THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES

ÉCOLE DOCTORALE Nº 601
Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Électronique
Spécialité : *Informatique*

Par
Samuel TAP

Construction de nouveaux outils pour un chiffrement homomorphe efficace

Constructing new tools for efficient homomorphic encryption

Thèse présentée et soutenue à Paris, le 19 décembre 2023
Unité de recherche : IRISA, UMR 6074

ACKNOWLEDGEMENT

Completing this PhD journey has been a profound and transformative experience, made possible by the support, guidance, and encouragement of many remarkable individuals.

First and foremost, I extend my deepest gratitude to my PhD advisor, Teddy Furron, for engaging in thought-provoking discussions and providing invaluable advice that has significantly shaped my academic and personal growth.

I am immensely thankful to my company supervisor, Pascal Paillier, for not only offering me the opportunity to pursue this PhD but also for being a constant source of inspiration through countless discussions and brainstorming sessions. Pascal's unwavering support, his eagerness to listen, and his thorough review of my work have been pivotal in my development. His cheerful disposition and motivation have been a beacon of light throughout this journey.

My heartfelt thanks go to my main co-authors: Jean-Baptiste Orfila, Damien Ligier, and Ilaria Chillotti. The journey we embarked on together, from the late nights working on paper submissions to our work-cations in the mountains, has been nothing short of extraordinary. Your advice, mentorship, and the camaraderie we shared have not only enriched my PhD experience but have also helped me grasp the essence of being a researcher.

Special thanks to Rand Hindi, the CEO of my company, for believing in me and my ideas, and for providing a perfect working environment. His interest in my work and results has been a source of great motivation.

I am grateful to Zama's Concrete team. The work we did togather has been instrumental in seeing the practical application of my research and has enriched my perspective by allowing me to interact with individuals from diverse backgrounds. A special mention goes to Quentin Bourgerie, Rudy Sicard, Jad Khatib, and Mayeul Debellabre, with whom I had the privilege to design our product and work closely on the optimizer. I also want to express my gratitude to Loris Bergerat for being one of my main co-authors; working with someone as dedicated and insightful as him has been a truly enjoyable and rewarding experience.

I owe a debt of gratitude to Patrick Bas and Timothée Pecatte, two individuals who provided invaluable assistance at key moments of my education. The former, one of my teachers, introduced me to the field of cryptography and ignited my passion for this domain. His support was crucial in keeping me motivated during the challenging times of my studies. The latter, a friend whose love for mathematics has been a source of inspiration and motivation, showed me the true beauty of math and encouraged me to share this passion with others.

Last but certainly not least, I thank my parents and my family, who have been my pillars of support. Their unwavering belief in me and their encouragement have been my guiding lights, nurturing my interests in mathematics and technology from an early age.

This thesis is not only a reflection of my hard work but also a testament to the contributions and faith of each individual mentioned above. To all of you, I am eternally grateful.

Table of Contents

List of Figures

List of Tables

Acronyms

$\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ type of atomic pattern introduced in Definition 34. 7, 10, 13, 177–183, 200–204

$\mathcal{A}^{(\mathbf{CGGI20})}$ type of atomic pattern introduced in Definition 33 inspired by [Chi+20a]. 9, 10, 127–129

$\mathcal{A}^{(\mathbf{CJP21})}$ type of atomic pattern introduced in Definition 30 inspired by [CJP21]. 7, 9, 10, 13, 113–117, 122, 124–135, 177–182, 200, 202–204, 217

$\mathcal{A}^{(\mathbf{GBA21})}$ type of atomic pattern introduced in Definition 32 inspired by [GBA21]. 7, 9, 10, 124–126, 177–179

$\mathcal{A}^{(\mathbf{KS\text{-}free})}$ type of atomic pattern composed of a dot product and a PBS. 10, 128, 129, 217

$\mathcal{A}^{(\mathbf{LMP21})}$ type of atomic pattern introduced in Section 6.2.3 inspired by [LMP21]. 7, 10, 179–181

AP atomic pattern. 9, 10, 13, 112–114, 121, 122, 124, 125, 127–132, 134, 135, 178, 181, 200–202

BSK bootstrapping key. 135

CRT Chinese Remainder Theorem. 11, 80, 81, 83–85, 177, 192, 197–199, 202, 204, 205, 239

DAG directed acyclic graph. 106, 108–113, 120, 121, 133–135

DP dot product. 10, 113, 114, 121, 127–132, 180, 181

FFT Fast Fourier Transform. 54, 89, 101–103, 132, 133

FHE fully homomorphic encryption. 29, 33, 80, 86, 96, 97, 106–114, 120, 121, 123, 129, 130, 132–135, 205, 237, 238

GCD Greatest Common Divisor. 131

GGSW generalized GSW [GSW13], see Definition 9. 46, 57, 66, 102, 171, 176

GLev collection of GLWE ciphertexts, see Definition 8. 46

GLWE General Learning With Errors. 17, 34, 35, 38–40, 42, 43, 45, 47, 49, 53–57, 66, 77, 90, 92, 94, 97–99, 101, 102, 106, 108, 109, 117, 133, 134, 143–145, 156, 157, 169, 176

KS key switching. 10, 109, 112, 114, 121, 126–129, 132, 134, 135, 171, 173, 176, 180, 181

KSK key switching key. 134, 135

LSB least significant bit(s). 79, 102, 132, 146, 186, 189, 191

LUT lookup table. 9, 10, 66, 69, 79, 80, 82–85, 120, 126, 169–171, 177–180, 189, 193, 198, 199, 202, 205, 237

LWE Learning With Errors. 33–35, 38, 39, 46, 49, 51, 53–55, 66, 79, 82–84, 89–95, 97, 106, 108, 109, 117, 132–134, 137, 143–145, 153, 156, 159, 160, 162, 169–171, 176, 187–189, 191, 199, 205, 208

MS modulus switching. 128

MSB most significant bit(s). 9, 78, 79, 102, 133, 146, 170, 177, 187, 189, 191, 194

PBS programmable bootstrapping. 7, 9, 63, 79–85, 109, 112–115, 120, 121, 124, 126, 127, 129–134, 169–171, 173, 175–177, 179, 180, 186, 187, 193, 194, 198, 199, 204, 205

RLWE ring learning with errors. 39, 54

WoP-PBS without padding programmable bootstrapping. 10, 84, 121, 169–171, 177–181, 203–205

Résumé en français

Dans notre vie de tous les jours, nous produisons une multitude de données à chaque fois que nous accédons à un service en ligne. Certaines sont partagées volontairement et d'autres à contrecœur. Ces données sont collectées et analysées en clair, ce qui menace la vie privée de l'utilisateur et empêche la collaboration entre entités travaillant sur des données sensibles. Un exemple classique en apprentissage automatique consiste à prédire si un cancer du sein est bénin ou non basé sur des tests médicaux. Bien que cela soit parfaitement faisable, en pratique, les hôpitaux ne peuvent pas utiliser une telle technologie car ils ne peuvent pas partager des données aussi sensibles avec un tiers non-fiable. Dans ce contexte, le *chiffrement complètement homomorphe* (*Fully Homomorphic Encryption*) apporte une lueur d'espoir en permettant d'effectuer des calculs sur des données chiffrées ce qui permet de les analyser et de les exploiter sans jamais y accéder en clair.

En 1978, Rivest, Adleman et Dertouzos [RAD78] furent les premiers à questionner l'existence des *privacy homomorphisms*, des fonctions qui peuvent être utilisées pour chiffrer des données tout en étant capables de réaliser des opérations sur celles-ci. Depuis, la réponse à cette question a été apportée : ces fonctions existent et un pan entier de la recherche en *cryptologie* se focalise sur ce sujet. Les schémas cryptographiques définis par ces fonctions sont appelés *schémas de chiffrement homomorphe* ou *schémas de chiffrement complètement homomorphe*. Dans ce contexte, *complètement* reflète la capacité de calculer n'importe quelle fonction sur une donnée chiffrée. Bien qu'il y ait différentes manières de créer des schémas complètement homomorphes, effectuer des calculs *efficacement* sur des données chiffrées est toujours un problème ouvert. En théorie, les schémas de ce type peuvent révolutionner notre manière d'interagir avec la technologie en garantissant la confidentialité de nos données tout en nous laissant la possibilité d'utiliser celles-ci dans le monde numérique. En pratique, ces techniques doivent être suffisamment efficaces pour ne pas gâcher l'expérience utilisateur et nous avons besoin d'outils permettant à un utilisateur profane de les utiliser. Dans cette thèse, notre considération première a été d'aider à combler l'écart entre ces découvertes scientifiques innovantes et leurs utilisations en pratique. Pour cela, nous avons introduit de nouvelles techniques qui surpassent les techniques à l'état de l'art ainsi que de nouveaux outils qui simplifient l'utilisation en pratique de cette nouvelle technologie.

Chiffrement Complètement Homomorphe. Les premiers schémas de chiffrement homomorphe étaient uniquement capables d'évaluer un certain type d'opération sur des chiffrés. Par exemple, en utilisant la primitive RSA [RSA78], il est possible de calculer un nombre infini de multiplications modulaires. La question de l'existence des schémas complètement homomorphes a longtemps été un problème ouvert et une réponse fut finalement apportée par Gentry en 2009 lorsqu'il publia la première instantiation d'un schéma complètement homomorphe [Gen09].

Une caractéristique commune au schéma introduit par Gentry et à tous les autres schémas est que les chiffrés contiennent de l'aléatoire appelé *bruit*. La vaste majorité des opérations homomorphes fait augmenter le bruit et s'il n'est pas contrôlé, le bruit peut compromettre le message, ce qui entraîne des résultats incorrects après le déchiffrement. Ce phénomène limite le nombre d'opérations qui peuvent être effectuées sur des chiffrés. L'idée révolutionnaire de Gentry qui rend le chiffrement complètement homomorphe possible est une technique appelée *bootstrapping* qui permet de réduire le bruit en utilisant uniquement des informations publiques, appelées clef de *bootstrapping*. De cette manière, il n'y a plus de limitation sur le nombre maximal d'opérations qui peuvent être effectuées et le schéma devient *complètement* homomorphe. Le principal défaut de la méthode introduite par Gentry est que le bootstrap est très lent (plusieurs minutes) et que la clef de *bootstrapping* est très grande (plusieurs gigabytes). Au cours de ces dix dernières années, des alternatives au schéma de Gentry ont été introduites mais le bootstrap reste le goulot d'étranglement en terme de temps d'exécution.

Pour concevoir un schéma cryptographique, il faut garantir que pour retrouver le message, un attaquant doit résoudre un problème mathématique difficile et suffisamment étudié. En évaluant le coût pour le résoudre à l'aide des meilleures techniques existantes, il est possible d'estimer la sécurité du nouveau schéma, c'est-à-dire le nombre d'opérations (additions, multiplications, etc.) qu'il faut réaliser. De nos jours, les schémas complètement homomorphes les plus utilisés sont BGV [BGV12], B/FV [Bra12; FV12], HEAAN [Che+17], GSW [GSW13], FHEW [DM15] et TFHE [Chi+20a]. Bien que le *bootstrapping* soit possible dans tous ces schémas, la plupart l'évite le plus possible car il reste une opération très coûteuse. Les schémas BGV, B/FV et HEAAN ont choisi d'adopter une approche *à niveaux*, ce qui consiste à choisir des paramètres suffisamment grands pour tolérer le bruit produit durant le calcul sans que cela modifie le résultat. Ainsi, il faut connaître la totalité du calcul ou circuit pour choisir des paramètres qui donneront une évaluation correcte. Ces schémas profitent de *l'encoding SIMD* [SV14] pour remplir un chiffré avec plusieurs messages et réaliser des opérations en parallèle sur tous ces messages en même temps. L'approche à niveaux est particulièrement pratique quand le même circuit doit être évalué sur plusieurs messages et si celui-ci n'est pas trop profond.

Cet état de fait a changé en 2015 lorsque Ducas et Micciancio ont introduit FHEW capable de réaliser le *bootstrapping* en moins d'une seconde pour de petits entiers. En 2016, Chillotti, Gama, Georgieva and Izabachène ont publié TFHE [Chi+16a], une version améliorée de FHEW avec un *bootstrapping* très efficace comparé aux autres schémas complètement homomorphes (dizaine de millisecondes). Le *bootstrapping* de FHEW et TFHE n'est pas seulement rapide mais il est aussi programmable. Cela signifie qu'il est possible d'évaluer une fonction arbitraire sur un chiffré tout en réduisant le bruit. Depuis la publication de TFHE, de nombreux papiers scientifiques proposent des alternatives ou des améliorations au *bootstrapping* mais il reste des problèmes ouverts. Bien que cette opération soit particulièrement rapide, elle reste l'opération la plus coûteuse de TFHE. Dans cette thèse, nous nous focalisons particulièrement sur TFHE et nous introduisons de nombreuses techniques permettant de l'améliorer en enlevant certaines de ces contraintes ainsi que des techniques alternatives pour réduire le bruit qui sont plus efficaces dans certains contextes.

Un des problèmes majeurs des schémas complètement homomorphes est de trouver des paramètres cryptographiques efficaces pour un cas d'utilisation précis. Ces paramètres doivent à la fois garantir la sécurité des données et être assez petits pour permettre une exécution efficace en terme de temps d'exécution, de mémoire ou de consommation énergétique. Résoudre ce problème est fondamental si nous voulons que ces schémas soit adoptés à grande échelle. Nous décrivons dans cette thèse une méthode d'optimisation qui apporte une solution concrète à ce problème.

Bootstrapping de TFHE. TFHE [Chi+16a] est particulièrement intéressant parce qu'il offre une technique de *bootstrapping* programmable. Malheureusement, en pratique, celle-ci souffre de plusieurs restrictions. Premièrement, bien qu'elle soit rapide, cette technique de *bootstrapping* reste le goulot d'étranglement en terme de temps d'exécution. En effet, le bootstrap prend en entrée un unique petit message (moins de 8 bits) et plus cet entier est grand, plus le bootstrap est coûteux. De plus, supporter des fonctions arbitraires lors du *bootstrapping* ajoute une contrainte supplémentaire : le bit de poids fort d'un message doit être connu. Cette contrainte nous empêche la construction d'une arithmétique modulaire efficace car nous devons nous assurer que ce bit n'est pas compromis durant tout le calcul.

Nos Contributions. Le premier schéma complètement homomorphe a été inventé il y a presque quinze ans et il semble être une solution adaptée pour garantir la confidentialité des données tout en permettant à un tiers de les exploiter. Toutefois, ces techniques n'ont toujours pas été adoptées par l'industrie et les utilisateurs. Dans cette thèse, nous nous sommes posé les questions suivantes :

Quels sont les freins à l'adoption des schémas complètement homomorphes ?
Est-ce que créer des algorithmes cryptographiques efficaces est suffisant pour garantir l'adoption de cette technologie ?

Plus généralement, nous nous sommes demandé :

Comment pouvons-nous rendre les schémas complètement homomorphes plus pratiques ?

Cette thèse est un résumé de nos tentatives durant ces trois dernières années pour trouver des réponses satisfaisantes à ces questions.

Dans le chapitre 3, nous introduisons les premiers composants de notre méthode d'optimisation que sont l'oracle de sécurité et les modèles de bruit. Un oracle de sécurité estime la variance de bruit minimale à utiliser au moment du chiffrement pour satisfaire une certaine sécurité. Un modèle de bruit est une collection de formules de bruit, des fonctions prédisant la distribution du bruit après des calculs. Ce modèle est utilisé pour estimer la distribution du bruit tout au long du calcul, ce qui est crucial pour garantir l'exactitude de celui-ci. Nous expliquons aussi comment corriger une formule théorique en prenant en compte l'implémentation en pratique des algorithmes.

