noise variance of the first and second inputs of the B-gate.

Blindrotate	Keyswitch	
	to TLWE	to TRLWE
$B+1$	1	1

Table 5.7 – TBM-gate time complexity

Using TMV as a building block

Similarly to TBM, logic gates can be computed as trees of depth 2. Note that if multiple logic gates share an input, they can also share the "selector". This allows us to reduce the amount of BlindRotate in a circuit. We summarize in Table 5.8 and Table 5.9 the noise variance, the error rate and the time complexity of a generic gate using TMV found following the formulas from Table 5.2 and Table 5.3. We consider the polynomials maximizing the error rate to bound the resulting error rate. From now on, we call TMV-gate gates computed with the TMV method.

Noise variance	Error rate
$(1 + (B+3) \cdot (B-1)^2) \cdot \mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,B} + \mathcal{E}_{KS}^{N,1}$	$\mathcal{F}_B(V_{\boldsymbol{c}_0}) + \mathcal{F}_B(V_{\boldsymbol{c}_1})$

Table 5.8 – TMV-gate generic error rate and noise variance. $V_{\boldsymbol{c}_0}$ and $V_{\boldsymbol{c}_1}$ respectively represent the noise variance of the first and second inputs of the logic gate.

Blindrotate	Keyswitch	
	to TLWE	to TRLWE
2	1	1

Table 5.9 – TMV-gate generic time complexity

Example of TMV-gate

Any B-gate can be computed with ease using CM and TBM. However, TMV-gates require to understand the multi-value bootstrapping technique and leads to a noticeable impact on the resulting noise variance. As an example, we describe

the B-gate $\mathsf{AddGate}(x, y) = x + y[B]$. Note that we work on half of the torus, which prevents the modulus operation to be performed using a homomorphic addition.

Since additions are commutative, the same tree is used whether we want to use the first or the second input as the selector. We show in Figure 5.2 an example of this tree when $B = 4$. The polynomials we end up with are the Q_i for $i \in [\![0, B-1]\!]$ where $Q_i = \sum_{k=0}^{\frac{N}{B}-1} \sum_{j=0}^{B-1} ((i+k) \mod B) X^{k \cdot \frac{N}{B} + j}$. To apply the multi-value bootstrapping technique, we consider $P_i = (1 - X) \cdot Q_i$. Then $P_0 = (B-1) + \sum_{k=1}^{B-1} X^{k \cdot \frac{N}{B}}$ and for all $i > 0$, $P_i = (2i-1) - B \cdot X^{(B-i) \cdot \frac{N}{B}} + \sum_{k=1}^{B-1} X^{k \cdot \frac{N}{B}}$ as polynomial for the noise formulas. We get that $\max_i(||P_i||_2^2) = (B-1) \cdot (5B-8)$.
We summarize in Table 5.10 and Table 5.11 the noise variance, the error rate and the time complexity of a TMV-AddGate. The results are calculated following the formulas from Table 5.2 and Table 5.3.

Noise variance	**Error rate**
$(1 + (B-1) \cdot (5B-8)) \cdot \mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,B} + \mathcal{E}_{KS}^{N,1}$	$\mathcal{F}_B(V_{\boldsymbol{c}_0}) + \mathcal{F}_B(V_{\boldsymbol{c}_1})$

Table 5.10 – TMV-AddGate error rate and noise variance. $V_{\boldsymbol{c}_0}$ and $V_{\boldsymbol{c}_1}$ respectively represent the noise variance of the first and second inputs of the B-gate.

Blindrotate	Keyswitch	
	to TLWE	**to TRLWE**
2	1	1

Table 5.11 – TMV-AddGate time complexity. Note that the selector may be shared between multiple gates. Then, one less BlindRotate is necessary for each gate after the first.

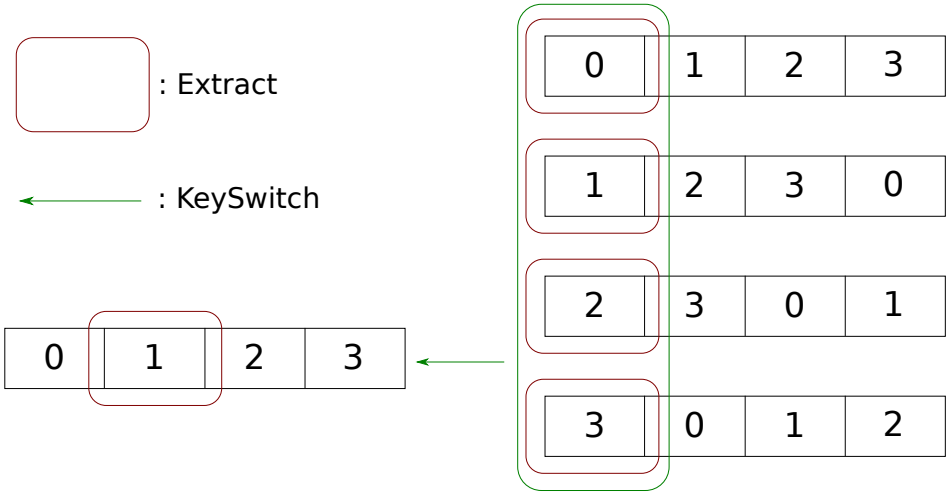


Figure 5.2 – Tree of AddGate

5.7 Empirical Performances

In this section we give empirical time results for computing generic B-gates. All computation were made on an Intel Core i5-8250U CPU @ 1.60GHz by extending the TFHE open source library[8]. We use the sets of parameters from Table 5.12, Table 5.13 and Table 5.14 to achieve 128 bits of security and an error rate of at most 2^{-32}.

The parameters are chosen following this methodology for each B-gate and each method:

- The parameters n and N are chosen as low as possible without jeopardizing security to ensure better speed performance.
- The parameters n, N, $\sigma_{\mathbb{T}}$ and $\sigma_{\mathbb{T}_N[X]}$ are chosen to reach at least $\lambda = 128$ bits of security.
- The parameters l, t, B_g, and B_{KS} lead to an error rate lower than 2^{-32} for the chosen B-gate and method for inputs with noise equal to the output of a B-gate. We keep l and t as low as possible for speed performance. We note respectively $B_g$bit and B_{KS}bit the $\log_2$ of B_g and B_{KS}.

[8] https://github.com/tfhe/tfhe

Basis	**n**	**N**	$\sigma_{\mathbb{T}}$	$\sigma_{\mathbb{T}_{\mathbf{N}}[\mathbf{X}]}$	**l**	$\mathbf{B_g bit}$	**t**	$\mathbf{B_{KS} bit}$
4	900	2048	$5.1 \cdot 10^{-7}$	$9.6 \cdot 10^{-11}$	3	8	6	3
8	1100	8192	$1.4 \cdot 10^{-8}$	$7 \cdot 10^{-65}$	1	31	7	3
16	1300	65536	$3.77 \cdot 10^{-10}$	$1 \cdot 10^{-300}$	1	31	4	6

Table 5.12 – Parameters sets with CM (λ=128)

Basis	**n**	**N**	$\sigma_{\mathbb{T}}$	$\sigma_{\mathbb{T}_{\mathbf{N}}[\mathbf{X}]}$	**l**	$\mathbf{B_g bit}$	**t**	$\mathbf{B_{KS} bit}$
4	800	1024	$3.1 \cdot 10^{-6}$	$5.6 \cdot 10^{-8}$	3	6	3	4
8	800	1024	$3.1 \cdot 10^{-6}$	$5.6 \cdot 10^{-8}$	5	4	7	2
16	900	2048	$5.1 \cdot 10^{-7}$	$9.6 \cdot 10^{-11}$	2	11	5	3

Table 5.13 – Parameters sets with TBM (λ=128)

Basis	**n**	**N**	$\sigma_{\mathbb{T}}$	$\sigma_{\mathbb{T}_{\mathbf{N}}[\mathbf{X}]}$	**l**	$\mathbf{B_g bit}$	**t**	$\mathbf{B_{KS} bit}$
4	800	2048	$3.1 \cdot 10^{-6}$	$9.6 \cdot 10^{-11}$	2	11	3	4
8	900	2048	$5.1 \cdot 10^{-7}$	$9.6 \cdot 10^{-11}$	3	8	3	5
16	1024	2048	$5.6 \cdot 10^{-8}$	$9.6 \cdot 10^{-11}$	6	5	3	6

Table 5.14 – Parameters sets with TMV (λ=128)

Given these sets of parameters, we show in Table 5.15 the time requirements of each method in basis 4, 8 and 16.

Basis	CM	TBM	TMV
4	177	351	528
8	617	1073	643
16	7016	3622	973

Table 5.15 – Gate Evaluation Time in ms

It is interesting to note that when using a small basis B, CM-gates are the most efficient. However, the quadratic growth of the plaintext space relatively to B greatly degrades its performances with larger bases. On the other side of the spectrum, TMV-gates become more and more interesting with larger bases. Besides, TBM-gates seem of limited interest since it is outclassed by CM-gates for small basis, and by TMV-gates for larger basis. This is due to the linear growth of the number of operations required to perform a TBM-gate with the size of the basis.