Dans le chapitre 4, nous introduisons notre méthode d'optimisation qui permet de sélectionner automatiquement les meilleurs paramètres pour exécuter homomorphiquement un graphe de calcul donné. Cette méthode résoud un des facteurs bloquants majeurs à l'adoption du chiffrement homomorphe. Notre méthode d'optimisation modélise le problème de sélection des paramètres comme un problème d'optimisation que nous pouvons simplifier et résoudre en utilisant des techniques classiques d'optimisation, comme le *branch-and-bound*.

Grâce à notre méthode d'optimisation, nous pouvons aussi comparer des algorithmes homomorphes différents qui réalisent la même opération sur les messages en satisfaisant des compromis différents entre le bruit et le temps d'exécution. De nombreuses comparaisons sont présentées dans cette thèse et mettent en lumière la relation entre la taille des messages et le temps d'exécution. En particulier, à l'issue de ce travail, il est clair qu'il est plus efficace de représenter un grand entier sur plusieurs chiffrés plutôt que sur un seul.

Dans le chapitre 5, nous introduisons de nombreux nouveaux algorithmes homomorphes. Nous avons étudié l'impact de chacun de ces algorithmes sur le bruit et sur le coût du calcul. Tout d'abord, nous avons généralisé le bootstrap de TFHE pour qu'il soit capable d'évaluer de multiples fonctions en même temps, sans conséquences sur le bruit ou sur le coût. Ensuite, nous expliquons comment évaluer une fonction sur une entrée arrondie. Finalement, nous avons étudié la multiplication de BFV [Bra12; FV12] et nous l'avons incluse à l'arsenal des opérateurs disponibles dans TFHE. Grâce à cet algorithme, nous obtenons une multiplication efficace entre chiffrés et sans avoir recourt à un *bootstrapping*.

Dans le chapitre 6, nous donnons trois nouveaux algorithmes permettant de réaliser un *bootstrapping* programmable sans avoir besoin de connaître le bit de poids fort du message. Les deux premières versions sont construites grâce à la multiplication précédemment introduite et sont parallélisables. Grâce à ces deux algorithmes, nous pouvons décomposer homomorphiquement un message en plusieurs morceaux. En combinant ces nouveaux algorithmes avec ceux de l'état de l'art, nous expliquons comment réduire le bruit et comment appliquer une fonction sur des chiffrés de grande précision. Le troisième *bootstrapping* introduit dans ce chapitre est capable d'appliquer une fonction multivariée sur des chiffrés. Cette méthode permet d'appliquer des fonctions arbitraires sur des grands messages représentés sur plusieurs chiffrés. Elle est particulièrement efficace sur les entiers de grande précision (plus de 8 bits).

Dans le chapitre 7, nous étudions comment créer des arithmétiques entières efficaces avec TFHE. Tout d'abord, nous généralisons la technique utilisée pour évaluer des circuits booléens avec TFHE. Pour cela, nous utilisons les algorithmes introduits précédemment, en particulier la multiplication entre chiffrés. Grâce à cette nouvelle méthode, il est possible de calculer efficacement des opérations sur des entiers ou des entiers modulaires. Finalement, en utilisant le dernier *bootstrapping* introduit dans le chapitre précédent, nous décrivons une arithmétique efficace sur des grands entiers représentés de différentes manières. Nous illustrons notre propos avec de nombreuses mesures de temps d'exécution des principales opérations possibles sur les entiers.

Dans le chapitre 8, nous introduisons deux nouveaux types de clefs secrètes ainsi que de nouveaux algorithmes adaptés à celles-ci. Les clefs partielles sont particulièrement utiles pour les entiers de grande précision (plus de 8 bits). Le deuxième type de clef appelé clefs partagées permet de réduire le coût de conversion entre différents types de chiffrés. Ces conversions étant fondamentales au bon fonctionnement de TFHE, ce nouveau type de clef est particulièrement intéressant.

Tous les algorithmes et leurs variantes sont testés et leur temps d'exécution est mesuré en utilisant notre méthode d'optimisation précédemment décrite.

Publications. La plupart du contenu des chapitres 5, 6 et 7 a été publié à la conférence *Asiacrypt 2021* [Chi+21]. La méthode d'optimisation introduite dans le chapitre 4 ainsi que le reste des chapitres 6 et 7 a été publié dans *Journal of Cryptology* [Ber+23a]. Les nouveaux types de clefs secrètes introduits dans le chapitre 8 seront soumis prochainement. En plus du travail présenté dans ce manuscrit, nous avons publié un papier focalisé sur la librairie *Concrete* à *WAHC 2021* [Chi+20b] ainsi qu'un papier à *WAHC 2023* [Dah+23]. Trois demandes de brevet ont été déposées sur des sujets détaillés dans cette thèse.

Conclusions. Le chiffrement complètement homomorphe est à un stade embryonnaire de son développement. Au début de la thèse, il y avait relativement peu de projets focalisés sur la création d'outils permettant à un profane d'utiliser cette technologie en pratique. En outre, ces projets sont souvent des initiatives individuelles et ne sont pas maintenus dans le temps. Pour utiliser ces outils, des connaissances techniques en chiffrement homomorphe sont nécessaires pour obtenir de bonnes performances et garantir l'exactitude d'un calcul et la sécurité des données. Tant qu'il n'existera pas d'outils mis à jour régulièrement capables de garantir de lui-même la sélection de paramètres sécurisés, efficaces et qui permettent une exécution exacte, le chiffrement homomorphe ne se démocratisera pas. La méthode d'optimisation que nous introduisons dans cette thèse est une des pierres angulaires d'un tel outil. Cette méthode est déjà intégrée dans un projet qui a pour but de transformer un graphe de calcul en un exécutable capable de réaliser des opérations sur des données chiffrées.

Le calcul utilisant le chiffrement homomorphe est plus lent qu'un calcul classique. En pratique, cela signifie qu'il n'est pas encore envisageable d'utiliser cette technologie dans des scénarios où il est nécessaire d'avoir une réponse issue d'un calcul complexe en temps réel. Dans cette thèse, nous introduisons de nombreuses nouvelles techniques avec de meilleures performances que celles de l'état de l'art. Avec nos contributions, l'écart entre l'exécution de calcul sur données chiffrées et sur données en clair n'a jamais été aussi faible.

Tout le travail réalisé durant cette thèse contribue à rendre le chiffrement homomorphe plus pratique et accessible aux utilisateurs profanes. Nous espérons sincèrement que ces résultats seront utilisés pour accélérer l'adoption généralisée du chiffrement homomorphe qui garantira la confidentialité de nos données tout en nous permettant de profiter de tout les services et les avantages que notre ère digitale peut nous offrir.

Chapter 1

Introduction

In our everyday life, we leave a trail of data whenever we access online services. Some are given voluntarily and others reluctantly. Those data are collected and analyzed in the clear which leads to major threats on the user's privacy and prevents collaborations between entities working on sensitive data. For instance, an emblematic machine learning use case consists in predicting whether a breast cancer is benign or malignant given some medical analysis. While this is totally doable, hospitals cannot use such technology as they can not share the patient information to a untrusted third-party. In this context, *Fully Homomorphic Encryption* brings a light of hope by enabling computation over encrypted data which removes the need to access data in the clear to analyze it and exploit it.

In 1978, Rivest, Adleman and Dertouzos [RAD78] were the first to question the existence of *privacy homomorphisms*, functions that can be used to encrypt some data while preserving the ability of performing operations over them. Since then, this question has been answered, such functions exist and an entire area of research in cryptology focuses on this subject. Cryptographic schemes defined by such privacy homomorphisms are called *Homomorphic Encryption* schemes (HE) or *Fully Homomorphic Encryption* schemes (FHE). *Fully* stands for the ability to compute arbitrary functions over encrypted data. While there are several ways to build an FHE scheme, the question of efficiently computing over encrypted data is still an open problem. In theory, using such schemes can revolutionize our way to interact with the technology by guaranteeing the privacy of our data while still being able to use them in the digital world. In practice, we need these techniques to be sufficiently efficient to not hinder the user experience and we need a set of tools to make it accessible to non-expert users. In this thesis, our primary focus was to help filling the gap between these innovative scientific discoveries and their practical use. To do so, we introduced several new techniques that are more efficient than the state-of-the-art techniques and some new tools to automatize the usage in practice of this new technology.

Fully Homomorphic Encryption. The first HE schemes were only able to perform a unique operation over ciphertexts. For instance, with the well-known RSA primitive [RSA78], we can only compute an unbounded number of modular multiplications. The question of the existence of FHE schemes has been an open question for a long time and was finally answered by Gentry in 2009 when he published the first instantiation of an FHE scheme [Gen09].

A common feature in Gentry's original cryptosystem and in all subsequent FHE schemes is that ciphertexts contain some randomness called *noise*. The vast majority of homomorphic operations make this noise grow and if not controlled, the noise can compromise the encrypted plaintext, which induces incorrect results at decryption time. This fact inherently limits the number of operations that can be performed on ciphertexts. The groundbreaking idea of Gentry, which made FHE possible, was a technique called *bootstrapping* enabling to reduce the noise when needed using only public information called *bootstrapping key*. The bootstrapping eliminates the limitation on the maximal number of operations that can be performed and the scheme becomes *fully* homomorphic. The main drawback of Gentry's method is that the bootstrap is very slow (tens of minutes) and the public material needed is very big (several gigabytes). Over the last ten years, alternatives to Gentry's scheme were introduced but the bottleneck in term of execution time of every single one of them was still the *bootstrap*.

When designing a cryptographic scheme, one needs to guarantee that to recover the message, an attacker should solve a well-known hard mathematical problem. By estimating the cost of executing the state-of-the-art techniques to solve it, one can estimate the security of its new scheme i.e., the number of operations (additions, multiplications, etc.) to execute. Nowadays, the most practical FHE schemes are based on the hardness assumption called *Learning With Errors* (LWE), introduced by Regev in 2005 [Reg05], and on its *ring* variant (RLWE) [Ste+09; LPR10].

The (R)LWE-based schemes mainly used are BGV [BGV12], B/FV [Bra12; FV12], HEAAN [Che+17], GSW [GSW13], FHEW [DM15] and TFHE [Chi+20a]. Even if bootstrapping is possible for all these schemes, most of them actually avoid it because the technique remains a bottleneck. In particular, this is the case of BGV, B/FV and CKKS. These schemes adopt a *leveled approach*, which consists in choosing parameters that are large enough to tolerate all the noise produced during the computation. The circuit needs to be known beforehand in order to choose parameters that will lead to a correct evaluation. These schemes take advantage of *SIMD encoding* [SV14] to pack many messages in a single ciphertext and perform the homomorphic evaluations in parallel on all of these messages at the same time, and they naturally perform homomorphic multiplications between ciphertexts. The *leveled approach* is very convenient when multiple inputs have to be evaluated with the same circuit as long as the evaluated circuit is not too deep.

This state of affairs changed in 2015 when Ducas and Micciancio introduced the FHEW cryptosystem [DM15] achieving bootstrapping in less than a second for small integer messages. In 2016, Chillotti, Gama, Georgieva and Izabachène introduced TFHE [Chi+16a], an improved version of FHEW with a very fast bootstrapping operation compared to the other FHE schemes (tens of milliseconds). In addition to this breakthrough in terms of efficiency, the bootstrap of FHEW and TFHE are programmable. It means that the bootstrap can be used to evaluate an arbitrary function over a ciphertext while reducing the noise. Since the publication of TFHE, lots of research have been carried out to improve it but open problems are still pending. While being fast, the bootstrap remains the costliest operator of TFHE. In this thesis, we focus primarily on this scheme and introduce a wide range of techniques to improve this bootstrap by removing some of its constraints. We also provide new bootstrapping techniques that are more efficient in some contexts.

One of the main problems of any FHE scheme remains to *find good cryptographic parameters* for a given use case. Such parameters need to both be secure and make the whole homomorphic processing as efficient as possible, in terms of either computational cost, memory or hardware resources. Solving this problem is fundamental if we aim for a large scale adoption of FHE schemes. In this thesis, we introduced an optimization framework that solves this problem.

TFHE's PBS. TFHE [Chi+16a] is particularly interesting because it offers an efficient bootstrapping technique that is programmable. Unfortunately, in practice, the bootstrapping operation still has some restrictions.

First and foremost, while being fast, the bootstrapping technique remains the main bottleneck in terms of execution time. The bootstrap takes as input a single ciphertext encrypting a small integer message (say at most 8 bits). Unfortunately, the bigger the integer, the less efficient the bootstrap. Supporting arbitrary functions in the bootstrap adds another restriction to TFHE: the most significant bit of a message must be known which prevents us from having an efficient modular arithmetic and forces us to always make sure that it is not overwritten during a computation.

Our contributions. The first FHE scheme was invented nearly fifteen years ago and it seems to be the perfect solution to guarantee the privacy of data while still permitting a third party to exploit it. However, there is still no mass adoption of FHE by companies and users. In this thesis, we asked ourselves the following questions:

What are the missing tools/technologies that hinder the adoption of FHE?

Are efficient cryptographic algorithms enough to lead to a widespread adoption?

and more generally:

How can we make FHE more practical?

This thesis is a summary of the attempts made during the course of the last three years to find satisfying answers to these questions.

The manuscript starts with a thorough state-of-the-art on FHE and more specifically on TFHE in Chapter 2. We recall some notions on the security of the LWE and GLWE problems, introduce the different types of ciphertexts and the main building blocks of TFHE and its variants. We also recall the different attempts at optimization for FHE. In Chapter 3, we focus on the noise, a key concept in FHE and explain how it is related to the correctness of a computation. In Chapter 4, we introduced our optimization framework for FHE that automatically selects the best LWE or GLWE instances for a given use case and, more generally, sets every degree of freedom available in any TFHE algorithm. In Chapters 5 and 6, we detail the new algorithms we found that remove some of the existing limitations of TFHE and improve the state of the art. In Chapter 7, we explain how to use the different techniques introduced in the previous chapters in combination with the state-of-the-art techniques to build efficient arithmetics over ciphertexts. In Chapter 8, we introduce two new types of secret keys and several new algorithms leveraging these new keys which lead to significant improvements in term of execution time. Finally, we summarize in the conclusion our main contributions and expose the new questions that have arisen during the course of this work. This thesis contains an annex where most of the noise analysis is presented in detail for both state-of-the-art algorithms and for the new algorithms introduced in this work.

Publications. Most of the contents of Chapter 5 and parts of Chapters 6 and 7 were published at the conference *Asiacrypt 2021* [Chi+21]. The optimization framework explained in Chapter 4 and parts of Chapters 6 and 7 have been published in *Journal of Cryptology* [Ber+23a]. The new types of secret keys introduced in Chapter 8 are soon to be submitted. In addition to the work presented in this thesis, we published a paper focusing on the Concrete Library at the workshop *WAHC 2021* [Chi+20b] and a paper on Threshold-FHE at *WAHC 2023* [Dah+23]. During the course of this work, three patent applications were filled on subjects detailed in this thesis.

Learnings. FHE is at an early stage of development. At the beginning of this thesis, there were already a few projects focusing on giving a non-expert user the ability to use FHE in real use cases. Unfortunately, these projects are often private initiatives and not maintained in the long run. To use those tools, some knowledge on FHE is needed to achieve secure, efficient and correct computations. As long as there are no maintained tools that target non-FHE experts and that guarantee efficiency, security and correctness, we will not see a widespread adoption of FHE. The optimization framework we introduce in this thesis is a corner stone of such a tool. It is already integrated in a bigger project[1] that takes as input a crypto-free graph of computation and outputs a binary executable ready to be run on encrypted data.

Furthermore, FHE computation is slower than classical computation. It means that for now, FHE must be excluded from pipelines that require real-time responses. In this thesis, we introduce lots of new algorithms that improve the state of the art in terms of efficiency. With these new techniques, the gap between cleartext computation and encrypted computation has become smaller than ever.

All the work carried out in this thesis contributes to make FHE more practical and accessible to non-expert users. We sincerely hope that our results will be used to speed up the adoption of FHE which will eventually guarantee the privacy of our data while allowing us to benefit from everything our digital era has to offer.

1. `https://github.com/zama-ai/concrete`

Chapter 2

Preliminaries

In this chapter, we provide a summary of the state of the art needed to put our research into perspective.

First, we give some details on the GLWE and the LWE problems. The security of most of the FHE schemes in use today relies on the hardness of those mathematical problems. We present the main attacks against those problems and introduce the lattice estimator, a tool actively maintained by the community to estimate the security of lattice-based cryptosystems.

Then, we introduce the different types of ciphertexts that are used by every FHE scheme based on (G)LWE. It includes (G)LWE ciphertexts, (G)Lev ciphertexts and (G)GSW ciphertexts.

Next, we review the main building blocks of TFHE and some of their recent improvements. We cover basic operations (addition, multiplication by an integer, etc.) and more complex algorithms (several key switch variants, the Programmable Bootstrap, etc.). Some of the limitations of TFHE are also discussed.

Then, we explain the different encoding methods compatible with TFHE, covering the traditional encoding and two more recent encodings (CRT and radix encodings).

Finally, we address the state of the art of optimization techniques for FHE, the goals of the different approaches and explain how TFHE stands out compared to other FHE schemes.

2.1 The Security of FHE

The most used FHE schemes rely on the hardness of the Learning With Errors problem and its variants introduced two decades ago by Regev in [Reg05] and later extended in [Reg09]. Some FHE schemes rely on other problems. For instance, [Dij+10] relies on the Approximate GCD problem [How01] and [LTV12; Bon+22] rely on the NTRU problem. These schemes are not discussed in this thesis.

First, we introduce the *Learning With Errors problem* and the *General Learning With Errors problem.* We then provide a brief description of the main attacks on those problems and finally describe the lattice estimator, the reference tool to estimate the security of lattice-based cryptosystems.

2.1.1 (G)LWE Problems

In 2005, Regev introduced the LWE problem and a simple cryptosystem based on it. Informally, given an LWE instance, it is believed to be hard to extract the key. An LWE instance is a system of noisy linear equations i.e., linear equations perturbated with a small randomness called noise. Without the noise, those linear equations can be easily solved using linear algebra techniques, for instance Gaussian elimination.

In the definition of the LWE problem below, $\mathbb{Z}_q$ refers to the ring $\mathbb{Z}/q\mathbb{Z}$.

Definition 1 (Learning With Errors (LWE)) *Let $n \in \mathbb{N}$ be the LWE dimension. Let $q \in \mathbb{N}$ be the ciphertext modulus. Let $\vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n$ be a secret, where for all $0 \leq i < n, s_i$ is sampled from a given distribution $\mathcal{D}(\mathbb{Z}_q)$, and let χ be an error distribution. We define $(\vec{a}, b = \sum_{i=0}^{n-1} a_i \cdot s_i + e) \in \mathbb{Z}_q^{n+1}$ to be a sample from the learning with errors distribution $\mathsf{LWE}_{q,n,\chi,\mathcal{D}(\mathbb{Z}_q)}$, such that $\vec{a} = (a_0, \ldots, a_{n-1}) \hookleftarrow \mathcal{U}(\mathbb{Z}_q)^n$, meaning that all $\{a_i\}_{i \in [\![0,n-1]\!]}$ are sampled uniformly from $\mathbb{Z}_q$, and the error or noise $e \in \mathbb{Z}_q$ is sampled from χ.*

The decisional $\mathsf{LWE}_{q,n,\chi,\mathcal{D}(\mathbb{Z}_q)}$ *problem [Reg05] consists in distinguishing independent samples from $\mathcal{U}(\mathbb{Z}_q)^{n+1}$ from the same amount of samples from $\mathsf{LWE}_{q,n,\chi,\mathcal{D}(\mathbb{Z}_q)}$.*

The search problem *consists in finding $\vec{s}$ given an arbitrary number of samples from the learning with error distribution $\mathsf{LWE}_{q,n,\chi,\mathcal{D}(\mathbb{Z}_q)}$.*

In [Reg05], Regev shows that both the decisional problem and the search problem are reducible to one another. The hardness of each of those problems depends on the parameters (q, n) and on the noise distribution χ and the secret key distribution $\mathcal{D}(\mathbb{Z}_q)$. The hardness of an LWE instance is described by a value λ which is called the *security level.* For an attacker to break an LWE instance with a security level λ, he should perform at least 2^λ operations. More precisely, breaking an LWE instance with a security level λ should take as many (or more) computational resources than those required to break a blockcipher with a λ-bit key[1]. In practice, $\lambda = 128$ is the default value to guarantee long term security of a particular instance.

1. `https://csrc.nist.gov/projects/post-quantum-cryptography/`

Several variants of the LWE problem were introduced in the following years [Ste+09; LPR10; BGV12; LS15]. Below, we defined the General Learning With Errors (GLWE) problem [BGV12; LS15]. We use the ring $\mathfrak{R}_{q,N}$ defined as $\mathfrak{R}_{q,N} = \mathbb{Z}_q[X] / \left\langle X^N + 1 \right\rangle$ with N a power of two.

Definition 2 (General Learning With Errors (GLWE)) *Let $N \in \mathbb{N}$ be a polynomial size (chosen as a power of 2). Let $k \in \mathbb{N}$ be a GLWE dimension. Let $q \in \mathbb{N}$ be a ciphertext modulus. Let $\vec{S} = (S_0, \cdots, S_{k-1}) \in \mathfrak{R}_{q,N}^k$ be a secret, where $S_i = \sum_{j=0}^{N-1} s_{i,j} X^j$ is sampled from a given distribution $\mathcal{D}(\mathfrak{R}_{q,N})$ for all $0 \leq i < k$, and let χ be an error distribution. We define $(\vec{A}, B = \sum_{i=0}^{k} A_i \cdot S_i + E) \in \mathfrak{R}_{q,N}^{k+1}$ to be a sample from the* general learning with errors distribution $\mathsf{GLWE}_{q,N,k,\chi,\mathcal{D}(\mathfrak{R}_{q,N})}$*, such that $\vec{A} = (A_0, \ldots, A_{k-1}) \hookleftarrow \mathcal{U}(\mathfrak{R}_{q,N})^k$, meaning that all the coefficients of A_i are sampled uniformly from $\mathbb{Z}_q$, and the error (noise) polynomial $E \in \mathfrak{R}_{q,N}$ is such that all the coefficients are sampled from χ.*

The decisional $\mathsf{GLWE}_{q,N,k,\chi,\mathcal{D}(\mathfrak{R}_{q,N})}$ problem [LS15; BGV12] consists in distinguishing m independent samples from $\mathcal{U}(\mathfrak{R}_{q,N})^{k+1}$ from the same amount of samples from $\mathsf{GLWE}_{q,N,k,\chi,\mathcal{D}(\mathfrak{R}_{q,N})}$, where $\vec{S} \in \mathfrak{R}_{q,N}^k$ follows the distribution $\mathcal{D}$.

The search problem consists in finding $\vec{s}$ given an arbitrary number of samples from the learning with error distribution $\mathsf{GLWE}_{q,N,k,\chi,\mathcal{D}(\mathfrak{R}_{q,N})}$.

If we choose $N = 1$ and $k = n$, the GLWE distribution $\mathsf{GLWE}_{N,k,\chi,\mathcal{D}(\mathfrak{R}_{q,N})}$ is the same as the LWE distribution $\mathsf{LWE}_{n,\chi,\mathcal{D}(\mathbb{Z}_q)}$. If we choose $N > 1$ and $k = 1$, the GLWE distribution is called the RLWE distribution [Ste+09; LPR10]. The GLWE problem can be seen as a generalization of both the LWE problem and the RLWE problem.

In general, the secret key distribution $\mathcal{D}(\mathfrak{R}_{q,N})$ is such that the polynomial coefficients are usually either sampled from a uniform binary distribution, a uniform ternary distribution or a Gaussian distribution ([Bou+23; App+09]).

In the definition below, we focus on more specific secret keys.

Definition 3 (Secret Keys With Fixed Hamming Weight (FHW)) *A fixed Hamming weight (FHW) binary (resp. ternary) GLWE secret key of hamming weight $h \in \mathbb{N}$ is a GLWE secret key such that its polynomial coefficients are in $\{0, 1\}$ (resp. $\{-1, 0, 1\}$) and contains exactly h non-zero coefficients. We note these two distributions $\mathcal{FHW}(h, \{0, 1\})$ and $\mathcal{FHW}(h, \{-1, 0, 1\})$ respectively. Such keys come along with public*

`post-quantum-cryptography-standardization/evaluation-criteria/`
`security-(evaluation-criteria)`

knowledge: the dimension $k \in \mathbb{N}$, the polynomial ring $\mathfrak{R}_{q,N}$ (including the polynomial size $N \in \mathbb{N}$), the distribution (binary or ternary), the hamming weight h.

This type of secret keys is in use in FHE schemes such as CKKS [Che+17], because it offers a smaller value for the worst-case noise growth. Table 2.1 summarizes public knowledge for different secret key types used in FHE.

Key Type	**Size**	**Ring**	**Distribution**	**Hamming Weight**
Uniform Binary	k	$\mathfrak{R}_{q,N}$	$\mathcal{U}(\{0,1\})$	unknown
Uniform Ternary	k	$\mathfrak{R}_{q,N}$	$\mathcal{U}(\{-1,0,1\})$	unknown
Gaussian	k	$\mathfrak{R}_{q,N}$	$\mathcal{N}(\mu,\sigma^2)$	unknown
Small Uniform	k	$\mathfrak{R}_{q,N}$	$\mathcal{U}(\mathbb{Z}_\alpha)$	unknown
Uniform	k	$\mathfrak{R}_{q,N}$	$\mathcal{U}(\mathbb{Z}_q)$	unknown
FHW Binary	k	$\mathfrak{R}_{q,N}$	$\mathcal{FHW}(h,\{0,1\})$	h
FHW Ternary	k	$\mathfrak{R}_{q,N}$	$\mathcal{FHW}(h,\{-1,0,1\})$	h

Table 2.1: Comparison between secret key types in terms of public knowledge.

2.1.2 Attacks on (G)LWE

We already introduced the LWE problem (Definition 1) and the GLWE problem (Definition 2) and we noticed that the hardness of the problem depends on some parameters (q, k, N) and some distributions. In this section, we give an overview of the main attacks currently known on LWE and GLWE.

Attacks on LWE.

We recall the attacks against LWE which are important to consider in the selection of secure parameters. These attacks and the associated references are used in the lattice estimator [APS15].

The first kind of attacks is called LWE primal attacks and was first formulated in [Alk+16a] and later studied and verified in [Alb+17b; Dac+20b; PV21]. It consists of using lattice reduction to solve an instance of uSVP (unique Shortest Vector Problem) generated from LWE samples. The most common way to perform this reduction is to use the BKZ algorithm [SE94] to reduce a lattice basis by using an SVP (Shortest Vector Problem) oracle. So, according to this attack, the security of an LWE instance is based on the cost of lattice reduction for solving uSVP. In [Alk+16a], the authors propose to analyze the hardness of RLWE as an LWE problem. All the research on this attack tend to find the best cost of solving uSVP in order to find the closest model of security for LWE and by extension for RLWE.

The second type of attack is the LWE dual attacks. It is explained in [MR09b]. It consists of solving an instance of the SIS (Short Integer Solution) problem in the dual lattice of the lattice formed by LWE samples. It means that the security of an LWE instance is based on the cost of solving the SIS problem. In [Alb17a], Albrecht ported the classical dual attack to a small secret key setting and in [Che+19b], the authors introduced a new hybrid of dual and meet-in-the-middle (MITM) attack.

The third kind of attacks is the coded-BKW attacks, which are based on the algorithm BKW (Blum, Kalai and Wasserman [BKW03]). This attack is explained in [GJS15a; KF15b]. The BKW algorithm is a recursive dimension reduction for LWE instances. In [GJS15a], the authors make use of these attacks against RLWE.

Attacks on RLWE/GLWE.

In the last decade, some attacks (for example [CDW17; PHS19; BR20; Ber+23c]) tried to take advantage of the polynomial ring structure of RLWE and GLWE to solve the id-SVP (ideal-Shortest Vector Problem). However, none of these attacks is as efficient as the LWE attacks presented before. It means that when one wants to efficiently attack a GLWE instance, they actually use LWE attacks so the security of $\mathsf{GLWE}_{q,N,k,\chi,\mathcal{D}(\mathfrak{R}_{q,N})}$ is estimated from the security of $\mathsf{LWE}_{q,k\cdot N,\chi,\mathcal{D}(\mathbb{Z}_q)}$.

Other Attacks.

Some other attacks are not based on a reduction to a classical problem but on the leakage of some fraction of the coordinates of the NTT transform of the RLWE secret. It is the case of the article [Dac+18] which proposes a more direct attack against RLWE under this leakage assumption.

2.1.3 Lattice Estimator

The lattice estimator is an initiative launched by Martin R. Albrecht, Rachel Player and Sam Scott [APS15]. The goal of the paper was to give an extensive survey of the known attacks on LWE/GLWE. Parts of the paper are outdated as a significant amount of work has been ongoing in the field since its publication. A more up-to-date version is available in Player's thesis [Pla18].

One of the contributions of this work was to deliver a tool written in *Sage* to help researchers estimate the security of particular LWE or GLWE instances using state of the art attacks. Since the publication of the paper, several attacks were improved and added to the tool[2].

The current version of the lattice estimator takes into account the attacks described in Section 2.1.2 and an additional attack leveraging Gröbner bases: Arora-GB, described in [AG11; Alb+14].

A code example using the lattice estimator is given later in Chapter 3 and Code Example 3.1 with the useful part of the output in Figure 3.1. This example can be tweaked to estimate the security of arbitrary LWE/GLWE instances.

In this thesis, we heavily rely on the lattice estimator to build our noise oracles (see Section 3.1) that are one of the main building blocks of our optimization framework (see Chapter 4).

2.2 The Morphology of FHE Ciphertexts

In this section, we describe several kinds of ciphertexts. The security of those ciphertexts relies on the hardness of the problems introduced in Definitions 1 and 2.

First, we recall the LWE and the GLWE ciphertexts which are the more common and low-level type of ciphertexts. GLev and GGSW ciphertexts are then defined as special collections of GLWE ciphertexts.

2.2.1 LWE, RLWE & GLWE Ciphertexts

The most common type of ciphertexts in TFHE as described in [Chi+20a] is the LWE ciphertext. The security of an LWE ciphertext relies on the LWE problem (Definition 1).

Definition 4 (LWE Ciphertext) *Let $q \in \mathbb{N}$ be a ciphertext modulus. Let $n \in \mathbb{N}$ be an LWE dimension. Given an encoded message $\widetilde{m} \in \mathbb{Z}_q$ and a secret key $\vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n$, with elements either sampled from a uniform binary distribution, uniform ternary distribution or Gaussian distribution, an LWE ciphertext of $\widetilde{m}$ under the secret key $\vec{s}$ is defined as the tuple:*

$$\mathsf{ct} = \left(a_0, \cdots, a_{n-1}, b = \sum_{i=0}^{n-1} a_i \cdot s_i + \widetilde{m} + e\right) \in \mathsf{LWE}_{\vec{s}}(\widetilde{m}) \subseteq \mathbb{Z}_q^{n+1} \tag{2.1}$$

2. `https://github.com/malb/lattice-estimator`

such that $\{a_i\}_{i=0}^{n-1}$ are integers sampled from the uniform distribution in $\mathbb{Z}_q$, e is a noise (error) in $\mathbb{Z}_q$ sampled from a Gaussian distribution χ_σ. The parameter $n \in \mathbb{Z}_{>0}$ represents the number of elements in the LWE secret key.