Note that the TFHElib does not natively have a TLWE to TRLWE KeySwitch implemented and our personal implementation of this KeySwitch does not make use of parallelism. Since this KeySwitch operation takes a large part of the computation time, the performances of TBM-gates and TMV-gates can be greatly optimised. Besides, the parameter sets used for TMV-gates are meant to work for any gate. In practice, the error bound must be tailored to specific gates, which will improve the overall performance of TMV-gates. Finally, performances must be optimized depending on the amount of memory available. For instance, multiple KeySwitch techniques exist as shown in Section 5.3 which lead to different trade-offs between noise growth, memory usage and speed.

5.8 Example: Sorting Algorithm

In this section, we compare a circuit dedicated to sorting a list of encrypted inputs in base B to the direct application of the tree-based method. Note that we do not know of any way of merging lists efficiently in the homomorphic domain. This prevents us from performing sorting algorithms with a worst-case complexity of $n\log(n)$. Thus, we describe in Algorithm 15 the bubble sort algorithm which is possible to implement homomorphically.

Algorithm 15 Bubble Sort

Input: A list $(\boldsymbol{c}_0, \boldsymbol{c}_1, ..., \boldsymbol{c}_{d-1})$ of d ciphertexts encrypting the messages $(m_0, m_1, ..., m_{d-1})$.

Output: A list of ciphertexts $(\boldsymbol{r}_0, \boldsymbol{r}_1, ..., \boldsymbol{r}_{d-1})$ encrypting the messages in sorted order.

1: $\boldsymbol{r}_0 = \boldsymbol{c}_0$
2: **for** $i \in [\![1, d-1]\!]$ **do**
3: $\quad (\boldsymbol{r}_0, \boldsymbol{r}_i) = (\min(\boldsymbol{r}_0, \boldsymbol{c}_i), \max(\boldsymbol{r}_0, \boldsymbol{c}_i))$
4: $\quad$ **for** $j \in [\![1, i-1]\!]$ **do**
5: $\quad\quad (\boldsymbol{r}_j, \boldsymbol{r}_i) = (\min(\boldsymbol{r}_j, \boldsymbol{r}_i), \max(\boldsymbol{r}_j, \boldsymbol{r}_i))$
return $(\boldsymbol{r}_0, \boldsymbol{r}_1, ..., \boldsymbol{r}_{d-1})$

The speed and noise performance of B-gates computed with CM and TBM are unrelated to the specific B-gates. Thus, we reuse the same parameter sets as in Table 5.12 and Table 5.13. Besides, the only required B-gates are $\min(x, y)$ and $\max(x, y)$ gates. Since the resulting noise using TMV depends on the function computed, we give the bound on the noise relatively to these two B-gates:

- min gate noise variance:

$$(1 + B \cdot (B-1)) \cdot \mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,B} + \mathcal{E}_{KS}^{N,1}$$

- max gate noise variance:

$$(1 + 4 \cdot (B-1)^2) \cdot \mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,B} + \mathcal{E}_{KS}^{N,1}$$

As such, both noise variances are bounded by $(1+4\cdot(B-1)^2)\cdot\mathcal{E}_{BR}+\mathcal{E}_{KS}^{N,B}+\mathcal{E}_{KS}^{N,1}$. We use this formula to find the parameters from Table 5.16, leading to an error rate lower than 2^{-32} per gate with TMV.

Basis	**n**	**N**	$\sigma_{\mathbb{T}}$	$\sigma_{\mathbb{T}_{\mathbf{N}}[\mathbf{X}]}$	**l**	$\mathbf{B_g}$**bit**	**t**	$\mathbf{B_{KS}}$**bit**
4	800	1024	$3.1 \cdot 10^{-6}$	$5.6 \cdot 10^{-8}$	6	3	3	4
8	900	2048	$5.1 \cdot 10^{-7}$	$9.6 \cdot 10^{-11}$	3	8	3	5
16	1024	2048	$5.6 \cdot 10^{-8}$	$9.6 \cdot 10^{-11}$	5	6	3	6

Table 5.16 – Parameters sets with TMV for sorting circuit (λ=128)

Using the tree-based method naively to sort a list of d encrypted inputs would require the computation of d trees of depth d, leading to a total of approximately $d \cdot B^{d-1}$ operations instead of d^2. We use the same parameter sets for the naive tree-based method as for TBM-gates to reach an error rate lower than 2^{-32} per BlindRotate.

We show in Table 5.17 the empirical time result we get for each method. We call CM-circuit, TBM-circuit and TMV-circuit, circuits build with CM-gates, TBM-gates and TMV-gates, respectively.

Basis	CM-**circuit**	TBM-**circuit**	TMV-**circuit**	**Tree-Based Method**
4	3.00	6.19	3.88	14.94
8	10.41	19.14	10.53	134.77
16	85.14	56.40	12.59	2083.63

Table 5.17 – Sorting Time (in s) of 4 inputs.

As seen in Table 5.17, the naive tree-based method becomes prohibitively long even for small bases and low number of inputs. Our circuit method allows more flexibility in the way homomorphic computation are performed and thus reach much better performances for highly structured functions such as sorting functions.

5.9 Conclusion

We introduced and compared multiple techniques to compute B-gates as efficient building blocks for the computation of base B logic circuits. Our approach allows for efficient evaluation of structured functions, largely improving on the speed performance of the tree-based method from the literature. As a side result, we introduced a keyswitching key specific to packing TLWE ciphertexts into TRLWE ciphertexts with redundancy, which is of separate interest.

To go further, it would be interesting to compare our method to [5] which relies on the circuit bootstrapping not implemented natively in TFHElib. It would also be interesting to optimize the implementation of B-gates by introducing parallelism to harness the full potential of this technique.

Our method can also be of interest to research on homomorphic compilers such as Cingulata [15], Transpiler [47] and the Concrete compiler [29] as an alternative to binary computation.

Chapter 6

Selection of Applications

This chapter compiles some of the more practical research done during this thesis on the functional bootstrapping of TFHE. In the first section, we compare 3 cryptosystems for the computation of small feedforward neural networks. In the second section, we describe our work on the implementation of a recurrent neural network over encrypted inputs. Finally, in the third section, we show how we performed the transciphering of Grain128-AEAD using TFHE.

6.1 Comparison of Cryptosystems on Small Neural Networks

This section is based on our article [34] on cryptosystem comparison for cloud evaluation of small neural networks, written in collaboration with Oana Stan and Martin Zuber. This is a work done early in the thesis which partly determined the direction we took for the remainder of the thesis.

6.1.1 Introduction

The use of remote machine learning algorithms, and in particular neural networks, raises major privacy issues which can be avoided through the use of FHE. However, choosing the right cryptosystem for a given application can be difficult and have tremendous impact on the performances of said application. Thus, we investigate in this work the differences between three of the most used FHE cryptosystems, namely BFV, CKKS and TFHE, regarding the evaluation of small neural networks in the cloud.

For each cryptosystem we built a small neural network with one hidden layer and

the same amount of neurons, with the task of discriminating between handwritten digits from the MNIST database [60]. We use FHE friendly activation functions specific to each cryptosystem. More specifically, we use the square function for BFV and CKKS, and the sign function for TFHE. We show in Figure 6.1 the structure of our neural networks. The activation function is simply replaced by the sign or square function depending on the cryptosystem.

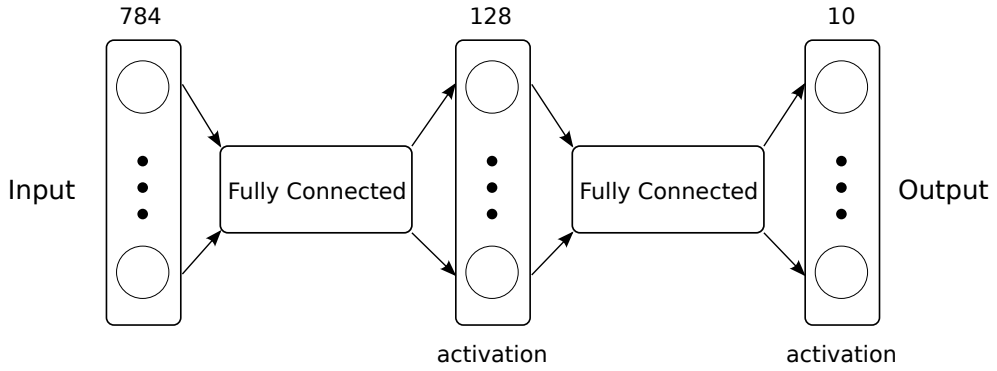


Figure 6.1 – Structure of our network

Even though this network is extremely simplistic, its homomorphic evaluation already highlights many challenges we have to face in order to compute neural network on encrypted data, and gives insight on the possibility to scale each method. Besides, we show in [6] that such simple architecture can already be of use for practical tasks such as driver behaviour analysis. Note that only the inference phase of the networks is performed on encrypted data and this work does not dive into the possibility of homomorphic training.

6.1.2 Sign Network

The main difficulty to tackle with TFHE is the strong discretization required to fit its small plaintext space. This is especially true since the lack of knowledge of the cryptosystem at the time lead us to use the default (and deprecated) 128-bit security parameters of TFHElib [27] (currently only 119 bits of security) rather than a custom set tailored for our purpose. We used 2^{30} values to quantize the torus as to ensure that the result of each operation could hold in one ciphertext. As such, errors due to noise overflowing are expected.