Inside most FHE algorithms, we use another type of ciphertexts, GLWE ciphertexts, that are seen as a generalization of the LWE ciphertexts. The security of a GLWE ciphertext relies on the hardness of the GLWE problem introduced in Definition 2. In Definition 5, we use $\mathfrak{R}_{q,N} = \mathbb{Z}_q[X] / \left\langle X^N + 1 \right\rangle$ with N a power-of-two.

Definition 5 (GLWE Ciphertext) *Given an encoded message $\widetilde{M} \in \mathfrak{R}_{q,N}$ and a secret key $\vec{S} = (S_0, \cdots, S_{k-1}) \in \mathfrak{R}_{q,N}^k$, with coefficients either sampled from a uniform binary, uniform ternary or Gaussian distribution, a GLWE ciphertext of $\widetilde{M}$ under the secret key $\vec{S}$ is defined as the tuple:*

$$\mathsf{CT} = \left(A_0, \cdots, A_{k-1}, B = \sum_{i=0}^{k-1} A_i \cdot S_i + \widetilde{M} + E \right) \in \mathsf{GLWE}_{\vec{S}}(\widetilde{M}) \subseteq \mathfrak{R}_{q,N}^{k+1}$$

such that $\{A_i\}_{i=0}^{k-1}$ are polynomials in $\mathfrak{R}_{q,N}$ with coefficients sampled from the uniform distribution in $\mathbb{Z}_q$ and E is a noise (error) polynomial in $\mathfrak{R}_{q,N}$, with coefficients sampled from a Gaussian distributions χ_σ. The parameter $k \in \mathbb{Z}_{>0}$ represents the number of polynomials in the GLWE secret key.

If we set $N = 1$ and $k = n$ in Definition 5, we obtain the same type of ciphertext as in Definition 4. In the special case where $N = 1$, the security of the encryption relies on the hardness of the LWE problem (Definition 1) and not on the GLWE problem (Definition 2). By convention all along this thesis, we will write an LWE ciphertext, an LWE secret key and a message with a lower case, e.g. ct, $\vec{s}$ and m. On the contrary, we use upper case for a GLWE ciphertext, a GLWE secret key and a polynomial message when $N > 1$, e.g. CT, $\vec{S}$ and M.

Another special case of Definition 5 is when we set $k = 1$ and $N > 1$. This ciphertext is called an *RLWE ciphertext*.

Remark 1 (Encoding of a Message) *In Definition 4 (respectively Definition 5), we encrypt an encoding $\widetilde{m}$ of a message m (respectively an encoding $\widetilde{M}$ of a message M). We will see in Section 2.4 how to encode a message for TFHE. One of the major difference between TFHE and other FHE schemes is the encoding step. Regardless of the encoding,* *we can use Definitions 4 and 5 to encrypt ciphertexts for other FHE schemes based on (G)LWE.*

Definition 6 (GLWE Decryption) *Given a secret key $\vec{S} = (S_0, \cdots, S_{k-1}) \in \mathfrak{R}_{q,N}^k$ and a GLWE encryption* CT *of $\widetilde{M}$ under the secret key $\vec{S}$ as defined in Definition 5, the decryption of* CT *is defined as:*

$$\begin{aligned} \overline{M} &= B - \sum_{i=0}^{k-1} A_i \cdot S_i \\ &= \widetilde{M} + E \subseteq \mathfrak{R}_{q,N} \end{aligned} \tag{2.2}$$

The decryption of a GLWE ciphertext is the modular addition between the encoded message $\widetilde{M}$ and the noise polynomial E. In [Chi+20a], $\overline{M}$ is called the phase *of* CT *and is noted $\phi\left(\mathsf{CT}, \vec{S}\right)$.*

Throughout the rest of this thesis, we will sometimes use ciphertexts that are trivially encrypted. The definition below explains what a trivial encryption is.

Definition 7 (Trivial Encryption) *Given an encoded message $\widetilde{M} \in \mathfrak{R}_{q,N}$ and a secret key $\vec{S} = (S_0, \cdots, S_{k-1}) \in \mathfrak{R}_{q,N}^k$, a trivial GLWE encryption of $\widetilde{M}$ under the secret key $\vec{S}$ is defined as the tuple:*

$$\mathsf{CT} = \left(0, \cdots, 0, B = \widetilde{M}\right) \in \mathsf{GLWE}_{\vec{S}}(\widetilde{M}) \subseteq \mathfrak{R}_{q,N}^{k+1}$$

Of course, as the mask polynomials and the noise polynomial are set to 0, these ciphertexts are not secure as there is no encryption. They are mainly used to simplify the notation in some algorithms referring to an input that can be either a ciphertext or a plaintext.

2.2.2 Lev, RLev & GLev Ciphertexts

Using Definition 5, we can build more complex types of ciphertexts that are useful to define the public material used in some FHE algorithms. For instance, the keyswitching key in Algorithm 1 can be defined with the help of GLev ciphertexts. Informally, a GLev ciphertext is just a collection of GLWE ciphertexts.

Definition 8 (GLev Ciphertext) *Given a ciphertext modulus q, a decomposition base $\mathfrak{B} \in \mathbb{N}^*$ and a decomposition level $\ell \in \mathbb{N}^*$, a GLev ciphertext of a plaintext $M \in \mathfrak{R}_{q,N}$ under a GLWE secret key $\vec{S} \in \mathfrak{R}_{q,N}^k$ is defined as follows:*

$$\overline{\mathsf{CT}} = (\mathsf{CT}_0, \ldots, \mathsf{CT}_{\ell-1}) \in \mathsf{GLev}_{\vec{S}}(M)^{\mathfrak{B},\ell} \subseteq \mathfrak{R}_{q,N}^{\ell \times (k+1)} \tag{2.3}$$

such that

$$\forall 0 \leq j < \ell, \mathsf{CT}_j \in \mathsf{GLWE}_{\vec{S}}\left(\frac{q}{\mathfrak{B}^{j+1}} M\right) \subseteq \mathfrak{R}_{q,N}^{k+1}.$$

A GLev ciphertext with $N = 1$ is a *Lev ciphertext* and in this case we consider the parameter $n = k$ for the size of the LWE secret key. A GLev ciphertext with $k = 1$ and $N > 1$ is a *RLev ciphertext.*

2.2.3 GSW, RGSW & GGSW Ciphertexts

Using the GLev ciphertexts introduced in Definition 8, we can redefine another type of ciphertexts, the GGSW ciphertexts that can also be used to describe the public material of some FHE algorithms, for instance the bootstrapping key (Algorithm 17). Informally, a GGSW ciphertext is a collection of GLev ciphertexts. It was first introduced in [GSW13].

Definition 9 (GGSW Ciphertexts [GSW13; Chi+21]) *Given a decomposition base $\mathfrak{B} \in \mathbb{N}^*$ and a decomposition level $\ell \in \mathbb{N}^*$, a GGSW ciphertext of a plaintext $M \in \mathfrak{R}_{q,N}$ under a GLWE secret key $\vec{S} \in \mathfrak{R}_{q,N}^k$ is defined as follows:*

$$\overline{\overline{\mathsf{CT}}} = \left(\overline{\mathsf{CT}}_0, \ldots, \overline{\mathsf{CT}}_k\right) \in \mathsf{GGSW}_{\vec{S}}(M) \subseteq \mathfrak{R}_{q,N}^{(k+1) \times \ell \times (k+1)} \tag{2.4}$$

such that

$$\forall 0 \leq i \leq k, \overline{\mathsf{CT}}_i \in \mathsf{GLev}_{\vec{S}}^{\mathfrak{B},\ell}(-S_i \cdot M) \subseteq \mathfrak{R}_{q,N}^{\ell \times (k+1)}.$$

with the convention $S_k = -1$.

A GGSW ciphertext with $N = 1$ is a *GSW ciphertext*, and a GGSW ciphertext with $k = 1$ and $N > 1$ is a *RGSW ciphertext.*

2.3 TFHE and Its Variants

TFHE [Chi+16a; Chi+17; Chi+20a] is an (G)LWE-based FHE scheme which differentiates from the other (G)LWE-based cryptosystems because it supports a *very efficient boot-*

strapping technique. TFHE was originally proposed as an improvement of *FHEW* [DM15], a GSW [GSW13] based scheme with a fast bootstrapping for the evaluation of homomorphic Boolean gates. Apart from improving the bootstrapping of FHEW, TFHE also introduces new techniques in order to support more functionalities and to improve the homomorphic evaluation of complex circuits. The efficiency of TFHE comes in part from the choice of a small ciphertext modulus which allows to use CPU native types to represent a ciphertext both in the standard domain and in the Fourier domain.

In this document, we use different notations compared to the original TFHE papers [Chi+16a; Chi+17; Chi+20a]. In these, the message and ciphertext spaces are expressed by using the real torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$. In the TFHE library [Chi+16b], $\mathbb{T}$ is implemented by using native arithmetic modulo 2^{32} or 2^{64}, which means that they work on $\mathbb{Z}_q$ (with $q = 2^{32}$ or $q = 2^{64}$). This is why we prefer to use $\mathbb{Z}_q$ instead of $\mathbb{T}$, as already adopted in [Chi+20b]. It is made possible because there is an isomorphism between $\mathbb{Z}_q$ and $\frac{1}{q}\mathbb{Z}/\mathbb{Z}$ as explained in [Bou+20, Section 1].

First, we recall the basic operations that can be performed over ciphertexts, the addition and the multiplication by a known integer. Then, we detail several variants of the key switch operation. Next, we cover every building block that composes the Programmable Bootstrapping operation (PBS). Finally, we describe other techniques used to homomorphically evaluate lookup tables.

2.3.1 Additions

The first and simplest operation that can be done over ciphertexts is the addition. We will use this theorem to show how to analyze the noise after an operation over ciphertexts, see Proof 1.

Informally, the method consists in encrypting ciphertexts using Definition 5, performing the operation that we want to study, decrypting the result using Definition 6 and expressing the output noise as a function of the cryptographic parameters and of the input noise. As noise analyses are long even for a simple operation, we put most of them in the appendices.

Theorem 1 (GLWE addition) *Let $\vec{S} = (S_0, \cdots, S_{k-1})$ a GLWE secret key. Let $\mathsf{CT}_1 = (A_{0,1}, \cdots, A_{k-1,1}, B_1)$ and $\mathsf{CT}_2 = (A_{0,2}, \cdots, A_{k-1,2}, B_2)$ be two GLWE ciphertexts encrypted under the GLWE secret key $\vec{S}$ and with noise polynomials that are sampled from two centered Gaussian distributions $\chi_{\sigma_1} = \mathcal{N}(0, \sigma_1^2)$ and $\chi_{\sigma_2} = \mathcal{N}(0, \sigma_2^2)$ with*

$(\sigma_1, \sigma_2) \in \mathbb{N}_{>0}^2$, *two standard deviations. We further assume* χ_{σ_1} *and* χ_{σ_2} *to be statistically independent.*

The noise of the GLWE ciphertext defined as

$$\mathsf{CT}_3 = \mathsf{CT}_1 + \mathsf{CT}_2 = (A_{0,1} + A_{0,2}, \cdots, A_{k-1,1} + A_{k-1,2}, B_1 + B_2) \in \mathfrak{R}_q^{k+1}$$

follows a centered Gaussian distribution $\chi_{\sqrt{\sigma_1+\sigma_2}} = \mathcal{N}(0, \sigma_1^2 + \sigma_2^2)$.

Proof 1 (Theorem 1) *Let* $\vec{S} = (S_0, \cdots, S_{k-1})$, *a GLWE secret key. Let* CT_1 *and* CT_2 *be two GLWE ciphertexts such that* $\forall j \in \{1,2\}, \mathsf{CT}_j = (A_{0,j}, \cdots, A_{k-1,j}, B_j) \in \mathsf{GLWE}_{\vec{S}}(\widetilde{M_j}) \subseteq \mathfrak{R}_q^{k+1}$ *with* $\forall j \in \{1,2\}, B_j = \sum_{i=0}^{k-1} A_{i,j} \cdot S_i + \widetilde{M_j} + E_j$. *We assume that* E_1 *(respectively* E_2*) is a polynomial with coefficients sampled from a centered Gaussian distribution* $\chi_{\sigma_1} = \mathcal{N}(0, \sigma_1^2)$ *(respectively* χ_{σ_2}*). For* $i \in \{1,2\}, \sigma_i \in \mathbb{N}_{>0}$ *and represents a standard deviation.*

Let us define $\mathsf{CT}_3 = \mathsf{CT}_1 + \mathsf{CT}_2 \subseteq \mathfrak{R}_q^{k+1}$. *In the following,* $[\cdot]_q$ *represents the modular reduction of each coefficient of the polynomials. We have*

$$\begin{aligned}\mathsf{CT}_3 &= (A_{0,3}, \cdots, A_{k-1,3}, B_3)\\ &= \left([A_{0,1} + A_{0,2}]_q, \cdots, [A_{k-1,1} + A_{k-1,2}]_q, [B_1 + B_2]_q\right)\end{aligned}$$

Now, let us decrypt CT_3 *using Definition 6.*

$$\begin{aligned}\overline{M_3} &= B_3 - \sum_{i=0}^{k-1} A_{i,3} \cdot S_i\\ &= B_1 + B_2 - \sum_{i=0}^{k-1} (A_{i,1} + A_{i,2}) \cdot S_i\\ &= \cancel{\sum_{i=0}^{k-1} (A_{i,1} + A_{i,2})} \cdot S_i + \widetilde{M_1} + \widetilde{M_2} + E_1 + E_2 - \cancel{\sum_{i=0}^{k-1} (A_{i,1} + A_{i,2})} \cdot S_i\\ &= \widetilde{M_1} + \widetilde{M_2} + \underbrace{E_1 + E_2}_{\textit{noise of } \mathsf{CT}_3}\end{aligned}$$

Assuming that the noise coefficients of E_1 *and* E_2 *are independent, each coefficient of the noise polynomial of* CT_3 *follows a Gaussian distribution* $\mathcal{N}(0, \sigma_1^2 + \sigma_2^2)$.

$\square$

With the help of Theorem 1, we can observe that after the addition, the variance of the noise is larger than the noise variances of the input ciphertexts. This remark is not only valid for the addition, in fact, most of the operations performed over ciphertexts will increase the variance of the noise, with the exception of the rotation (Theorem 3) and the bootstrapping operations (see Section 2.3.3). In this thesis, we will sometimes say that a first ciphertext contains more noise than a second ciphertext, it means that the variance of the noise in the first ciphertext is larger than the noise variance of the second ciphertext.

Before explaining how to multiply a GLWE ciphertext with a polynomial, let us see what happens when we multiply a GLWE ciphertext with X. To do that, we first need to recall some interesting properties of $\mathfrak{R}_{q,N} = \mathbb{Z}_q[X] / \left\langle X^N + 1 \right\rangle$.

Theorem 2 (Rotation in $\mathfrak{R}_{q,N}$) *Let $P \in \mathfrak{R}_{q,N}$ be a polynomial such that $P = \sum_{i=0}^{N-1} p_i X^i$ and $\omega \in \mathbb{Z}$.*

Posing $\overline{\omega} = \omega \mod 2N$, we have $\overline{\omega} \in [\![0, 2N-1]\!]$. In $\mathfrak{R}_{q,N}$, we have

$$\begin{cases} X^N & = -1 \\ X^{2N} & = 1 \\ X^{\omega} & = X^{\overline{\omega}} \end{cases} \tag{2.5}$$

When $\overline{\omega} \in [\![0, N-1]\!]$, we have:

$$P \cdot X^{\omega} = \sum_{j=\overline{\omega}}^{N-1} p_{j-\overline{\omega}} \cdot X^j - \sum_{j=0}^{\overline{\omega}-1} p_{N+j-\overline{\omega}} \cdot X^j \tag{2.6}$$

Proof 2 (Theorem 2) *The first equations are direct consequences of the definition of the ring $\mathfrak{R}_{q,N}$.*

For the last equation, we have:

$$\begin{aligned} P \cdot X^{\omega} &= P \cdot X^{\overline{\omega}} \\ &= \sum_{j=0}^{N-1} p_j \cdot X^j \cdot X^{\overline{\omega}} \\ &= X^{\overline{\omega}} \cdot \sum_{j=0}^{N-1-\overline{\omega}} p_j \cdot X^j + X^{\overline{\omega}} \cdot \sum_{j=N-\overline{\omega}}^{N-1} p_j \cdot X^j \\ &= \sum_{j=\overline{\omega}}^{N-1} p_{j-\overline{\omega}} \cdot X^j + \underbrace{X^N}_{-1} \cdot \sum_{j=0}^{\overline{\omega}-1} p_{N+j-\overline{\omega}} \cdot X^j \\ &= \sum_{j=\overline{\omega}}^{N-1} p_{j-\overline{\omega}} \cdot X^j - \sum_{j=0}^{\overline{\omega}-1} p_{N+j-\overline{\omega}} \cdot X^j \end{aligned}$$

□

Now, let us recall the multiplication between a GLWE ciphertext and a power of X.

Theorem 3 (Multiplication between a GLWE Ciphertext and a Power of X) *Let $\vec{S} = (S_0, \cdots, S_{k-1})$ be a GLWE secret key. Let $\mathsf{CT}_{\mathsf{in}}$ be a GLWE ciphertext encrypted under the GLWE secret key $\vec{S}$, so that the coefficients of E are independently sampled from a centered Gaussian distribution χ_σ. Let $\omega \in \mathbb{N}$ such that $0 \leq \omega \leq N-1$.*

Each noise coefficient of the GLWE ciphertext $\mathsf{CT}_{\mathsf{out}} = X^\omega \cdot \mathsf{CT}_{\mathsf{in}}$ follows a centered Gaussian distribution χ_σ.

Proof 3 (Theorem 3) *Let $\mathsf{CT}_{\mathsf{in}} = \left(\vec{A}, B\right) \in \mathsf{GLWE}_{\vec{S}}(M)$ with $\vec{A} \in \mathfrak{R}_q^k$ and $B = \left\langle \vec{A}, \vec{S} \right\rangle + \widetilde{M} + E$ with $E = \sum_{j=0}^{N-1} e_j \cdot X^j$ and for $0 \leq j \leq N-1, e_j$ is drawn from a centered Gaussian distribution χ_σ. Let $\omega \in \mathbb{N}$ such that $0 \leq \omega \leq N-1$.*

We pose $\mathsf{CT}_{\mathsf{out}} = X^\omega \cdot \mathsf{CT}_{\mathsf{in}}$ and study the noise in $\mathsf{CT}_{\mathsf{out}}$.

$$\begin{aligned}\phi\left(\mathsf{CT}_{\mathsf{out}}, \vec{S}\right) &= X^\omega \cdot B - \sum_{i=0}^{k-1} A_i \cdot S_i \cdot X^\omega \\ &= X^\omega \cdot \left(\sum_{i=0}^{k-1} A_i \cdot S_i + M + E\right) - \sum_{i=0}^{k-1} A_i \cdot S_i \cdot X^\omega \quad = M \cdot X^\omega + E \cdot X^\omega\end{aligned}$$

Using Theorem 2, we have

$$E \cdot X^\omega = \sum_{j=\overline{\omega}}^{N-1} e_{j-\omega} \cdot X^j - \sum_{j=0}^{\omega-1} e_{N+j-\omega} \cdot X^j$$

Since, for $0 \leq j \leq N-1, \pm e_j$ is drawn from a centered Gaussian χ_σ, we conclude that each noise coefficient of $\mathsf{CT}_{\mathsf{out}}$ follows a centered Gaussian distribution χ_σ.

□

As we can add GLWE ciphertexts together (Theorem 1) and multiply them by powers of X (Theorem 3), we can also define the product between a GLWE ciphertext and an integer polynomial.

Theorem 4 (Multiplication between a GLWE Ciphertext and a Polynomial) *Let $\vec{S} = (S_0, \cdots, S_{k-1})$ be a GLWE secret key. Let $\mathsf{CT}_{\mathsf{in}}$ be a GLWE ciphertext encrypted under the GLWE secret key $\vec{S}$. The coefficients of E are independently sampled from a centered Gaussian distribution χ_σ. Let $D \in \mathbb{Z}[X]$ s.t. $D = \sum_{i=0}^{N-1} d_i \cdot X^i$.*

Each noise coefficient of the GLWE ciphertext $\mathsf{CT}_{\mathsf{out}} = D \cdot \mathsf{CT}_{\mathsf{in}}$ follows a centered Gaussian distribution $\chi_{\nu \cdot \sigma}$ where ν is the 2-norm of D defined as $\nu^2 = \sum_{i=0}^{N-1} d_i^2$.

Proof 4 (Theorem 4) *Let* $\mathsf{CT}_{\mathsf{in}} = \left(\vec{A}, B\right) \in \mathsf{GLWE}_{\vec{S}}(M)$ *with* $\vec{A} \in \mathfrak{R}_q^k$ *and* $B = \left\langle \vec{A}, \vec{S} \right\rangle + \widetilde{M} + E$ *with* $E = \sum_{j=0}^{N-1} e_j \cdot X^j$*. Let* $D \in \mathbb{Z}[X]$ *s.t.* $D = \sum_{i=0}^{N-1} d_i \cdot X^i$*.*

Setting $\mathsf{CT}_{\mathsf{out}} = D \cdot \mathsf{CT}_{\mathsf{in}}$*, we study the noise in* $\mathsf{CT}_{\mathsf{out}}$*.*

$$\begin{aligned}\phi\left(\mathsf{CT}_{\mathsf{out}}, \vec{S}\right) &= D \cdot B - \sum_{j=0}^{k-1} (D \cdot A_j) \cdot S_j \\ &= D \cdot M + \underbrace{D \cdot E}_{\textit{noise of } \mathsf{CT}_{\mathsf{out}}}\end{aligned}$$

Moreover,

$$\begin{aligned}D \cdot E &= \left(\sum_{j=0}^{N-1} e_j \cdot X^j\right) \cdot \left(\sum_{j=0}^{N-1} d_j \cdot X^j\right) \\ &= \sum_{i=0}^{N-1} X^i \cdot \left(\sum_{j=0}^{i} e_{i-j} \cdot d_j + \underbrace{X^N}_{\textit{reduction}} \cdot \sum_{j=i+1}^{N-1} e_{N+i-j} \cdot d_j\right) \\ &= \sum_{i=0}^{N-1} X^i \cdot \left(\sum_{j=0}^{i} e_{i-j} \cdot d_j - \sum_{j=i+1}^{N-1} e_{N+i-j} \cdot d_j\right) \\ &= \sum_{i=0}^{N-1} X^i \cdot \underbrace{\left(\sum_{j=0}^{N-1} \underbrace{(-1)^{\alpha_{i,j}} e_{\alpha_{i,j} \cdot N+i-j}}_{\epsilon_{i,j}} \cdot d_j\right)}_{e_{\mathsf{out},i}} \quad \textit{with } \alpha_{i,j} = \begin{cases} 0 \textit{ if } i-j \geq 0 \\ 1 \textit{ otherwise} \end{cases}\end{aligned}$$

As $\forall\ 0 \leq j < N, e_j$ *follows a centred Gaussian distribution* χ_σ*, we immediately have that each coefficient of* $D \cdot E$ *follows a centered Gaussian distribution* $\chi_{\nu \cdot \sigma}$ *with* ν *the 2-norm of* $\{d_i\}_{i=0}^{N-1}$ *defined as* $\nu^2 = \sum_{i=0}^{N-1} d_i^2$*.*

□

Remark 2 (Several Noise Distributions) *If we assume that for* $i \in \{0, \cdots, N-1\}$*, we have* $e_i \hookleftarrow \mathcal{N}(0, \sigma_i^2)$*, we get that* $\epsilon_{i,j} \hookleftarrow \mathcal{N}\left(0, \sigma^2_{\alpha_{i,j} \cdot N+i-j}\right)$*. Finally, with* $e_{\mathsf{out},i}$ *the ith coefficient of* $D \cdot E$*, we have:*

$$e_{\mathsf{out},i} \hookleftarrow \mathcal{N}\left(0, \sum_{j=0}^{N-1} \sigma^2_{\alpha_{i,j} \cdot N+i-j} \cdot d_j^2\right)$$

Remark 3 (GLev and GGSW Additions) *Theorems 1 and 4 can be trivially extended for LWE, GLev and GGSW ciphertexts.*

Using Theorem 1 and Theorem 4, we can define the dot product.

Theorem 5 (Homomorphic Dot Product) *Let* $\mathsf{ct}_i \in \mathsf{LWE}_{\vec{s}}(\widetilde{m_i}) \subseteq \mathbb{Z}_q^{n+1}$ *for* $0 \leq i \leq \alpha - 1$ *be a list of LWE ciphertexts under* $\vec{s} = (s_0, \cdots, s_{n-1})$ *an LWE secret key so that noises are sampled independently from a centered Gaussian distribution* χ_σ*. Let* $\{\omega_i\}_{0 \leq i \leq \alpha-1} \in \mathbb{Z}^\alpha$ *be arbitrary integers.*

A homomorphic dot product is a dot product between a vector of LWE ciphertexts and a vector of integers, resulting in an LWE ciphertext $\mathsf{ct}_{\mathsf{out}}$ *i.e.,*

$$\mathsf{ct}_{\mathsf{out}} = \sum_{i=0}^{\alpha-1} \mathsf{ct}_i \cdot \omega_i \tag{2.7}$$

The noise in the output ciphertext follows the distribution $\chi_{\nu \cdot \sigma} = \mathcal{N}(0, \nu^2\sigma^2)$ *with* $\nu^2 = \sum_{i=0}^{\alpha-1} \omega_i^2$*, the squared 2-norm.*

Thus, given a dot product between a vector of ciphertexts with the same (Gaussian) noise distribution and a vector of integers, we only need the 2-norm ν *to characterize the output noise of a dot product.*

Proof 5 (Theorem 5) *The proof is just a composition of the formulae in Theorems 1 and 4.*

□

We saw with Theorem 1 that we can add ciphertexts encrypted under the same GLWE keys. Let us see what happens when we try to add ciphertexts encrypted under different keys. Let CT_1 and CT_2 be two GLWE ciphertexts with CT_1 (respectively CT_2) encrypted under a GLWE secret key $\vec{S_1}$ (respectively $\vec{S_2}$). Let us define CT_3 such that $\mathsf{CT}_3 = \mathsf{CT}_1 + \mathsf{CT}_2$. We have

$$\mathsf{CT}_3 = (A_{0,1} + A_{0,2}, \cdots, A_{k-1,1} + A_{k-1,2}, B_1 + B_2)$$

with

$$B_1 + B_2 = \sum_{i=0}^{k-1} A_{i,1} \cdot S_{i,1} + \sum_{i=0}^{k-1} A_{i,2} \cdot S_{i,2} + \widetilde{M_1} + \widetilde{M_2} + E_1 + E_2$$

As one can see, it is no longer possible to decrypt CT_3 because we do not have access to $(A_{0,1}, \cdots, A_{k-1,1})$ and $(A_{0,2}, \cdots, A_{k-1,2})$ anymore. In the following, we introduce the concept of key switch. A key switch allows to change the key of a ciphertext without having to know the secret key. This operation can be useful if we want to add ciphertexts that are encrypted under different keys.

2.3.2 Key Switches

As explained above, a key switch allows to homomorphically change the key of a ciphertext. Given a ciphertext $\mathsf{CT}_{\mathsf{in}} \in \mathsf{GLWE}_{\vec{S}_{\mathsf{in}}}\left(\widetilde{M}\right)$ we can use a key switch to obtain a ciphertext $\mathsf{CT}_{\mathsf{out}} \in \mathsf{GLWE}_{\vec{S}_{\mathsf{out}}}\left(\widetilde{M}\right)$. $\vec{S}_{\mathsf{out}}$ can have different parameters than $\vec{S}_{\mathsf{in}}$, for instance the polynomial size N and the GLWE dimension k could be different. Several algorithms allow to perform a key switch in the state of the art. We will describe the main versions here. Every version needs some public material to perform the key switch, namely the keyswitching key. Intuitively, the keyswitching key is composed of the key $\vec{S}_{\mathsf{in}}$ encrypted under $\vec{S}_{\mathsf{out}}$.

First, we introduce the radix decomposition: this algorithm is used as a sub-routine in every key switch algorithm and in other FHE algorithms as well. We then provide a summary of the existing key switch techniques.

Radix Decomposition

Several ways exist to perform a radix decomposition. Here, we focus on the classical way to do it as described in [Chi+20a]. We refer the reader to [Joy21] for a more complete analysis of the impact of the decomposition algorithm. In particular, Joye introduces a new decomposition that behaves asymptotically better than other known decompositions.

Definition 10 *Let $\mathfrak{B} \in \mathbb{N}^*$ be a decomposition base and $\ell \in \mathbb{N}^*$ be a decomposition level. The decomposition algorithm with respect to $\mathfrak{B}$ and ℓ is noted $\mathsf{dec}^{(\mathfrak{B},\ell)}$. It takes as input an integer $x \in \mathbb{Z}_q$ and outputs a vector of integers $(x_1, \cdots, x_\ell) \in \mathbb{Z}_q^\ell$ such that*

$$\left\langle \mathsf{dec}^{(\mathfrak{B},\ell)}(x), \left(\frac{q}{\mathfrak{B}} \cdots \frac{q}{\mathfrak{B}^\ell}\right)\right\rangle = \left\lfloor x \cdot \frac{\mathfrak{B}^\ell}{q} \right\rceil \cdot \frac{q}{\mathfrak{B}^\ell} \in \mathbb{Z}_q.$$

$\mathsf{dec}^{(\mathfrak{B},\ell)}(x)$ *is referred to as the decomposition vector of x. In [Chi+20a], $(x_1, \cdots, x_\ell)$ are defined as the unique integers satisfying*

$$\left\lfloor x \cdot \frac{\mathfrak{B}^\ell}{q} \right\rceil \cdot \frac{q}{\mathfrak{B}^\ell} = \sum_{i=1}^{\ell} x_i \cdot \frac{q}{\mathfrak{B}^i} \text{ with } x_i \in \left[\!\left[-\frac{\mathfrak{B}}{2}, \frac{\mathfrak{B}}{2} \right[\!\right[.$$

In [Joy21], a balanced decomposition algorithm is introduced where $x_i \in \llbracket -\frac{\mathfrak{B}}{2}, \frac{\mathfrak{B}}{2} \rrbracket$*, which is better for the noise propagation as it gives a centered distribution for the* $\{x_i\}_{i=1}^{\ell}$*.*

Usually, the decomposition is performed starting from the most significant bits. When applying the decomposition on a vector of integers, the result is a vector of decomposition vectors of integers.

Note that it is possible to decompose an integer polynomial $P \in \mathfrak{R}_q$ with this algorithm i.e.,

$$\left\langle \mathsf{dec}^{(\mathfrak{B},\ell)}(P), \left(\frac{q}{\mathfrak{B}} \cdots \frac{q}{\mathfrak{B}^{\ell}}\right) \right\rangle = \left\lfloor P \cdot \frac{\mathfrak{B}^{\ell}}{q} \right\rceil \cdot \frac{q}{\mathfrak{B}^{\ell}} \in \mathfrak{R}_q$$

resulting in a vector of polynomials. Applying this kind of decomposition on a vector of polynomials, we get a vector of vectors of polynomials.