We discretize the weights of the network over 8 bits while keeping the inputs as is. The sign activation function allows us to naturally reduce the number of bits required at each step of the homomorphic evaluation of the network which fits

perfectly with the requirements on the plaintext space of a TFHE encryption. Besides, the sign function is freely computed during the bootstrapping procedure. This also mitigates the errors due to noise overflow as long as the sign of the message is not changed.

On the other hand, BFV and CKKS are ill suited to compute such a network since the sign function cannot be approximated properly near 0 with polynomials. This highlights one of the main benefits of the TFHE cryptosystem: its ability to compute non polynomial and even discontinuous functions thanks to its bootstrapping.

6.1.3 Square Network

Similarly to the sign network, we discretized the weights of the network over 8 bits and kept the inputs as is when using TFHE and BFV. However, the use of the square activation function makes the output of each layer larger and larger, which prevents TFHE from holding all the information about the output of each cell in only one ciphertext. This leads us to use TFHE in binary mode.

The CKKS cryptosystem handles floating point arithmetic which allows us to avoid any explicit discretization on the weights of the cryptosystem.

Besides, we investigate multiple batching techniques to make the best use of BFV and CKKS. We independently end up with similar batching techniques as in [13] and as such reuse some of their terminology. The different vector representations that we identify are:

- **Dense representation:** a vector $\boldsymbol{v} = (v_0, ..., v_{n-1})$ is represented as a message $\mu = \sum_{i=0}^{n-1} v_i \cdot X^i$.
- **Redundant representation:** a vector $\boldsymbol{v} = (v_0, ..., v_{n-1})$ is represented as a message $\mu = \sum_{i=0}^{n-1} \sum_{k=0}^{r-1} v_i \cdot X^{r \cdot i + k}$ where r is a redundancy variable. This representation is not used in [13].
- **Stacked representation:** a vector $\boldsymbol{v} = (v_0, ..., v_{n-1})$ is represented as a message $\mu = \sum_{i=0}^{n-1} \sum_{k=0}^{r-1} v_i \cdot X^{n \cdot k + i}$.

We also identify the corresponding matrix vector multiplications given a matrix $M = [m_{i,j}] \in \mathcal{M}_{r,n}$ and a message μ encrypting the vector $\boldsymbol{v} = (v_0, ..., v_{n-1})$. In

each case, we note s the number of slots available in a ciphertext (which is a power of 2) and we assume that $r = \frac{s}{n}$ with n divisor of s:

- **Redundant vector multiplication:**
  1. Multiply μ and $(m_{0,0}, m_{1,0}, ..., m_{m-1,0}, m_{0,1}, ..., m_{m-1,n-1})$ to get an encryption of $(m_{0,0} \cdot v_0, m_{1,0} \cdot v_0, ..., m_{m-1,0} \cdot v_0, m_{0,1} \cdot v_1, ..., m_{m-1,n-1} \cdot v_{n-1})$.
  2. Apply $\log_2(n)$ rotations and additions to get $(l_0, l_1, ..., l_{m-1}, l_0, ..., l_{m-1})$ where l_j is the result of the dot product between the j^{th} line of M and the vector $\boldsymbol{v}$. Thus we end up with a stacked representation of the result of the multiplication between M and $\boldsymbol{v}$.
- **Stacked vector multiplication:**
  1. Multiply μ and $(m_{0,0}, m_{0,1}, ..., m_{0,n-1}, m_{1,0}, ..., m_{m-1,n-1})$ to get an encryption of $(m_{0,0} \cdot v_0, m_{0,1} \cdot v_1, ..., m_{0,n-1} \cdot v_{n-1}, m_{1,0} \cdot v_0, ..., m_{m-1,n-1} \cdot v_{n-1})$.
  2. Apply $\log_2(n)$ rotations and additions to get $(l_0, *, ..., *, l_1, ..., l_{m-1})$ where l_j is in the $(j \times n)^{\text{th}}$ slot of the resulting ciphertext. The other slots hold useless values in regard to the matrix multiplication. Thus we end up with a dense representation of a vector holding the result of the multiplication between M and $\boldsymbol{v}$ in specific slots.

Note that our assumptions on r and n are not restrictive as we can either extend the matrix M and vector v with zeros and/or cut the matrix M in sub-matrices along the lines so that each matrix multiplication fits the assumption.

Since our network has only one hidden layer, we can chain the representations redundant $\rightarrow$ stacked $\rightarrow$ dense from input to output to get the wanted results. Other vector representations have to be used to compute a deeper network.

6.1.4 Performances and Conclusions

All the experiments are made on an Intel Core i7-6600U CPU @ 2.60GHz. We used the version 3.5.4 of SEAL [85] to use the BFV and CKKS cryptosystems, and the version 1.1 of TFHElib for TFHE.

In order to obtain secure parameters, the SEAL implementation of BFV and CKKS requires that we give the polynomial degree N and either the number of bits needed to write the plaintext modulus p with BFV, or a list which represents the number of bits of all the primes in the decomposition of the ciphertext modulus with CKKS.

For BFV, we need p to be large enough to encode the result of the neural network when each step is scaled to integers. Besides, we need it to be as small as possible

for efficiency. Thus we end up with a modulo p over 50 bits. Furthermore, the multiplicative depth of computation that can be evaluated is approximately $\frac{q}{p}$ where q is the ciphertext modulus. Knowing that the polynomial degree N follows approximately the formula $N = 1024 * \log_2(q)/27$ and must be a power of 2, we settled in our case with $N = 16384$. We then use the built in function of the SEAL library to get secure parameters according to N and the number of bits of p. This gives a ciphertext modulus using 390 bits. According to Albrecht's lwe-estimator [2], the security was of 146 bits at the time of the experiment which is more than high enough for most applications.

For CKKS, the list of primes must have two more values than the number of multiplications to be made. A generally good strategy to choose them according to SEAL's guidelines, is to have the first values at 60 bits as it will give the most precision, the last value also at 60 bits as it must be at least as large as the other values, and choose every other values close to each other. The resulting list is $\{60, 40, 40, 40, 40, 60\}$ which makes a ciphertext modulo using 280 bits. The sum of the values in the list must stay lower to a bound depending on the polynomial degree N. Same as with BFV, we chose a polynomial degree of $N = 16384$, as it was the smallest value that allowed the list above. We found a security of 279 bits at the time of the experiment using the lwe-estimator.

Finally, we used the default parameters for 128 bits of security of TFHElib for the TFHE implementation.

We summarize in Table 6.1 the computation time of each method.

BFV	**CKKS**	**TFHE (sign)**	**Binary TFHE**
$0.97s$	$0.24s$	$2.5s$	> 3 days*

Table 6.1 – Time performance with each cryptosystem
* : estimation based on previous performance.

Using BFV and CKKS did not lead to any degradation of the accuracy of the evaluated network. This shows that neural networks can be extremely resilient to discretization. Besides the fast evaluation of the network makes it viable for real life situation. However, this is mainly due to the fact that the network using a square activation is a low degree polynomial, which can be computed naturally and efficiently with FHE cryptosystems. The main hindrances to use these two cryptosystems for neural networks evaluation appear to be the following:

- The bootstrapping procedure must be avoided to reach low latency. This limits the depth of the evaluated network as choosing bigger parameters can quickly lead to prohibitive costs in time and memory.

- It is hard to find fitting batching techniques to reach low latency with a deeper network. Indeed, most works focus on low throughput rather than low latency.
- Finding a fitting polynomial approximation for standard activation functions is a challenge.

Using TFHE for the sign network gives us a drop of 6 percentage points in accuracy due to the high noise relatively to the heavy discretization of the torus[1]. Considering the very high level of noise relatively to the discretization of the torus in this experiment, this proves that this network has good resilience against noisy inputs and noisy computation. Furthermore, we obtain a relatively low latency even though we are using the bootstrapping procedure. However, using TFHE in binary mode leads to prohibitively long latency. The main hindrances to use TFHE for neural networks evaluation appear to be the following:

- A heavy discretization of the torus leads to noisy computation, while a light discretization of the torus leads to a very small plaintext space.
- There is no automated tool to choose TFHE parameters and this task can be tedious.
- The use of binary TFHE is prohibitively long for such a task.

These pieces of insight lead us to dig deeper into TFHE's properties. Notably, this gave an incentive to analyse TFHE's functional bootstrapping and noise level as shown in Chapter 4 and to find ways to compute non binary circuits with TFHE as shown in Chapter 5.

6.2 Homomorphic LSTM

This section summarizes our article [97] on building blocks for the evaluation of LSTMs done in collaboration with Daphné Trama, Aymen Boudguiga and Renaud Sirdey and published in CSCML 2023.

6.2.1 Introduction

A Long Short-Termed Memory network (LSTM) is a type of recurrent neural network introduced in 1997 by Hochreiter and Schmidhuber [51]. It has a wide range of applications, from anomaly detection in time series to music composition, as well as for prediction in medical care pathways and more. Some of these application, notably those related to medical data, require that the privacy of the client

[1]The article does not mention this drop in accuracy due to a mistake in implementation.

is ensured. Thus, LSTM networks fall in the wide range of neural networks benefiting from the privacy gained through homomorphic evaluation. However, the recurring nature of this type of network naturally excludes LHE schemes as the multiplicative depth of the computation is not known in advance. This naturally leads us to try and compute such a network with the TFHE cryptosystem and its efficient bootstrapping.