LWE Key Switch

Algorithm 1: $\mathsf{CT} \leftarrow \mathsf{KS}(\mathsf{ct}, \mathsf{KSK})$

```
Context: { $\vec{s} = (s_0, \cdots, s_{n-1})$ : the input LWE secret key
         { $\vec{S} = (S_0, \ldots, S_{k-1})$ : the output GLWE secret key
Input: { $\mathsf{ct} \in \mathsf{LWE}_{\vec{s}}(\widetilde{m}) = (a_0, \cdots, a_{n-1}, b)$
       { $\mathsf{KSK} = \left\{\overline{\mathsf{CT}}_i \in \mathsf{GLev}_{\vec{S}}^{\mathfrak{B},\ell}(s_i)\right\}_{0 \le i \le n-1}$ : keyswitching key from $\vec{s}$ to $\vec{S}$
Output: $\mathsf{CT} \in \mathsf{GLWE}_{\vec{S}}(\widetilde{m})$
1 begin
2     $\mathsf{CT} \in \mathsf{GLWE}_{\vec{S}}(\widetilde{m}) \leftarrow (0, \cdots, 0, b) - \sum_{i=0}^{n-1} \left\langle \overline{\mathsf{CT}}_i, \mathsf{dec}^{(\mathfrak{B},\ell)}(a_i) \right\rangle$
3 end
```

Using Theorem 1 and Definition 10, Algorithm 1 defines the LWE key switch as introduced in [Chi+17, Algorithm 1]. The output ciphertext is a GLWE ciphertext which, of course, can also be an LWE ciphertext when $N = 1$.

Theorem 6 (LWE Key Switch) *Let* $\mathsf{ct} \in \mathsf{LWE}_{\vec{s}}(\widetilde{m}) \in \mathbb{Z}_q^{n+1}$ *be an LWE ciphertext encrypting* $\widetilde{m} \in \mathbb{Z}_q$*, under the LWE secret key* $\vec{s} = (s_0, \ldots, s_{n-1}) \in \mathbb{Z}_q^n$*, with noise sampled from* χ_σ*. Let* $\vec{S}$ *be a GLWE secret key such that* $\vec{S} = (S_0, \ldots, S_{k-1}) \in \mathfrak{R}_q^{k+1}$*. Let* $\mathsf{KSK} = \left\{\overline{\mathsf{CT}}_i \in \mathsf{GLev}_{\vec{S}}^{\mathfrak{B},\ell}(s_i) \in \mathfrak{R}_q^{\ell \times (k+1)}\right\}_{0 \le i \le n-1}$ *be a key switching key from* $\vec{s}$ *to* $\vec{S}$ *with noise sampled from* $\chi_{\sigma_{\mathsf{KSK}}}$*.*

The variance of the noise after the key switch is

$$\begin{aligned}\mathsf{Var}_{\mathsf{KS}} = \sigma^2 + n \cdot \left(\frac{q^2}{12\mathfrak{B}^{2\ell}} - \frac{1}{12}\right) \cdot \left(\mathsf{Var}(s_i) + \mathbb{E}^2(s_i)\right) \\ + \frac{n}{4} \cdot \mathsf{Var}(s_i) + n \cdot \ell \cdot \sigma^2_{\mathsf{KSK}} \cdot \frac{\mathfrak{B}^2 + 2}{12} \ .\end{aligned} \tag{2.8}$$

The algorithmic complexity of Algorithm 1, referred to as the cost is

$$\begin{aligned}\mathsf{Cost}\left(\mathsf{KS}\right)^{(\ell,n,k,N)} &= n\mathsf{Cost}\left(dec\right)^{(\ell)} + \ell n(k+1)N\mathsf{Cost}\left(mul\right) \\ &+ \left((\ell n - 1)(k+1)N\right)\mathsf{Cost}\left(add\right) \ .\end{aligned} \tag{2.9}$$

The size (in bits) of the key switching key $\mathsf{Size}\left(\mathsf{KSK}\right)$ *is computed with the following formula:*

$$\mathsf{Size}\left(\mathsf{KSK}\right) := n \cdot \ell \cdot (k+1) \cdot N \cdot \lceil \log_2(q) \rceil$$

Proof 6 (Theorem 6) *The proof of the noise variance is detailed in Appendix A.3.*

$\square$

Remark 4 (Cost of an Algorithm) *In Theorem 6, we provide a cost function for the key switch. This function is used to estimate the running time of Algorithm 1 on a single thread. To define it, we count the number of low-level operations (additions, multiplications, castings between integer types, etc.) in the algorithm. Cost functions are especially useful for the optimization framework introduced in Chapter 4. More details are given in Section 4.1.1.*

Remark 5 (Public function in Key Switch) *It is possible to leverage the key switch algorithm to also compute a public function* $f : \mathbb{Z}_q^\alpha \to \mathbb{Z}_q$ *on several input ciphertexts* $\{\mathsf{ct}_i\}_{i=0}^{\alpha-1}$*.* f *must verify the following properties:*

- $\forall (x,y) \in \left(\mathbb{Z}_q^\alpha\right)^2, f(x+y) = f(x) + f(y)$
- $\forall s \in \mathbb{Z}, \forall x \in \mathbb{Z}_q^\alpha, f(s \cdot x) = s \cdot f(x)$
- $\exists R, \forall (x,y) \in \left(\mathbb{Z}_q^\alpha\right)^2, |f(x) - f(y)| < R|x-y|$

To do that, one needs to compute

$$\mathsf{CT} \in \mathsf{GLWE}_{\vec{S}}\left(\widetilde{m}\right) \leftarrow \left(0, \cdots, 0, f\left(\{b_i\}_{i=0}^{\alpha-1}\right)\right) + \sum_{i=0}^{n-1} \left\langle \overline{\mathsf{CT}}_i, \mathsf{dec}^{(\mathfrak{B},\ell)}\left(f\left(\{a_{i,j}\}_{j=0}^{\alpha-1}\right)\right)\right\rangle .$$

In [Chi+20a], this is called a Public Functional Key Switching*, see [Chi+17, Algorithm 1].*

We now specify in Algorithm 2, a GLWE key switch that works the same way as the LWE key switch of Algorithm 1. The dot product between the keyswitching key and the polynomial decompositions can be performed using the FFT which significantly speeds up the execution time of the GLWE key switch compared to the LWE key switch.

Algorithm 2: $\mathsf{CT}_{\mathsf{out}} \leftarrow \mathsf{GlweKeySwitch}(\mathsf{CT}_{\mathsf{in}}, \mathsf{KSK})$

```
Context: $\begin{cases} \vec{S}_{\mathsf{in}} \in \mathfrak{R}_{q,N}^{k_{\mathsf{in}}} : \text{the input GLWE secret key} \\ \vec{S}_{\mathsf{in}} = (S_{\mathsf{in},0}, \cdots, S_{\mathsf{in},k_{\mathsf{in}}-1}) \\ \vec{S}_{\mathsf{out}} \in \mathfrak{R}_{q,N}^{k_{\mathsf{out}}} : \text{the output GLWE secret key} \\ \ell \in \mathbb{N} : \text{ the number of levels of the decomposition} \\ \mathfrak{B} \in \mathbb{N} : \text{ the base of the decomposition} \end{cases}$
Input: $\begin{cases} \mathsf{CT}_{\mathsf{in}} = (A_0, \cdots, A_{k_{\mathsf{in}}-1}, B) \in \mathsf{GLWE}_{\vec{S}_{\mathsf{in}}}(P) \subseteq \mathfrak{R}_{q,N}^{k_{\mathsf{in}}+1}, \text{ with } P \in \mathfrak{R}_{q,N} \\ \mathsf{KSK} = \{\overline{\mathsf{CT}}_i\}_{0 \le i \le k_{\mathsf{in}}-1}, \text{ with} \\ \overline{\mathsf{CT}}_i \in \mathsf{GLev}_{\vec{S}_{\mathsf{out}}}^{\mathfrak{B},\ell}(S_{\mathsf{in},i}), \text{ for } 0 \le i \le k_{\mathsf{in}}-1 \end{cases}$
Output: $\mathsf{CT}_{\mathsf{out}} \in \mathsf{GLWE}_{\vec{S}_{\mathsf{out}}}(P)$
/* Keep the $B$ part                                              */
1 Set $\mathsf{CT}_{\mathsf{out}} := (0, \cdots, 0, B) \in \mathfrak{R}_{q,N}^{k_{\mathsf{out}}+1}$
2 for $i \in [\![0; k_{\mathsf{in}}-1]\!]$ do
      /* Decompose the mask                                        */
3     Update $\mathsf{CT}_{\mathsf{out}} = \mathsf{CT}_{\mathsf{out}} - \left\langle \overline{\mathsf{CT}}_i, \mathsf{dec}^{(\mathfrak{B},\ell)}(A_i) \right\rangle$
4 return $\mathsf{CT}_{\mathsf{out}}$
```

Private LWE Key Switch

We saw in Remark 5 that it was possible to compute a public linear function on several ciphertexts when doing a key switch. Here, we want to apply a private linear function. To keep this function private, we need to hide it in the keyswitching key. We show in Algorithm 3 how to do that for one input ciphertext and refer to [Chi+17, Algorithm 2] for a version with several input ciphertexts. In the definition of the keyswitching key, we impose $s_n = -1$ by convention.

Theorem 7 (LWE Private Key Switch) *Let f be a linear function from $\mathbb{Z}_q \to \mathbb{Z}_q$ such that $\exists R \in \mathbb{Z}, \forall x \in \mathbb{Z}_q, f(x) = R \cdot x$. Let* $\mathsf{ct} \in \mathsf{LWE}_{\vec{s}}(\widetilde{m}) \in \mathbb{Z}_q^{n+1}$ *be an LWE ciphertext*

```
Algorithm 3: $\mathsf{CT} \leftarrow \mathsf{PrivateKS}(\mathsf{ct}, \mathsf{KSK})$
  Context: $\begin{cases} \vec{s} = (s_0, \cdots, s_{n-1}) : \text{ the input LWE secret key} \\ \vec{S} = (S_0, \ldots, S_{k-1}) : \text{ the output GLWE secret key} \\ s_n = -1 \end{cases}$
  Input: $\begin{cases} \mathsf{ct} \in \mathsf{LWE}_{\vec{s}}(\widetilde{m_i}) = (a_0, \cdots, a_{n-1}, b) \\ \mathsf{KSK} = \left\{ \overline{\mathsf{CT}}_i \in \mathsf{GLev}^{\mathfrak{B},\ell}_{\vec{S}}(f(s_i)) \right\}_{0 \le i \le n} : \text{ keyswitching key from } \vec{s} \text{ to } \vec{S} \end{cases}$
  Output: $\mathsf{CT} \in \mathsf{GLWE}_{\vec{S}}(\widetilde{m})$
1 begin
2     $\mathsf{CT} \in \mathsf{GLWE}_{\vec{S}}(\widetilde{m}) \leftarrow -\left\langle \overline{\mathsf{CT}}_n, \mathsf{dec}^{(\mathfrak{B},\ell)}(b) \right\rangle - \sum_{i=0}^{n-1} \left\langle \overline{\mathsf{CT}}_i, \mathsf{dec}^{(\mathfrak{B},\ell)}(a_i) \right\rangle$
3 end
```

encrypting $\widetilde{m} \in \mathbb{Z}_q$ under the LWE secret key $\vec{s} = (s_0, \ldots, s_{n-1}) \in \mathbb{Z}_q^n$, with noise sampled respectively from χ_σ. Let $\vec{S}$ be a GLWE secret key such that $\vec{S} = (S_0, \ldots, S_{k-1}) \in \mathfrak{R}_q^{k+1}$. Let $\mathsf{KSK} = \left\{ \overline{\mathsf{CT}}_i \in \mathsf{GLev}^{\mathfrak{B},\ell}_{\vec{S}}(f(s_i)) \in \mathfrak{R}_q^{\ell \times (k+1)} \right\}_{0 \le i \le n}$ be a key switching key from $\vec{s}$ to $\vec{S}$ with noise sampled from $\chi_{\sigma_{\mathsf{KSK}}}$ and $s_n = -1$ by convention.

The variance of the noise after the private key switch is:

$$\begin{aligned} \mathsf{Var}_{\mathsf{PrivateKS}} = R^2\sigma^2 + (n+1) \cdot \left(\frac{q^2}{12\mathfrak{B}^{2\ell}} - \frac{1}{12} \right) \cdot \left(\mathsf{Var}(s_i) + \mathbb{E}^2(s_i) \right) \\ + \frac{n+1}{4} \cdot \mathsf{Var}(s_i) + (n+1) \cdot \ell \cdot \sigma_{\mathsf{KSK}}^2 \cdot \frac{\mathfrak{B}^2 + 2}{12} \end{aligned} \tag{2.10}$$

The algorithmic complexity of Algorithm 3, referred to as the cost is

$$\begin{aligned} \mathsf{Cost}\left(\mathsf{PrivateKS}\right)^{(\ell,n,k,N)} = (n+1)\,\mathsf{Cost}\left(dec\right)^{(\ell)} + \ell\,(n+1)\,(k+1)N\mathsf{Cost}\left(mul\right) \\ + ((\ell(n+1)-1)(k+1)N)\mathsf{Cost}\left(add\right) \end{aligned} \tag{2.11}$$

The size (in bits) of the private LWE key switching key $\mathsf{Size}\left(\mathsf{KSK}\right)$ *is computed with the following formula:*

$$\mathsf{Size}\left(\mathsf{KSK}\right) := (n+1) \cdot \ell \cdot (k+1) \cdot N \cdot \lceil \log_2(q) \rceil$$

Proof 7 (Theorem 7) *The proof of Theorem 6 given in Appendix A.3 can be easily adapted to prove Theorem 7 using the fact that $\forall x \in \mathbb{Z}_q, f(x) = R \cdot x$.*

□

Packing Key Switch

The packing key switch enables to pack several LWE ciphertexts (Definition 4) into one GLWE ciphertext (Definition 5). It takes as input a set of α LWE ciphertexts as well as a set of α indices. Given the set of indices $\{i_j\}_{j=0}^{\alpha-1}$, we have

$$\mathsf{CT}_{\mathsf{out}} \in \mathsf{GLWE}_{\vec{S}}\left(\sum_{j=0}^{\alpha-1} m_j \cdot X^{i_j}\right) \leftarrow \mathsf{PackingKS}(\{\mathsf{ct}_j\}_{j=0}^{\alpha-1}, \{i_j\}_{j=0}^{\alpha-1}, \mathsf{KSK})$$

where ct_j is an LWE encryption of $m_j \in \mathbb{Z}_q$.

The packing key switch can be described easily with the help of the LWE key switch (Algorithm 1 and Theorem 6), the multiplication by a power of X (Theorem 3) and the addition (Theorem 1). In short, it takes as input several LWE ciphertexts, keyswitches them using Algorithm 1, multiplies each of them by a power of X and finally adds the keyswitched ciphertexts together. Theorem 8 gives the noise variance and the cost of this operation.

Theorem 8 (LWE-to-GLWE Packing Key Switch) *We start with the simplest case where we pack a single LWE ciphertext. Let* $\mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}}(\widetilde{m}) \in \mathbb{Z}_q^{n+1}$ *be an LWE ciphertext encrypting* $\widetilde{m} \in \mathbb{Z}_q$ *under the LWE secret key* $\vec{s} = (s_0, \dots, s_{n-1}) \in \mathbb{Z}_q^n$*, with noise sampled from* χ_σ*. Let* $\vec{S'}$ *be a GLWE secret key such that* $\vec{S'} = \left(S'_0, \dots, S'_{k-1}\right) \in \mathfrak{R}_q^{k+1}$*. Let* $\mathsf{KSK} = \left\{\overline{\mathsf{CT}}_i \in \mathsf{GLev}_{\vec{S'}}^{\mathfrak{B},\ell}(s_i) \in \mathfrak{R}_q^{\ell\times(k+1)}\right\}_{0\le i\le n-1}$ *be a key switching key from* $\vec{s}$ *to* $\vec{S'}$ *with noise sampled from* $\chi_{\sigma_{\mathsf{KSK}}}$*.*

There are two different variances after a packing key switch: one for the coefficient we just filled written $\mathsf{Var}_{\mathsf{fill}}$ *and another for the empty coefficients* $\mathsf{Var}_{\mathsf{emp}}$*. Those variances are estimated by:*

$$\begin{aligned}\mathsf{Var}_{\mathsf{fill}}^{(1)} &= \sigma^2 + n\left(\frac{q^2}{12\mathfrak{B}^{2\ell}} - \frac{1}{12}\right)\left(\mathsf{Var}(s_i) + \mathbb{E}^2(s_i)\right) + \frac{n}{4}\mathsf{Var}(s_i) + n\ell\sigma_{\mathsf{KSK}}^2\frac{\mathfrak{B}^2+2}{12} \\ \mathsf{Var}_{\mathsf{emp}}^{(1)} &= n\ell\sigma_{\mathsf{KSK}}^2\frac{\mathfrak{B}^2+2}{12}\end{aligned} \tag{2.12}$$

When we pack $1 \le \alpha \le N$ *LWE ciphertexts, we have* $\mathsf{Var}_{\mathsf{fill}}^{(\alpha)} = \mathsf{Var}_{\mathsf{fill}}^{(1)} + (\alpha-1)\cdot\mathsf{Var}_{\mathsf{emp}}^{(1)}$ *and* $\mathsf{Var}_{\mathsf{emp}}^{(\alpha)} = \alpha\cdot\mathsf{Var}_{\mathsf{emp}}^{(1)}$ *The cost of the algorithm is:*

$$\begin{aligned}\mathsf{Cost}\left(PackingKS\right)^{(\alpha,\ell,n,k,N)} &= \alpha n\mathsf{Cost}\left(dec\right)^{(\ell)} + \alpha\ell n(k+1)N\mathsf{Cost}\left(mul\right) \\ &\quad + ((\alpha\ell n-1)(k+1)N + \alpha)\mathsf{Cost}\left(add\right)\end{aligned} \tag{2.13}$$

Proof 8 (Theorem 8) *In the proof, we express the decryption of the resulting ciphertext, obtaining the message added to the noise so we can estimate the two variances. The detailed computation leading us to the aforementioned noise formulae are provided in Appendix A.3.*

□

Fast Keyswitch

In [Che+20], the authors introduce an LWE key switch that leverages the FFT. The algorithm is described with RLWE ciphertexts but can be easily extended with GLWE ciphertexts. The first step of the algorithm is to convert an LWE ciphertext into a GLWE ciphertext, then perform a GLWE keyswitch (Algorithm 2) and finally compute a sample extract (see below Algorithm 5). The algorithm is presented in Algorithm 4.

Algorithm 4 is very efficient because the GLWE key switch can leverage the FFT. The main drawback is that the algorithm takes as input an LWE ciphertext with a power-of-two LWE dimension. In Chapter 8, we will introduce a new type of secret key and we will be able to adapt Algorithm 4 to remove the need of a power-of-two LWE dimension.

```
Algorithm 4: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{FFTLweKeySwitch}(\mathsf{ct}_{\mathsf{in}}, \mathsf{KSK})$

Context: $\begin{cases} \vec{s}_{\mathsf{in}} \in \mathbb{Z}_q^N : \text{the input LWE secret key s.t. } \vec{s}_{\mathsf{in}} = (s_{\mathsf{in},0}, \cdots, s_{\mathsf{in},N-1}) \\ \vec{S}_{\mathsf{in}} \in \mathbb{Z}_q^N : \text{the polynomial version of } \vec{s}_{\mathsf{in}} \text{ s.t. } \vec{S}_{\mathsf{in}} = \sum_{i=0}^{N-1} s_i X^{-i} \\ \vec{s}_{\mathsf{out}} \in \mathbb{Z}_q^N : \text{the output LWE secret key s.t. } \vec{s}_{\mathsf{out}} = (s_{\mathsf{out},0}, \cdots, s_{\mathsf{out},N-1}) \\ \vec{S}_{\mathsf{out}} \in \mathfrak{R}_{q,N} : \text{the polynomial version of } \vec{s}_{\mathsf{out}} \text{ s.t. } \vec{S}_{\mathsf{out}} = \sum_{i=0}^{N-1} s_{\mathsf{out},i} X^{i} \\ \ell \in \mathbb{N} : \text{ the number of levels of the decomposition} \\ \mathfrak{B} \in \mathbb{N} : \text{ the base of the decomposition} \end{cases}$
Input: $\begin{cases} \mathsf{ct}_{\mathsf{in}} = (a_0, \cdots, a_{N-1}, b) \in \mathsf{LWE}_{\vec{s}_{\mathsf{in}}}(\widetilde{m}) \subseteq \mathbb{Z}_q^{N+1} \\ \mathsf{KSK} = \left\{\overline{\mathsf{CT}}_i\right\}_{0 \le i \le k_{\mathsf{in}}-1}, \text{ with} \\ \quad \overline{\mathsf{CT}}_i \in \mathsf{GLev}_{\vec{S}_{\mathsf{out}}}^{\mathfrak{B},\ell}(S_{\mathsf{in},i}), \text{ for } 0 \le i \le k_{\mathsf{in}}-1 \end{cases}$
Output: $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s}_{\mathsf{out}}}(\widetilde{m})$
  /* Convert the input LWE into a GLWE                                  */
1 $\mathsf{CT}_1 \in \mathfrak{R}_{q,N}^2 \leftarrow \left(\sum_{i=0}^{N-1} a_i X^i, b\right)$
  /* Apply the GLWE key switch (Algorithm 2)                             */
2 $\mathsf{CT}_2 \in \mathfrak{R}_{q,N}^2 \leftarrow \mathsf{GlweKeySwitch}(\mathsf{CT}_1, \mathsf{KSK})$
  /* Sample Extract the result with Algorithm 5                          */
3 $\mathsf{ct}_{\mathsf{out}} \in \mathbb{Z}_q^{N+1} \leftarrow \mathsf{SampleExtract}(\mathsf{CT}_2, 0)$
4 return $\mathsf{ct}_{\mathsf{out}}$
```

2.3.3 PBS & Its Building Blocks

In this section, we define the building blocks of TFHE's PBS as described in [Chi+16a; Chi+17; Chi+20a]. To do so, we need to introduce computational steps called the sample extract, the modulus switch and the external product. On top of the external product, we build the cmux. Using the cmux, we explain how to perform a blind rotate and finally, using the modulus switch, the blind rotate and the sample extract, we define the PBS algorithm.

Sample Extract

Algorithm 5: $\mathsf{ct} \leftarrow \mathsf{SampleExtract}\left(\mathsf{CT}, \alpha\right)$

```
Context: {
    $\vec{s} = (s_0, \cdots, s_{kN-1})$ :  the output LWE secret key
    $\vec{S} = (S_0, \ldots, S_{k-1})$ :  the input GLWE secret key
    $\forall 0 \leq i \leq k-1,\ S_i = \sum_{j=0}^{N-1} s_{i \cdot N + j} X^j$
    $\widetilde{M} = \sum_{i=0}^{N-1} \widetilde{m_i} \cdot X^i$ :  a polynomial message
    $\forall 0 \leq i \leq k-1,\ A_i = \sum_{j=0}^{N-1} a_{i,j} X^j$
    $B = \sum_{j=0}^{N-1} b_j X^j$
Input: {
    $\mathsf{CT} \in \mathsf{GLWE}_{\vec{S}}\left(\widetilde{M}\right) = (A_0, \cdots, A_{k-1}, B)$
    $\alpha \in \{0, \cdots, N-1\}$
Output: $\mathsf{ct} \in \mathsf{LWE}_{\vec{s}}\left(\widetilde{m_p}\right)$
1  begin
2      $b' \leftarrow b_p$
3      for $0 \leq i \leq k-1$ do
4          for $0 \leq j \leq \alpha$ do
5              $a'_{i \cdot N + j} \leftarrow a_{i, \alpha - j}$
6          end
7          for $\alpha + 1 \leq j < N$ do
8              $a'_{i \cdot N + j} \leftarrow -a_{i, N + \alpha - j}$
9          end
10     end
11     $\mathsf{ct} \in \mathsf{LWE}_{\vec{s}}\left(\widetilde{m_p}\right) \leftarrow (a'_0, \cdots, a'_{kN-1}, b')$
12 end
```

The sample extract is an operation taking as input a GLWE ciphertext (Definition 6) $\mathsf{CT} \in \mathsf{GLWE}_{\vec{S}}\left(\sum_{i=0}^{N-1} \widetilde{m_i} X^i\right)$ and outputting an LWE ciphertext (Definition 4) $\mathsf{ct} \in \mathsf{LWE}_{\vec{s}}\left(\widetilde{m_p}\right)$ as output with $0 \leq p \leq N-1$. The output ciphertext ct will be encrypted using a flattened representation of the input GLWE key $\vec{S}$. We explain in Definition 11 what a flattened GLWE secret key is.

Definition 11 (Flattened Representation of a GLWE Secret Key) *A GLWE secret key* $\vec{S} = \left(S_0 = \sum_{j=0}^{N-1} s_{0,j}X^j, \cdots, S_{k-1} = \sum_{j=0}^{N-1} s_{k-1,j}X^j\right) \in \mathfrak{R}_{q,N}^k$ *can be* flattened *into an LWE secret key* $\bar{\vec{s}} = (\bar{s}_0, \cdots, \bar{s}_{kN-1}) \in \mathbb{Z}^{kN}$ *in the following manner:* $\bar{s}_{iN+j} := s_{i,j}$*, for* $0 \leq i < k$ *and* $0 \leq j < N$*.*

A version of the sample extract for RLWE ciphertexts was introduced in [Chi+20a, Section 4.2] and can be easily extended to GLWE ciphertexts as we did in Algorithm 5. Informally, the sample extract consists in simply rearranging some of the coefficients of the GLWE input ciphertext to build the output LWE ciphertext encrypting one of the coefficients of the input polynomial plaintext.

Theorem 9 (Sample Extract) *Let* $0 \leq \alpha \leq N-1$ *be an index. Let* $\mathsf{CT} \in \mathsf{GLWE}_{\vec{S}}\left(\sum_{i=0}^{N-1} m_i X^i\right)$ *be a GLWE ciphertext (Definition 6) with coefficients sampled from* χ_σ*. After the sample extract (Algorithm 5), we obtain a ciphertext* $\mathsf{ct} \in \mathsf{LWE}_{\bar{\vec{s}}}(m_\alpha)$ *encrypted with* $\bar{\vec{s}}$*, the flattened representation of* $\vec{S}$ *(Definition 11).*

The variance after the sample extract is:

$$\mathsf{Var}_{\mathsf{SampleExtract}} = \sigma^2 \tag{2.14}$$

It means that the sample extract does not add any noise.

The cost of the sample extract is:

$$\mathsf{Cost}\left(\mathsf{SampleExtract}\right) = (k+1) \cdot N \cdot \mathsf{Cost}\left(\mathsf{Copy}\right) \tag{2.15}$$

In practice, most of the time we can neglect the cost of the sample extract as it is very fast compared to the other FHE algorithms.

Proof 9 (Theorem 9) *A proof of a more generic algorithm is given in Appendix A.4, taking* $\phi = k \cdot N$ *gives the proof of Theorem 9.*

$\square$

Modulus Switch

The modulus switch takes as input an LWE ciphertext with some ciphertext modulus q and outputs a ciphertext modulus w while preserving the underlying plaintext. The

```
Algorithm 6: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{MS}(\mathsf{ct}_{\mathsf{in}})$
    Context: $\begin{cases} \vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n : \text{ the input LWE secret key} \\ N : \text{ a polynomial size} \end{cases}$
    Input: $\mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}}(\widetilde{m}) = (a_0, \cdots, a_{n-1}, a_n = b) \subseteq \mathbb{Z}_q^{n+1}$
    Output: $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s}}\left(\left\lfloor \frac{\widetilde{m}\cdot 2N}{q} \right\rceil\right) \subseteq \mathbb{Z}_{2N}^{n+1}$
1 begin
2     for $0 \leq i \leq n$ do
3         $a'_i \leftarrow \left[\left\lfloor \frac{a_i \cdot 2N}{q} \right\rceil\right]_{2N}$
4     end
5     $\mathsf{ct}_{\mathsf{out}} \leftarrow \left(a'_0, \cdots, a'_{n-1}, b' = a'_n\right)$
6 end
```

algorithm is described in Algorithm 6. In the context of TFHE's PBS, $w = 2 \cdot N$ with N the polynomial size of the ring $\mathfrak{R}_{q,N}$.

Theorem 10 (Modulus Switch) *Let q and w be two ciphertext moduli. Let $\vec{s}$ be an LWE secret key such that $\vec{s} = (s_0, \cdots, s_{n-1})$. Let $\mathsf{ct}_{\mathsf{in}}$ be an LWE ciphertext (Definition 4) such that $\mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}}(\widetilde{m}) \subseteq \mathbb{Z}_q^{n+1}$.*

The output of the sample extract (Algorithm 5) is a ciphertext $\mathsf{ct}_{\mathsf{out}}$ such that $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s}}(\widetilde{m}) \subseteq \mathbb{Z}_w^{n+1}$.

The variance of the noise of $\mathsf{ct}_{\mathsf{out}}$ is:

$$\mathsf{Var}(\mathsf{MS}) = \frac{w^2\sigma_{\mathsf{in}}^2}{q^2} + \frac{1}{12} - \frac{w^2}{12q^2} + \frac{n}{24} + \frac{nw^2}{48q^2} . \tag{2.16}$$

The cost of the modulus switch is:

$$\mathsf{Cost}(\mathsf{MS})^{n,q,w} = (n+1) \cdot \mathsf{Cost}(\mathsf{Round})^{(q,w)} . \tag{2.17}$$

Proof 10 (Theorem 10) *The proof for a more generic algorithm is detailed in Appendix A.2. Taking $\vartheta = 0$ and $\varkappa = 0$ gives the variance claimed above.*

□

External Product.