The role of cells in usual neural network is played by units inside of an LSTM network. Similarly, an LSTM layer plays the same role as a layer of neurons except that an LSTM layer loops on itself due to its recurring nature. A unit in the i^{th} loop takes three vector of inputs: the i^{th} input to the network x_i, the activation vector of the last layer a_{i-1}, and the memory cell vector c_{i-1}. The outputs of the i^{th} loop are the i^{th} output of the network y_i, an activation vector a_i, and a memory variable vector c_i. The outputs are computed through a forget gate $\Gamma_f^{\langle i \rangle}$, an update gate $\Gamma_u^{\langle i \rangle}$, and an output gate $\Gamma_o^{\langle i \rangle}$. The equations needed to evaluate the i^{th} loop are as follows:

$$
\begin{aligned}
\tilde{c}_i &= \mathsf{tanh}(W_c \cdot [a_{i-1}, x_i] + b_c) \\
\Gamma_u^{\langle i \rangle} &= \sigma(W_u \cdot [a_{i-1}, x_i] + b_u) \\
\Gamma_f^{\langle i \rangle} &= \sigma(W_f \cdot [a_{i-1}, x_i] + b_f) \\
\Gamma_o^{\langle i \rangle} &= \sigma(W_o \cdot [a_{i-1}, x_i] + b_o) \\
c_i &= \Gamma_u^{\langle i \rangle} \odot \tilde{c}_i + \Gamma_f^{\langle i \rangle} \odot c_{i-1} \\
a_i &= \Gamma_o^{\langle i \rangle} \odot \mathsf{tanh}(c_i) \\
y_i &= f(a_i)
\end{aligned}
$$

where σ denotes the sigmoid function, $\odot$ denotes the Hadamard product, W_j are weight matrices and b_j are bias vectors. We summarize in Figure 6.2 the basic structure of a LSTM layer.

Given the equation to compute, we aim to efficiently compute dot products, sigmoid functions, and tanh functions with enough precision to prevent the precision of the neural network to drop. In this work, we focused more specifically on the computation of the activation functions sigmoid and tanh to evaluate the feasibility of evaluating LSTMs in the encrypted domain.

6.2.2 LSTM Discretization

In order to evaluate the efficiency attainable with TFHE for the computation of LSTM units, we based our network on the work of Woodbridge et al., [98]. The full network contains an embedding layer, a 128 units LSTM layer that we want

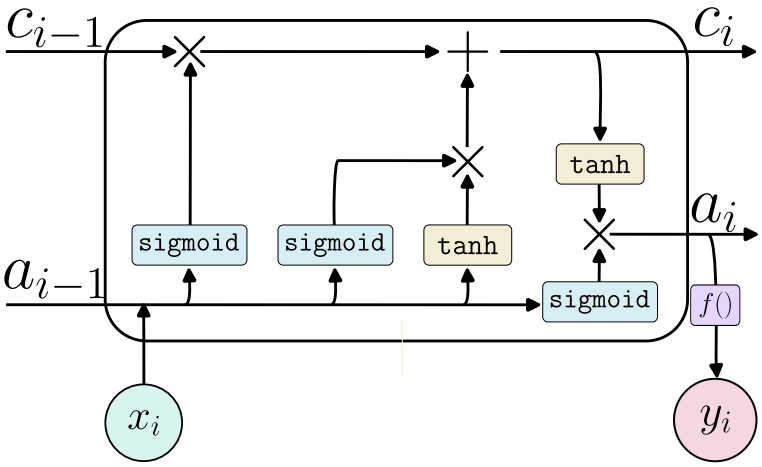


Figure 6.2 – Structure of a LSTM layer

to evaluate, and a fully connected layer. It was trained for Domain Generation Algorithm (DGA) recognition and we found an accuracy of 95.6% on a test set when reproducing their experimental results. However, TFHE does not handle floating points arithmetic which lead us to discretize the LSTM layer.

We first experiment in the clear domain to find the strongest level of quantization on the weights and inputs that does not impact the accuracy of the neural network negatively. We find that using only 4 bits of information is enough for each coefficient of the inputs and each weight coefficient to maintain the same level of accuracy.

We then aim to find a suitable discretization of the activation functions. As a first step, we try to approximate both the Sigmoid and Tanh using piecewise linear functions. We settle with 5-piece approximations for both the Sigmoid and Tanh functions defined as follows:

$$\begin{aligned}\sigma(x) &\simeq 0.13 \cdot \mathbb{1}_{]-6,-1]}(x) \;+\; \mathbb{1}_{]-1,1]}(x) \cdot (0.24x + 0.5) \;+\; 0.87 \cdot \mathbb{1}_{]1,6]}(x) \\ &\quad +\; \cdot \mathbb{1}_{]6,\infty[}(x) \\ \mathsf{Tanh}(x) &\simeq -1 \cdot \mathbb{1}_{]-\infty,-3]}(x) \;-\; 0.875 \cdot \mathbb{1}_{]-3,-1]}(x) \;+\; \mathbb{1}_{]-1,1]}(x) \cdot (0.76x) \\ &\quad +\; 0.92 \cdot \mathbb{1}_{]1,6]}(x) \;+\; \cdot \mathbb{1}_{]6,\infty[}(x).\end{aligned}$$

These approximations deteriorate the accuracy down to 93.6% which we estimate to be acceptable while keeping the computation simple enough for homomorphic computation. As a comparison, using the sign and Heaviside functions to approximate the Tanh and Sigmoid deteriorates the accuracy down to 50%.

We then discretize the linear part over 12 values which maintains the 93.6% accuracy thanks to the k-means algorithm. The final discretization used is summarized in Table 6.2 where StepSigmoid is the discretized Sigmoid and StepTanh is the discretized Tanh.

$x \in$	StepSigmoid(x)
]-∞, -6]	0
]-6, -1]	0.13
]-1, -0.834]	0.26
]-0.834, -0.668]	0.30
]-0.668, -0.501]	0.34
]-0.501, -0.335]	0.39
]-0.335, -0.169]	0.43
]-0.169, 0]	0.47
]0, 0.166]	0.52
]0.166, 0.332]	0.56
]0.332, 0.499]	0.60
]0.499, 0.665]	0.65
]0.665, 0.831]	0.69
]0.831, 1]	0.74
]1, 6]	0.87
]6, +∞]	1

$x \in$	StepTanh(x)
]-∞, -3]	-1
]-3, -1]	-0.875
]-1, -0.834]	-0.76
]-0.834, -0.668]	-0.622
]-0.668, -0.501]	-0.484
]-0.501, -0.335]	-0.346
]-0.335, -0.169]	-0.207
]-0.169, 0]	-0.069
]0, 0.166]	0.069
]0.166, 0.332]	0.207
]0.332, 0.499]	0.346
]0.499, 0.665]	0.484
]0.665, 0.831]	0.622
]0.831, 1]	0.76
]1, 3]	0.92
]3, +∞]	1

Table 6.2 – Our 16-steps StepSigmoid and StepTanh.

6.2.3 FHE implementation

The FHE implementation of the activation functions uses the tree-based method with inputs in basis 4 and outputs in basis 16. For each activation function, we map each of the 16 possible outputs of the quantized function to a value in $\{\frac{0}{32}, ..., \frac{15}{32}\}$.

In order to evaluate the precision of the homomorphic activation functions, we computed the rest of the network in the clear domain, which did not lead to any loss of accuracy for the network. We performed our experiment with the default parameters of TFHElib on a 4-core Intel Core i7-7600U 2.90GHz CPU (*with only one core activated*) and 16GiB total system memory with a Ubuntu 20.04.5 LTS server.

The evaluation of each activation function takes 0.15 second which would lead to a total of 96 seconds to compute all the activation functions of the 128 units of the LSTM.

6.2.4 Conclusions and Perspectives

Our experiment shows that LSTM networks are resilient to a relatively strong discretization, which is promising for their implementation in the homomorphic domain. However, even with our simplified framework, we evaluate that the computation of the LSTM layer would take minutes. Thus, a low latency LSTM in the encrypted domain is still a challenging task to perform.

Besides, we raise a notable difficulty with our approach. The uneven discretization that we obtain using the k-means algorithm for the output of each activation function makes the computation of the dot products harder. As such, we still need to investigate techniques to make this unusual discretization fit with the evaluation of the rest of the network. This can be avoided by using either base decomposition as in Chapter 5 at the cost of a slower computation of the activation function.

Another line of investigation is to verify how resilient the network is to noisy computation. This would allow us to avoid base decomposition entirely by considering a large plaintext space with TFHE at the cost of non negligible noise in the computation. This could lead to great improvement of the latency assuming that it does not impact the accuracy of the network too heavily.

6.3 Transciphering

This section summarizes our article [4] on transciphering with TFHE done in collaboration with Adda-Akram Bendoukha, Aymen Boudguiga and Renaud Sirdey, and published in DBSEC2023.

6.3.1 Introduction

One notable drawback of FHE is the expansion factor of the data when encrypted. This limits its applicability for embedded systems and other systems with low memory requirements. Transciphering allows such systems to encrypt their data with a low expansion factor cryptosystem while allowing for FHE computation. To that end, the client encrypts his message m with a symmetric encryption scheme as: $\mathsf{SYM}_{sk}(m)$, and encrypts the symmetric key sk with a homomorphic cryptosystem as: $\mathsf{FHE}_{\boldsymbol{s}}(sk)$. At the reception of $\mathsf{SYM}_{sk}(m)$ and $\mathsf{FHE}_{\boldsymbol{s}}(sk)$, the evaluation server homomorphically runs the symmetric cryptosystem's decryption function. This operation results in an encryption of m under the FHE cryptosystem $\mathsf{FHE}_{\boldsymbol{s}}(m)$.

With a stream-cipher, the client encrypts his message m with a keystream ks as: $m \oplus ks$, where $\oplus$ is the `XOR` operator. He then sends $m \oplus ks$ and the stream-cipher

secret key $\mathsf{FHE}_{\boldsymbol{s}}(sk)$ to the evaluation server which runs the stream-cipher warm-up homomorphically thanks to the encrypted symmetric key $\mathsf{FHE}_{\boldsymbol{s}}(sk)$ where sk is the secret key of the symmetric scheme and $\boldsymbol{s}$ is the secret key of the FHE scheme. This operation outputs an encryption of the keystream $\mathsf{FHE}_{\boldsymbol{s}}(ks)$. The server finally computes $m \oplus ks \oplus \mathsf{FHE}_{\boldsymbol{s}}(ks)$ homomorphically to obtain $\mathsf{FHE}_{\boldsymbol{s}}(m)$.

In [4], we explore the use of TFHE and its functional bootstrapping as a transciphering technique and validate its efficiency on Grain128-AEAD [50].

6.3.2 Grain128-AEAD

Grain128-AEAD [50] was a stream-cipher finalist in the NIST competition on lightweight cryptography which ended in February 2023. It builds on Grain128a [1] and extends it to support an Authenticated Encryption with Associated Data (AEAD) mode. This mode allows for the encryption of a subset of plaintext bits using a mask d with the formula $c_i = m_i \oplus (ks_i \cdot d_i)$. Additionally, a 64-bit Message Authentification Code (MAC) is computed on the encrypted data. The structure of Grain128-AEAD contains two main blocks: a 256-bit pre-output generator and a 128-bit authenticator generator. The pre-output generator contains a 128-bit Linear Feedback Shift Register (LFSR) and a 128-bit Non-linear Feedback Shift Register (NFSR) while the authenticator contains a 64-bit accumulator and a 64-bit shift registers for MAC computation. Figure 6.3 describes the structure of this stream cipher.

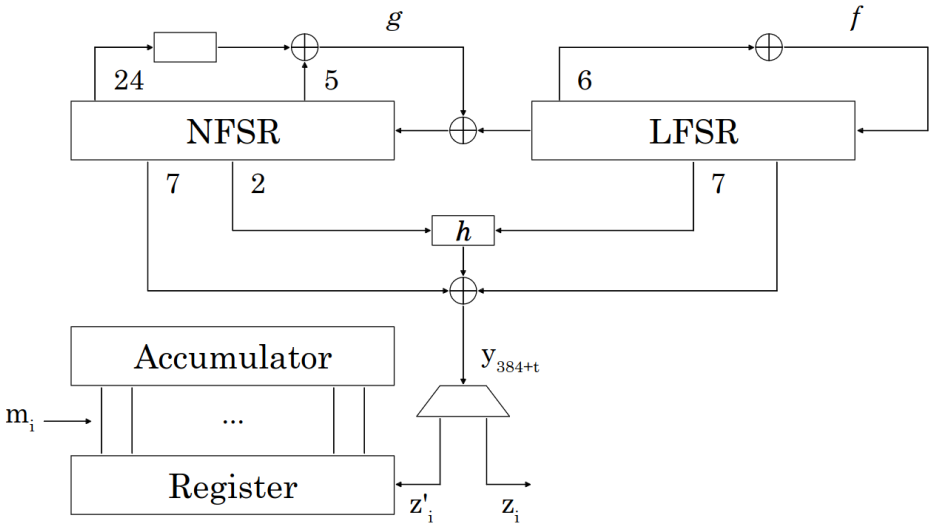


Figure 6.3 – Structure of Grain128-AEAD. Image extracted from [50]

After a 384 rounds warm up phase, Grain128-AEAD generates two streams of bits, namely the encryption keystream (ks) and the MAC keystream (ms), which are extracted from the main keystream using bit parity. Given that y_t denotes the t^{th} bit of the keystream outputted by Grain128-AEAD, ks_i is equal to y_{384+2i} and ms_i is equal to $y_{384+2i+1}$.

Notably, the only operations required to evaluate this stream cipher homomorphically are bitwise XOR and bit shifting operations.

6.3.3 The Set Up

In [14], Canteaut et al., suggest an efficient set up to minimize the amount of data exchanged between the embedded system and the evaluation server. They distinguish between 3 phases:

1. An offline key set up.
2. An offline decompression phase.
3. An online decompression phase.

In our stream-cipher context, these phases would correspond to:

1. The client send the evaluation keys to the server via a non memory constrained device. This operation is done once and for all before the embedded device start sending messages to the server.
2. The server can compute the homomorphic keystream $\mathsf{FHE}_{\boldsymbol{s}}(ks)$ from the evaluation keys.
3. The server only needs to compute the homomorphic XOR operation between the messages sent by the embedded system and the precomputed homomorphic keystream.

Using this simple set up, the constrained device does not need to send any FHE encrypted information such as the evaluation keys which can be very memory intensive. In particular, this legitimizes the use of memory intensive keyswitching keys which allow for faster computation with less noise growth, as discussed in 5.3.

6.3.4 Base B Adaptation

We aim to adapt this stream-cipher to multiple different bases B, and more specifically to bases in $\{2, 4, 16\}$. This allows us to compare the efficiency of the usual

binary TFHE and larger bases TFHE on this specific use case. For this experimentation, we used the CM type of B-gates as defined in Chapter 5 Section 5.6.1. As such, the natural modulus of TFHE corresponds to a modulus $2 \cdot B^2$. We now adapt some operations to this specific setting.

The first thing to note is that the bitwise XOR between the keystream and the message can be replaced by an addition in $\mathbb{Z}_B$ without any consequence on the security. As such, the homomorphic decryption becomes $[[m+ks]_B + \mathsf{FHE}_{\boldsymbol{s}}(ks)]_B$ where $[.]_B$ denotes the modulo B operation. We aim to make use of the natural modulus of TFHE to compute this sum without needing any functional bootstrapping for the modulus operation. To that end we can simply rescale values as follows:

$$[2B \cdot [m + ks]_B + \mathsf{FHE}_{\boldsymbol{s}}(2B \cdot ks)]_{2B^2}$$

The value $2B \cdot [m + ks]_B$ can be obtained easily since $[m + ks]$ is already in clear, and $\mathsf{FHE}_{\boldsymbol{s}}(2B \cdot ks)$ can be obtained at no extra cost by simply changing the last functional bootstrapping of the homomorphic computation of the keystream from $\mathsf{FHE}_{\boldsymbol{s}}(ks)$ to $\mathsf{FHE}_{\boldsymbol{s}}(2B \cdot ks)$. Thanks to this trick, the only costly part of the FHE evaluation lies in the homomorphic generation of the keystream which can be performed independently from the message m.

We compute the homomorphic generation of the keystream in a very straightforward fashion by simply implementing B-gates corresponding to the bitwise XOR operation and bit shift operations.

6.3.5 Experimental Results

We ran all the performance tests on a 12th Gen Intel(R) Core(TM) i7-12700H v6 @ 2.60GHz with 22GB RAM using a single core.

We show in Table 6.3 the parameter sets used to reach 128 bits of security. Note that here we use the memory intensive keyswitch keys, as it is not handled by the memory restricted device anyway. These parameters are chosen to reach an error rate of approximately 2^{-32} for the whole warm up circuit.

Basis	**n**	**N**	$\sigma_{\mathbb{T}}$	$\sigma_{\mathbb{T}_{\mathbf{N}}[\mathbf{X}]}$	**l**	$\mathbf{B_g}$**bit**	**t**	$\mathbf{B_{KS}}$**bit**
2	595	1024	$1.26 \cdot 10^{-4}$	$5.6 \cdot 10^{-8}$	4	5	2	10
4	740	2048	$9.17 \cdot 10^{-6}$	$9.6 \cdot 10^{-11}$	3	11	2	10
16	930	65536	$3 \cdot 10^{-7}$	$1 \cdot 10^{-100}$	1	32	2	10

Table 6.3 – Parameters sets for λ=128

We show in Table 6.4 the speed of the warm up phase given each basis. Note that generating additional bits simply requires $\frac{1}{384}^{\text{th}}$ of these timings per bit.

Basis	2	4	16
# of bootstrappings	18912	16608	7718
Single bootstrapping (in ms)	13	17	92
Warm up circuit (in min)	4.80	3.98	11.83

Table 6.4 – Time performance

Note that for the basis 16, the TMV was approximately 7 times faster than the CM method in 5.8. As such, it is fair to assume that a better choice of B-gates for basis 16 would have lead to much better performances, potentially even achieving better results than 3.98 minutes of basis 4. Nonetheless, the experimental results show that using bases larger than 2 can lead to better performances, even for applications which are extremely binary oriented. This reinforce our idea that exploring non binary bases to compute circuits is a worthwhile research domain.

This work inspired us to apply a similar transciphering technique to the AES in [96], where we successfully achieved a latency of 30 seconds for a full homomorphic evaluation of the AES.

Conclusion

Toward Efficient FHE

Data privacy is a growing concern in our increasingly connected society. Fully homomorphic encryption is a powerful tool that enables privacy in unforeseen circumstances. Indeed, as opposed to what common sense dictates, it is possible to make meaningful computation on encrypted data. This allows cloud services to provide more privacy to their clients and enables the conception of completely new applications intrinsically relying on privacy.

However, the main bottleneck for the adoption of this technology is its current computational cost, too high for many applications. As such, reducing the cost of homomorphic computation is the current main challenge for research on FHE. The conception of specific hardware devices for homomorphic computation gives promising results but are not enough on their own to make up for the homomorphic overhead. Thus, it is necessary to keep on enhancing software implementations to lower the cost of homomorphic computation.

The memory overhead coming with the use of FHE cryptosystem also needs to be taken into account. This makes transciphering friendly cryptosystems, such as TFHE, more interesting for a wide panel of applications as it mitigates or even completely avoid any memory overhead.

Finally, the ease of use of FHE cryptosystems can determinate whether this technology will get adopted. Indeed, allowing people with little to no expertise in cryptography to easily build applications relying on FHE is a good way to raise interest in this technology. To that end, the community works on FHE compilers such as Cingulata [15] or the more recent Concrete compiler [29].

The TFHE cryptosystem gathers many desirable characteristics, such as an efficient bootstrapping procedure, and the possibility to build logic circuits naturally. This makes it particularly well suited for the implementation of FHE compilers and for transciphering algorithms. Could it be the cryptosystem that will make

homomorphic computation cheap enough for real life implementation?

Contributions

Neural networks are a natural use case for FHE evaluation thanks to their great versatility. Hence, our first aim was to build neural networks that could be evaluated efficiently with encrypted inputs as shown in Section 6.1. This gave us much needed insight on the challenges of homomorphic computation and the pros and cons of each cryptosystems.

The main contribution, shown in Chapter 4, naturally followed to investigate in more detail the possibilities of TFHE's functional bootstrapping. We successfully built an efficient full torus functional bootstrapping technique which truly benefits from using the whole torus. The noise and error rate analysis described in the paper also allows for optimized parameter selection given a specific use case.

A second important contributions is shown in Chapter 5. It expands the toolkit of homomorphic circuits evaluation thanks to easy to compute non binary logic gates called B-gates. These B-gates are an interesting new building block for FHE computations which can notably be interesting to optimize the resulting circuits of FHE compilers.

Besides, we investigated multiple applications of homomorphic computation and TFHE's functional bootstrapping, such as LSTM evaluations described in Section 6.2 and transciphering described in Section 6.3.

Perspectives

Some applications remain extremely challenging to achieve in the homomorphic domain. In particular, we tried without success to train a small neural network entirely in the homomorphic domain. Indeed, the state of the art on homomorphic training generally rely on techniques such as federated learning where the homomorphic part of the computation is restricted to a small "aggregation" part. However, the literature is scarce concerning a completely homomorphic training, and papers on the subject such as [65, 74] omit crucial details to achieve this goal, such as how to manage the precision of the weights during training while using a strong quantization. As such, this subject is essentially unexplored as of now.

Looking for ways to make FHE faster is not the only way to improve homomorphic computation's applicability. Even though this thesis mainly focuses on improving

homomorphic operations to eventually compute neural networks in the homomorphic domain, the opposite is also a nice subject of research: how can we build accurate and FHE-friendly neural networks? Research on neural networks for embedded systems is a fruitful line of research which can give some answers to this question. Considering other technologies such as transfer learning is also a good way to lessen the burden on the homomorphic computation.

Fully homomorphic encryption has matured a lot since its birth in 2009 and continues to improve at great speed. It will certainly be a part of our daily life in the near future, opening new and exciting opportunities.

Bibliography

[1] Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. "Grain-128a: a new version of Grain-128 with optional authentication". In: *IJWMC* 5 (2011), pp. 48–59.

[2] Martin R. Albrecht, Rachel Player, and Sam Scott. *On the concrete hardness of Learning with Errors*. Cryptology ePrint Archive, Paper 2015/046. 2015.

[3] Joseph Antony, Kevin McGuinness, Noel E. O. Connor, and Kieran Moran. "Quantifying Radiographic Knee Osteoarthritis Severity Using Deep Convolutional Neural Networks". In: *arXiv:1609.02469 [cs]* (Sept. 2016). arXiv: 1609.02469 [cs].

[4] Adda-Akram Bendoukha, Pierre-Emmanuel Clet, Aymen Boudguiga, and Renaud Sirdey. *Optimized stream-cipher-based transciphering by means of functional-bootstrapping*. Cryptology ePrint Archive. 2023. URL: https://eprint.iacr.org/2023/1111.

[5] Loris Bergerat, Anas Boudi, Quentin Bourgerie, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. *Parameter Optimization & Larger Precision for (T)FHE*. Cryptology ePrint Archive, Paper 2022/704. 2022.

[6] Aymen Boudguiga, Oana Stan, Abdessamad Fazzat, Houda Labiod, and Pierre-Emmanuel Clet. "Privacy Preserving Services for Intelligent Transportation Systems with Homomorphic Encryption". In: Jan. 2021, pp. 684–693. DOI: 10.5220/0010349706840693.

[7] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. "CHIMERA: Combining Ring-LWE-based Fully Homomorphic Encryption Schemes". In: *Journal of Mathematical Cryptology* 14.1 (2020), pp. 316–338. DOI: https://doi.org/10.1515/jmc-2019-0026.

[8] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. "Simulating Homomorphic Evaluation of Deep Learning Predictions". In: *Cyber Security Cryptography and Machine Learning*. Ed. by Shlomi Dolev, Danny Hendler, Sachin Lodha, and Moti Yung. Cham: Springer International Publishing, 2019, pp. 212–230.

[9] F. Bourse, M. Minelli, M. Minihold, and P. Paillier. "Fast Homomorphic Evaluation of Deep Discretized Neural Networks". In: *Proceedings of CRYPTO 2018*. Springer, 2018.

[10] Florian Bourse, Olivier Sanders, and Jacques Traoré. *Improved Secure Integer Comparison via Homomorphic Encryption*. Cryptology ePrint Archive, Report 2019/427. 2019.

[11] Zvika Brakerski. "Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP". In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 868–886. ISBN: 978-3-642-32009-5.

[12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) Fully Homomorphic Encryption without Bootstrapping". In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ITCS '12. Cambridge, Massachusetts: Association for Computing Machinery, 2012, pp. 309–325. ISBN: 9781450311151. DOI: 10.1145/2090236.2090262.

[13] Alon Brutzkus, Oren Elisha, and Ran Gilad-Bachrach. "Low Latency Privacy Preserving Inference". In: *CoRR* abs/1812.10659 (2018). arXiv: 1812.10659. URL: http://arxiv.org/abs/1812.10659.

[14] 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*. Cryptology ePrint Archive, Paper 2015/113. 2015. URL: https://eprint.iacr.org/2015/113.

[15] 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 '15. Singapore, Republic of Singapore: Association for Computing Machinery, 2015, pp. 13–19. ISBN: 9781450334471. DOI: 10.1145/2732516.2732520. URL: https://doi.org/10.1145/2732516.2732520.

[16] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. "New Techniques for Multi-value Input Homomorphic Evaluation and Applications". In: *Topics in Cryptology – CT-RSA 2019*. Ed. by Mitsuru Matsui. Cham: Springer International Publishing, 2019, pp. 106–126. ISBN: 978-3-030-12612-4.

[17] Herve Chabanne, Roch Lescuyer, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. "Recognition Over Encrypted Faces: 4th International Conference, MSPN 2018, Paris, France". In: 2019.

[18] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. *Privacy-Preserving Classification on Deep Neural Network*. Cryptology ePrint Archive, Report 2017/035. 2017.

[19] Hao Chen, Ilaria Chillotti, and Yongsoo Song. *Improved Bootstrapping for Approximate Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2018/1043. 2018. URL: https://eprint.iacr.org/2018/1043.

[20] Jun Chen, Hanwen Chen, Mengmeng Wang, Guang Dai, Ivor W. Tsang, and Yong Liu. *Learning Discretized Neural Networks under Ricci Flow*. 2023. arXiv: 2302.03390 [cs.LG].

[21] Gong CHENG, Pujian LAI, Decheng GAO, and Junwei HAN. "Class attention network for image recognition". In: *SCIENCE CHINA Information Sciences* 66.3 (2023), pp. 132105–. DOI: https://doi.org/10.1007/s11432-021-3493-7.

[22] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. *Bootstrapping for Approximate Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2018/153. 2018. URL: https://eprint.iacr.org/2018/153.

[23] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. "Homomorphic Encryption for Arithmetic of Approximate Numbers". In: (2017). Ed. by Tsuyoshi Takagi and Thomas Peyrin.

[24] Jung Hee Cheon, Duhyeong Kim, and Jai Hyun Park. "Towards a Practical Clustering Analysis over Encrypted Data". In: *IACR Cryptology ePrint Archive* (2019).

[25] 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*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 3–33. ISBN: 978-3-662-53887-6.

[26] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. "Faster Packed Homomorphic Operations and Efficient Circuit Bootstrapping for TFHE". In: *ASIACRYPT*. 2017.

[27] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. *TFHE: Fast Fully Homomorphic Encryption Library*.

[28] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. "TFHE: Fast Fully Homomorphic Encryption Over the Torus". In: *Journal of Cryptology* 33 (Jan. 2020). DOI: 10.1007/s00145-019-09319-x.

[29] Ilaria Chillotti, Marc Joye, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. *CONCRETE: Concrete Operates oN Ciphertexts Rapidly by Extending TfhE*. WAHC 2020 - 8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography. HAL id:⟨hal-03926650⟩. 2020.

[30] Ilaria Chillotti, Marc Joye, and Pascal Paillier. "Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks". In: *Cyber Security Cryptography and Machine Learning*. Ed. by Shlomi Dolev, Oded Margalit, Benny Pinkas, and Alexander Schwarzmann. Cham: Springer International Publishing, 2021, pp. 1–19. ISBN: 978-3-030-78086-9.

[31] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. *Improved Programmable Bootstrapping with Larger Precision and Efficient Arithmetic Circuits for TFHE*. 2021. URL: https://ia.cr/2021/729.

[32] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. "Faster CryptoNets: Leveraging Sparsity for Real-World Encrypted Inference". In: *CoRR* abs/1811.09953 (2018). arXiv: 1811.09953. URL: http://arxiv.org/abs/1811.09953.

[33] Pierre-Emmanuel Clet, Aymen Boudguiga, Renaud Sirdey, and Martin Zuber. "ComBo: A Novel Functional Bootstrapping Method for Efficient Evaluation of Nonlinear Functions in the Encrypted Domain". In: *Progress in Cryptology - AFRICACRYPT 2023*. Springer, 2023. DOI: 10.1007/978-3-031-37679-5_14.

[34] Pierre-Emmanuel Clet, Oana Stan, and Martin Zuber. "BFV, CKKS, TFHE: Which One is the Best for a Secure Neural Network Evaluation in the Cloud?" In: *Applied Cryptography and Network Security Workshops*. Lecture Notes in Computer Science. Springer, 2021, pp. 279–300. ISBN: 978-3-030-81645-2. DOI: 10.1007/978-3-030-81645-2_16.

[35] Pierre-Emmanuel Clet, Martin Zuber, Aymen Boudguiga, Renaud Sirdey, and Cédric Gouy-Pailler. *Putting up the swiss army knife of homomorphic calculations by means of TFHE functional bootstrapping*. Cryptology ePrint Archive, Paper 2022/149. 2022. URL: https://eprint.iacr.org/2022/149.

[36] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. "Decaf: A Deep Convolutional Activation Feature for Generic Visual Recognition". In: *International Conference on Machine Learning*. PMLR, 2014, pp. 647–655.

[37] Léo Ducas and Daniele Micciancio. "FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second". In: *Advances in Cryptology – EUROCRYPT 2015*. Ed. by Elisabeth Oswald and Marc Fischlin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 617–640. ISBN: 978-3-662-46800-5.

[38] T. Elgamal. "A public key cryptosystem and a signature scheme based on discrete logarithms". In: *IEEE Transactions on Information Theory* 31.4 (1985), pp. 469–472. DOI: 10.1109/TIT.1985.1057074.

[39] Junfeng Fan and Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Report 2012/144. 2012. URL: https://ia.cr/2012/144.

[40] Robin Geelen, Ilia Iliashenko, Jiayi Kang, and Frederik Vercauteren. *On Polynomial Functions Modulo p^e and Faster Bootstrapping for Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2022/1364. 2022. DOI: 10.1007/978-3-031-30620-4_9. URL: https://eprint.iacr.org/2022/1364.

[41] Robin Geelen and Frederik Vercauteren. *Bootstrapping for BGV and BFV Revisited*. Cryptology ePrint Archive, Paper 2022/1363. 2022. DOI: 10.1007/s00145-023-09454-6. URL: https://eprint.iacr.org/2022/1363.

[42] Craig Gentry. "A Fully Homomorphic Encryption Scheme". PhD thesis. Stanford, CA, USA, 2009. ISBN: 9781109444506.

[43] Craig Gentry, Shai Halevi, and Nigel P. Smart. *Better Bootstrapping in Fully Homomorphic Encryption*. 2011. URL: https://eprint.iacr.org/2011/680.

[44] Craig Gentry, Shai Halevi, and Nigel P. Smart. *Fully Homomorphic Encryption with Polylog Overhead*. Cryptology ePrint Archive, Paper 2011/566. 2011. URL: https://eprint.iacr.org/2011/566.

[45] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. "CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, June 2016, pp. 201–210. URL: https://proceedings.mlr.press/v48/gilad-bachrach16.html.

[46] Shafi Goldwasser and Silvio Micali. "Probabilistic Encryption & How to Play Mental Poker Keeping Secret All Partial Information". In: *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*. Association for Computing Machinery, 1982. ISBN: 0897910702. DOI: 10.1145/800070.802212.

[47] Shruthi Gorantala et al. *A General Purpose Transpiler for Fully Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2021/811. 2021. URL: https://eprint.iacr.org/2021/811.

[48] Antonio Guimarães, Edson Borin, and Diego F. Aranha. "Revisiting the functional bootstrap in TFHE". In: 2021 (Feb. 2021), pp. 229–253. DOI: 10.46586/tches.v2021.i2.229-253.

[49] Minglun Han, Qingyu Wang, Tielin Zhang, Yi Wang, Duzhen Zhang, and Bo Xu. *Complex Dynamic Neurons Improved Spiking Transformer Network for Efficient Automatic Speech Recognition*. 2023. arXiv: 2302.01194 [cs.NE].

[50] Martin Hell, Thomas Johansson, Alexander Maximov, Willi Meier, and Hirotaka Yoshida. *Grain-128AEADv2: Strengthening the Initialization Against Key Reconstruction*. Cryptology ePrint Archive, Report 2021/751. 2021. URL: https://ia.cr/2021/751.

[51] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[52] Kai Huang, Bowen Li, Dongliang Xiong, Haitian Jiang, Xiaowen Jiang, Xiaolang Yan, Luc Claesen, Dehong Liu, Junjian Chen, and Zhili Liu. "Structured Dynamic Precision for Deep Neural Networks Quantization". In: *ACM Trans. Des. Autom. Electron. Syst.* 28.1 (2023). ISSN: 1084-4309. DOI: 10.1145/3549535.

[53] M. Izabachène, R. Sirdey, and M. Zuber. "Practical Fully Homomorphic Encryption for Fully Masked Neural Networks". In: *Cryptology and Network Security - 18th International Conference, CANS 2019, Proceedings*. Vol. 11829. Lecture Notes in Computer Science. Springer, 2019, pp. 24–36.

[54] Angela Jäschke and Frederik Armknecht. "Unsupervised Machine Learning on Encrypted Data". In: *IACR Cryptology ePrint Archive* (2018).

[55] Charanjit Singh Jutla and Nathan Manohar. *Sine Series Approximation of the Mod Function for Bootstrapping of Approximate HE*. Cryptology ePrint Archive, Paper 2021/572. 2021. URL: https://eprint.iacr.org/2021/572.

[56] Brady Kieffer, Morteza Babaie, Shivam Kalra, and H. R. Tizhoosh. "Convolutional Neural Networks for Histopathology Image Classification: Training vs. Using Pre-Trained Networks". In: *arXiv:1710.05726 [cs]* (Oct. 2017). arXiv: 1710.05726 [cs].

[57] Kamil Kluczniak and Leonard Schild. *FDFB: Full Domain Functional Bootstrapping Towards Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Report 2021/1135. 2021. URL: https://ia.cr/2021/1135.

[58] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: *University of Toronto* (May 2012).

[59] Kwok-Yan Lam, Xianhui Lu, Linru Zhang, Xiangning Wang, Huaxiong Wang, and Si Qi Goh. *Efficient FHE-based Privacy-Enhanced Neural Network for AI-as-a-Service*. Cryptology ePrint Archive, Paper 2023/647. 2023. URL: https://eprint.iacr.org/2023/647.

[60] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[61] Jaehyeok Lee, Phap Ngoc Duong, and Hanho Lee. "Configurable Encryption and Decryption Architectures for CKKS-Based Homomorphic Encryption". In: *Sensors* 23.17 (2023). ISSN: 1424-8220. DOI: 10.3390/s23177389. URL: https://www.mdpi.com/1424-8220/23/17/7389.

[62] Yongwoo Lee, Joon-Woo Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and HyungChul Kang. *High-Precision Bootstrapping for Approximate Homomorphic Encryption by Error Variance Minimization*. Cryptology ePrint Archive, Paper 2020/1549. 2020. URL: https://eprint.iacr.org/2020/1549.

[63] Feng-Hao Liu and Han Wang. "Batch Bootstrapping II: Bootstrapping in Polynomial Modulus Only Requires $\tilde{O}(1)$ FHE Multiplications in Amortization". In: Springer-Verlag, 2023. DOI: 10.1007/978-3-031-30620-4_12.

[64] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. "Large-Precision Homomorphic Sign Evaluation Using FHEW/TFHE Bootstrapping". In: *Advances in Cryptology – ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part II*. Taipei, Taiwan: Springer-Verlag, 2023, pp. 130–160. ISBN: 978-3-031-22965-7. DOI: 10.1007/978-3-031-22966-4_5.

[65] Qian Lou, Bo Feng, Geoffrey Charles Fox, and Lei Jiang. "Glyph: Fast and Accurately Training Deep Neural Networks on Encrypted Data". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 9193–9202.

[66] Qian Lou and Lei Jiang. "SHE: A Fast and Accurate Privacy-Preserving Deep Neural Network Via Leveled TFHE and Logarithmic Data Representation". In: *CoRR* abs/1906.00148 (2019). arXiv: 1906.00148. URL: http://arxiv.org/abs/1906.00148.

[67] Sergei Lozhkin and Alexander Shiganov. "On a Modification of Lupanov's Method with More Uniform Distribution of Fan-out." In: *Electronic Colloquium on Computational Complexity (ECCC)* 18 (Jan. 2011), p. 130.

[68] V Lyubashevsky, C. Peikert, and O. Regev. "On ideal lattices and learning with errors over rings". In: (2010).

[69] Abbass Madi, Oana Stan, Aurélien Mayoue, Arnaud Grivet-Sébert, Cédric Gouy-Pailler, and Renaud Sirdey. "A Secure Federated Learning framework using Homomorphic Encryption and Verifiable Computing". In: 2021, pp. 1–8. DOI: 10.1109/RDAAPS48126.2021.9452005.

[70] Anne-Sofie Maerten and Derya Soydaner. *From paintbrush to pixel: A review of deep neural networks in AI-generated art*. 2023. arXiv: 2302.10913 [cs.LG].

[71] Carlos Aguilar Melchor, Philippe Gaborit, and Javier Herranz. "Additively Homomorphic Encryption with d-Operand Multiplications". In: *Advances in Cryptology – CRYPTO 2010*. Springer Berlin Heidelberg, 2010, pp. 138–154. DOI: 10.1007/978-3-642-14623-7_8.

[72] Daniele Micciancio. "The Shortest Vector Problem is NP-hard to approximate to within some constant". In: *SIAM Journal on Computing* 30.6 (Mar. 2001). Preliminary version in FOCS 1998, pp. 2008–2035. DOI: 10.1137/S0097539700373039.

[73] Romain Mormont, Pierre Geurts, and Raphaël Marée. "Comparison of Deep Transfer Learning Strategies for Digital Pathology". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 2262–2271.

[74] Karthik Nandakumar, Nalini Ratha, Sharath Pankanti, and Shai Halevi. "Towards Deep Neural Network Training on Encrypted Data". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2019, pp. 40–48. DOI: 10.1109/CVPRW.2019.00011.

[75] Varun Kumar Ojha, Ajith Abraham, and Václav Snášel. "Metaheuristic design of feedforward neural networks: A review of two decades of research". In: *Engineering Applications of Artificial Intelligence* 60 (2017), pp. 97–116. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2017.01.013.

[76] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].

[77] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT'99. Prague, Czech Republic: Springer-Verlag, 1999, pp. 223–238. ISBN: 354065 8890.

[78] Hariharan Ravishankar, Prasad Sudhakar, Rahul Venkataramani, Sheshadri Thiruvenkadam, Pavan Annangi, Narayanan Babu, and Vivek Vaidya. "Understanding the Mechanisms of Deep Transfer Learning for Medical Images". In: *arXiv:1704.06040 [cs]* (Apr. 2017). arXiv: 1704.06040 [cs].

[79] Oded Regev. "On Lattices, Learning with Errors, Random Linear Codes, and Cryptography". In: (2005).

[80] R L Rivest, L Adleman, and M L Dertouzos. "On Data Banks and Privacy Homomorphisms". In: *Foundations of Secure Computation, Academia Press* (1978), pp. 169–179.

[81] R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Commun. ACM* (1978). ISSN: 0001-0782. DOI: 10.1145/359340.359342.

[82] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[83] Tomas Sander, Adam Young, and Moti Yung. "Non-Interactive Crypto-Computing For NC1". In: *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*. FOCS '99. USA: IEEE Computer Society, 1999, p. 554. ISBN: 0769504094.

[84] Murat H. Sazlı. "A brief review of feed-forward neural networks". In: *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering* 50.01 (2006). DOI: 10.1501/commua1-2_0000000026.

[85] *Microsoft SEAL (release 3.2)*. https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA. Feb. 2019.

[86] Arnaud Grivet Sébert, Rafael Pinot, Martin Zuber, Cédric Gouy-Pailler, and Renaud Sirdey. "SPEED: secure, PrivatE, and efficient deep learning". In: *Mach. Learn.* 110.4 (2021), pp. 675–694. DOI: 10.1007/s10994-021-05970-3.

[87] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. "Overfeat: Integrated Recognition, Localization and Detection Using Convolutional Networks". In: *arXiv preprint arXiv:1312.6229* (2013). arXiv: 1312.6229.

[88] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2014, pp. 806–813.

[89] David Silver et al. "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529 (2016), pp. 484–489. DOI: 10.1038/nature16961.

[90] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: (2014). arXiv: 1409.1556.

[91] N. P. Smart and F. Vercauteren. *Fully Homomorphic SIMD Operations*. Cryptology ePrint Archive, Paper 2011/133. 2011. URL: https://eprint.iacr.org/2011/133.

[92] Oana Stan., Vincent Thouvenot., Aymen Boudguiga., Katarzyna Kapusta., Martin Zuber., and Renaud Sirdey. "A Secure Federated Learning: Analysis of Different Cryptographic Tools". In: *Proceedings of the 19th International Conference on Security and Cryptography - SECRYPT*. INSTICC. SciTePress, 2022, pp. 669–674. ISBN: 978-989-758-590-6. DOI: 10.5220/0011322700003283.

[93] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. *Efficient Public Key Encryption Based on Ideal Lattices*. Cryptology ePrint Archive, Paper 2009/285. 2009. URL: https://eprint.iacr.org/2009/285.

[94] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. "A Survey on Deep Transfer Learning". In: *International Conference on Artificial Neural Networks*. Springer, 2018, pp. 270–279.

[95] Yingjie Tian, Yuqi Zhang, and Haibin Zhang. "Recent Advances in Stochastic Gradient Descent in Deep Learning". In: *Mathematics* 11.3 (2023). ISSN: 2227-7390. DOI: 10.3390/math11030682.

[96] Daphné Trama, Pierre-Emmanuel Clet, Aymen Boudguiga, and Renaud Sirdey. "At Last! A Homomorphic AES Evaluation in Less than 30 Seconds by Means of TFHE". In: *IACR Cryptol. ePrint Arch.* (2023), p. 1020.

[97] Daphné Trama, Pierre-Emmanuel Clet, Aymen Boudguiga, and Renaud Sirdey. "Building Blocks For LSTM Homomorphic Evaluation With TFHE". In: *Cyber Security, Cryptology, and Machine Learning: 7th International Symposium, CSCML 2023, Be'er Sheva, Israel, June 29–30, 2023, Proceedings*. Be'er Sheva, Israel: Springer-Verlag, 2023, pp. 117–134. ISBN: 978-3-031-34670-5. DOI: 10.1007/978-3-031-34671-2_9. URL: https://doi.org/10.1007/978-3-031-34671-2_9.

[98] Jonathan Woodbridge, Hyrum S. Anderson, Anjum Ahuja, and Daniel Grant. "Predicting Domain Generation Algorithms with Long Short-Term Memory Networks". In: *CoRR* abs/1611.00791 (2016). arXiv: 1611.00791. URL: http://arxiv.org/abs/1611.00791.

[99] Pengtao Xie, Misha Bilenko, Tom Finley, Ran Gilad-Bachrach, Kristin E. Lauter, and Michael Naehrig. "Crypto-Nets: Neural Networks over Encrypted Data". In: *CoRR* (2014).

[100] Zhaomin Yang, Xiang Xie, Huajie Shen, Shiying Chen, and Jun Zhou. *TOTA: Fully Homomorphic Encryption with Smaller Parameters and Stronger Security*. Cryptology ePrint Archive, Report 2021/1347. 2021. URL: https://ia.cr/2021/1347.

[101] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. "How Transferable Are Features in Deep Neural Networks?" In: (2014). arXiv: 1411.1792.

[102] Xixun Yu, Zheng Yan, and Athanasios V Vasilakos. "A survey of verifiable computation". In: *Mobile Networks and Applications* 22 (2017), pp. 438–453.

[103] Matthew D. Zeiler and Rob Fergus. "Visualizing and Understanding Convolutional Networks". In: *European Conference on Computer Vision*. Springer, 2014, pp. 818–833.

[104] Yutong Zhong. "An Overview of RSA and OAEP Padding". In: *Highlights in Science, Engineering and Technology* 1 (2022). DOI: 10.54097/hset.v1i.431.

[105] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. "A Comprehensive Survey on Transfer Learning". In: *Proceedings of the IEEE* 109.1 (2020), pp. 43–76.

[106] Mingyu Zong and Bhaskar Krishnamachari. *a survey on GPT-3*. 2022. arXiv: 2212.00857 [cs.CL].

[107] Martin Zuber, Sergiu Carpov, and Renaud Sirdey. "Towards Real-Time Hidden Speaker Recognition by Means of Fully Homomorphic Encryption". In: *Information and Communications Security*. Ed. by Weizhi Meng, Dieter Gollmann, Christian D. Jensen, and Jianying Zhou. Cham: Springer International Publishing, 2020, pp. 403–421. ISBN: 978-3-030-61078-4.

[108] Martin Zuber and Renaud Sirdey. "Efficient homomorphic evaluation of k-NN classifiers". In: *Proc. Priv. Enhancing Technol.* 2021.2 (2021), pp. 111–129. DOI: 10.2478/popets-2021-0020.