In Algorithm 7, we describe the external product. The external product takes as inputs a GGSW ciphertext (Definition 9) encrypting $Q \in \mathfrak{R}_{q,N}$ and a GLWE ciphertext (Definition 5) encrypting P and returns a GLWE ciphertext of $Q \cdot P$. It is very similar to

```
Algorithm 7: $\mathsf{CT}_{\mathsf{out}} \leftarrow \mathsf{ExternalProduct}\left(\mathsf{CT}_{\mathsf{in}}, \overline{\mathsf{CT}}\right)$
Context: $\begin{cases} \vec{S} \in \mathfrak{R}_{q,N}^{k} : \text{the input secret key} \\ Q \in \mathfrak{R}_{q,N} \\ \mathsf{CT}_{\mathsf{in}} = (A_0, \cdots, A_{k-1}, B) \in \mathfrak{R}_{q,N}^{k+1} \\ \mathsf{CT}_{i,j} \in \mathsf{GLWE}_{\vec{S}}\left(\frac{q}{\mathfrak{B}^{j+1}} \cdot Q \cdot S_i\right), \text{ for } 0 \leq i \leq k-1 \text{ and } 0 \leq j \leq \ell - 1 \\ \mathsf{CT}_{k,j} \in \mathsf{GLWE}_{\vec{S}}\left(\frac{q}{\mathfrak{B}^{j+1}} \cdot Q\right), \text{ for } 0 \leq j \leq \ell - 1 \\ \ell \in \mathbb{N} : \text{ the number of levels in the decomposition} \\ \mathfrak{B} \in \mathbb{N} : \text{ the base in the decomposition} \end{cases}$
Input: $\begin{cases} \mathsf{CT}_{\mathsf{in}} \in \mathsf{GLWE}_{\vec{S}}(P), \text{ with } P \in \mathfrak{R}_{q,N} \\ \overline{\mathsf{CT}} = \left\{ \vec{K}_i = (\mathsf{CT}_{i,0}, \cdots, \mathsf{CT}_{i,\ell-1}) \right\}_{0 \leq i \leq k} \end{cases}$
Output: $\mathsf{CT}_{\mathsf{out}} \in \mathsf{GLWE}_{\vec{S}}(Q \cdot P)$
/* Decompose the $B$ part                                              */
1 $\mathsf{CT}_{\mathsf{out}} \leftarrow \left\langle \vec{K}_k, \mathsf{dec}^{(\mathfrak{B},\ell)}(B) \right\rangle$
2 for $i \in [\![0; k-1]\!]$ do
      /* Decompose the mask                                            */
3     $\mathsf{CT}_{\mathsf{out}} \leftarrow \mathsf{CT}_{\mathsf{out}} - \left\langle \vec{K}_i, \mathsf{Decomp}^{(\mathfrak{B},\ell)}(A_i) \right\rangle$
4 return $\mathsf{CT}_{\mathsf{out}}$
```

the private key switch (Algorithm 3); in fact, the external product is a private key switch with a GLWE ciphertext as input instead of an LWE ciphertext and with the constraint that the input key is the same as the output key. Later, we present an algorithm that generalizes both the external product and the private key switch (Algorithm 38).

Theorem 11 (External Product) *Let* $\mathsf{CT}_{\mathsf{in}} \in \mathsf{GLWE}_{\vec{S}}(P) \in \mathfrak{R}_{q,N}^{k+1}$ *be a GLWE ciphertext encrypting* $P \in \mathfrak{R}_{q,N}$*, under the GLWE secret key* $\vec{S} = (S_0, \ldots, S_{k-1}) \in \mathfrak{R}_{q,N}^{k+1}$*, with noise sampled from* χ_σ*. Let* $Q \in \mathfrak{R}_{q,N}$ *and* $\overline{\mathsf{CT}} = \left\{ \overline{\mathsf{CT}}_i \in \mathsf{GLev}_{\vec{S}}^{\mathfrak{B},\ell}(Q \cdot S_i) \in \mathfrak{R}_q^{\ell \times (k+1)} \right\}_{0 \leq i \leq k-1}$ *be a GGSW (Definition 9) with noise sampled from* $\chi_{\sigma_{\mathsf{BSK}}}$ *and* $S_k = 1$*.*

In the special case where Q *is a constant polynomial with a message drawn uniformly in* $\{0, 1\}$*, the variance after the external product is:*

$$\begin{aligned}\mathsf{Var}\left(\mathsf{ExternalProduct}\right) = \ell \cdot (k+1) \cdot N \cdot \frac{\mathfrak{B}^2+2}{12} \cdot \sigma_2^2 + \\ + \frac{\sigma_1^2}{2} + \frac{q^2 - \mathfrak{B}^{2\ell}}{24\mathfrak{B}^{2\ell}} \cdot \Big(1 + kN \cdot \Big(\mathsf{Var}\left(s_i\right) + \mathbb{E}^2\left(s_i\right)\Big)\Big) \\ + \frac{kN}{8} \cdot \mathsf{Var}(s_i) + \frac{1}{16} \cdot \left(1 - kN \cdot \mathbb{E}(s_i)\right)^2\end{aligned} \tag{2.18}$$

where $\mathbb{E}(s_i)$ and $\mathsf{Var}\left(s_i\right)$ are the expected value and the variance of the secret key coefficients.

The cost of the algorithm is:

$$\begin{aligned}\mathsf{Cost}\left(\mathsf{ExternalProduct}\right)^{(\ell,k,N)} = (k+1)N\mathsf{Cost}\left(dec\right)^{(\ell)} + \ell(k+1)\mathsf{Cost}\left(FFT\right) \\ + (k+1)\ell(k+1)N\mathsf{Cost}\left(multFFT\right) \\ + (k+1)(\ell(k+1)-1)N\mathsf{Cost}\left(addFFT\right) \\ + (k+1)\mathsf{Cost}\left(iFFT\right)\end{aligned} \tag{2.19}$$

Proof 11 (Theorem 11) *The proof of the noise after an external product is detailed in Appendix A.2.*

□

Remark 6 (Internal Product [DM15]) *One of the main improvements of TFHE [Chi+20a] compared to FHEW [DM15] is the use of the external product instead of an internal product. Informally, the internal product takes two GGSW ciphertexts (Definition 9) instead of one GLWE ciphertext and one GGSW ciphertext. It outputs one GGSW ciphertext instead of a GLWE ciphertext. The internal product is composed of several external products. In practice, the noise after an internal product is the same as after an external product. As there are several external products, the cost of the internal product is higher than the one of an external product.*

CMUX.

The CMUX is a homomorphic selector. Given two GLWE ciphertexts (Definition 5) encrypting two polynomials P_0 and P_1 and a GGSW ciphertexts (Definition 9) encrypting a bit β, the CMUX returns a GLWE encryption of P_0 if $\beta = 0$ and a GLWE encryption of P_1 if $\beta = 1$ i.e. it returns a GLWE encryption of P_β. Intuitively, the algorithm homomorphically computes $P_\beta = (P_1 - P_0) \cdot \beta + P_0$.

Algorithm 8: $\mathsf{CT}_{\mathsf{out}} \leftarrow \mathsf{CMUX}\left(\mathsf{CT}_0, \mathsf{CT}_1, \overline{\overline{\mathsf{CT}}}\right)$

```
Context: $\begin{cases} \vec{S} \in \mathfrak{R}^k_{q,N} : \text{the input GLWE secret key} \\ \beta \in \{0,1\} \\ \mathsf{CT}_{\mathsf{in}} = (A_0, \cdots, A_{k-1}, B) \in \mathfrak{R}^{k+1}_{q,N} \\ \mathsf{CT}_{i,j} \in \mathsf{GLWE}_{\vec{S}}\left(\frac{q}{\mathfrak{B}^{j+1}} \cdot \beta \cdot S_i\right), \text{for } 0 \le i \le k-1 \text{ and } 0 \le j \le \ell - 1 \\ \mathsf{CT}_{k,j} \in \mathsf{GLWE}_{\vec{S}}\left(\frac{q}{\mathfrak{B}^{j+1}} \cdot \beta\right), \text{for } 0 \le j \le \ell - 1 \\ \ell \in \mathbb{N} : \text{the number of levels of the decomposition} \\ \mathfrak{B} \in \mathbb{N} : \text{the base of the decomposition} \end{cases}$
Input: $\begin{cases} \mathsf{CT}_0 \in \mathsf{GLWE}_{\vec{S}}(P_0), \text{ with } P_0 \in \mathfrak{R}_{q,N} \\ \mathsf{CT}_1 \in \mathsf{GLWE}_{\vec{S}}(P_1), \text{ with } P_1 \in \mathfrak{R}_{q,N} \\ \overline{\overline{\mathsf{CT}}} = \left\{\vec{K}_i = (\mathsf{CT}_{i,0}, \cdots, \mathsf{CT}_{i,\ell-1})\right\}_{0 \le i \le k} \end{cases}$
Output: $\mathsf{CT}_{\mathsf{out}} \in \mathsf{GLWE}_{\vec{S}}(P_\beta)$
/* External Product using Algorithm 7 and Theorem 11                    */
1 $\mathsf{CT}_{\mathsf{out}} \leftarrow \mathsf{ExternalProduct}\left((\mathsf{CT}_1 - \mathsf{CT}_0), \overline{\overline{\mathsf{CT}}}\right) + \mathsf{CT}_0$
2 return $\mathsf{CT}_{\mathsf{out}}$
```

This algorithm is built on top of the external product (Algorithm 7 and Theorem 11) and is the cornerstone of TFHE's PBS. The algorithm is detailed in Algorithm 8 and in the following theorem, we give the noise formula for a CMUX.

Theorem 12 (CMUX) *Let* $\forall i \in \{0,1\}, \mathsf{CT}_i \in \mathsf{GLWE}_{\vec{S}}(P_i) \in \mathfrak{R}^{k+1}_{q,N}$ *be two GLWE ciphertexts encrypting respectively* $P_0 \in \mathfrak{R}_{q,N}$ *and* $P_1 \in \mathfrak{R}_{q,N}$*, under the GLWE secret key* $\vec{S} = (S_0, \ldots, S_{k-1}) \in \mathfrak{R}^{k+1}_{q,N}$*, with noise sampled from* χ_σ*. Let* $\beta \in \{0,1\}$*. Let* $\overline{\overline{\mathsf{CT}}} = \left\{\overline{\mathsf{CT}}_i \in \mathsf{GLev}^{\mathfrak{B},\ell}_{\vec{S}}(\beta \cdot S_i) \in \mathfrak{R}_q^{\ell \times (k+1)}\right\}_{0 \le i \le k}$ *be a GGSW (Definition 9) with noise sampled from* $\chi_{\sigma_{\mathsf{BSK}}}$*.*

The variance after the cmux is:

$$\begin{aligned} \mathsf{Var}(\mathsf{CMUX}) = \ell \cdot (k+1) \cdot N \cdot \frac{\mathfrak{B}^2 + 2}{12} \cdot \sigma_2^2 + \\ + \frac{\sigma_1^2}{2} + \frac{q^2 - \mathfrak{B}^{2\ell}}{24\mathfrak{B}^{2\ell}} \cdot \left(1 + kN \cdot \left(\mathsf{Var}(s_i) + \mathbb{E}^2(s_i)\right)\right) \\ + \frac{kN}{8} \cdot \mathsf{Var}(s_i) + \frac{1}{16} \cdot (1 - kN \cdot \mathbb{E}(s_i))^2 \end{aligned} \tag{2.20}$$

where $\mathbb{E}(s_i)$ *and* $\mathsf{Var}(s_i)$ *are the expected value and variance of the secret key coefficients.*

```
Algorithm 9: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{BlindRotate}\left(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \mathsf{CT}_f\right)$
Context: $\begin{cases} \vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n : \text{ the LWE input secret key} \\ \vec{S'} = \left(S'_0, \ldots, S'_{k-1}\right) \in \mathfrak{R}_q^k : \text{ a GLWE secret key} \\ P_f \in \mathfrak{R}_q : \text{a } r\text{-redundant LUT for } x \mapsto f(x) \\ f : \mathbb{Z} \to \mathbb{Z} : \text{a function} \\ \widetilde{m} : \text{ encoded value of the message m} \end{cases}$
Input: $\begin{cases} \mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}}(m) = (a_0, \cdots, a_{n-1}, a_n = b) \in \mathbb{Z}_{2N}^{n+1} \\ \mathsf{BSK} = \left\{\overline{\overline{\mathsf{CT}}}_i \in \mathsf{GGSW}_{\vec{S'}}^{\mathfrak{B},\ell}\left(s_i\right)\right\}_{i=0}^{n-1} : \text{ a bootstrapping key from } \vec{s} \text{ to } \vec{S'} \\ \mathsf{CT}_f \in \mathsf{GLWE}_{\vec{S'}}\left(P_f\right) \in \mathfrak{R}_q^{k+1} \end{cases}$
Output: $\mathsf{CT}_{\mathsf{out}} \in \mathsf{GLWE}_{\vec{S'}}\left(P_f \cdot X^{-m}\right)$
1 begin
2     $\mathsf{CT}_{\mathsf{out}} \leftarrow \mathsf{CT}_f \cdot X^{-b}$
3     for $0 \leq i \leq n-1$ do
4         $\mathsf{CT}_0 \leftarrow \mathsf{CT}_{\mathsf{out}}$
5         $\mathsf{CT}_1 \leftarrow \mathsf{CT}_{\mathsf{out}} \cdot X^{a_i}$
6         $\mathsf{CT}_{\mathsf{out}} \leftarrow \mathsf{CMUX}\left(\mathsf{CT}_0, \mathsf{CT}_1, \overline{\overline{\mathsf{CT}}}_i\right)$
7     end
8 end
9 return $\mathsf{CT}_{\mathsf{out}}$
```

The cost of the algorithm is:

$$\begin{aligned} \mathsf{Cost}\left(\mathsf{ExternalProduct}\right)^{(\ell,k,N)} &= (k+1)N\mathsf{Cost}\left(dec\right)^{(\ell)} + \ell(k+1)\mathsf{Cost}\left(FFT\right) \\ &+ (k+1)\ell(k+1)N\mathsf{Cost}\left(multFFT\right) \\ &+ (k+1)(\ell(k+1)-1)N\mathsf{Cost}\left(addFFT\right) \\ &+ (k+1)\mathsf{Cost}\left(iFFT\right) + 2\cdot(k+1)\,N\mathsf{Cost}\left(add\right)\ . \end{aligned} \tag{2.21}$$

Proof 12 (Theorem 12) *The proof of the noise after a cmux in the context of a PBS is detailed in Appendix A.2.*

□

Blind Rotate

The blind rotate algorithm is one of the three main building blocks of the PBS. It consists in rotating a lookup table homomorphically. Given a GLWE ciphertext CT encrypting a polynomial P and an LWE ciphertext encrypting an encoded value $\widetilde{m}$ of a message m, it returns a GLWE ciphertext encrypting $P \cdot X^{-m} \in \mathfrak{R}_{q,N}$. Using Theorem 2, we know that the constant term of $P \cdot X^{-m}$ is the m-th coefficient of P if $0 \leq m < N$.

We need to introduce the concept of redundant lookup table. The motivation behind those lookup tables will be explained in the next part when we will introduce the PBS.

Definition 12 (Redundant Lookup Table) *A redundant LUT is a lookup table encoding a function f, whose entries are redundantly represented inside the coefficients of a polynomial in $\mathfrak{R}_{q,N}$. In practice, the redundancy consists in a r times (with r a system parameter) repetition of the entries $f(i)$ of the LUT with a certain shift. We call* mega-cell *each block of successive redundant values in a r-redundant lookup table:*

$$P_f = X^{-r/2} \cdot \sum_{i=0}^{N/r-1} X^{i \cdot r} \cdot \left(\sum_{j=0}^{r-1} \widetilde{f(i)} \cdot X^j \right) \tag{2.22}$$

with $\widetilde{f(i)}$ an encoded value of a message $f(i)$. The redundancy is used to perform the rounding operation during bootstrapping.

We described the blind rotate in Algorithm 9 and in Theorem 13.

Theorem 13 (Blind Rotate) *Let $\mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}}(\widetilde{m})$ be an LWE ciphertext (Definition 4) encrypting an encoded value $\widetilde{m}$ of a message m such that $\mathsf{ct}_{\mathsf{in}} = (a_0, \cdots, a_{n-1}, a_n = b) \in \mathbb{Z}_{2N}^{n+1}$ and $\vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n$. Let $\mathsf{CT}_f \in \mathsf{GLWE}_{\vec{S}'}(P_f) \subseteq \mathfrak{R}_{q,N}^{k+1}$, a GLWE ciphertext encrypting an r-redundant LUT for $f : \mathbb{Z} \to \mathbb{Z}$ (Definition 12) with $\vec{S}' = \left(S'_0, \ldots, S'_{k-1}\right) \in \mathfrak{R}_{q,N}^k$. Let BSK a collection of GGSW ciphertexts (Definition 9) such that $\mathsf{BSK} = \left\{ \overline{\overline{\mathsf{CT}}}_i \in \mathsf{GGSW}_{\vec{S}'}^{\mathfrak{B},\ell}(s_i) \right\}_{i=0}^{n-1}$. The noises in BSK are drawn fom a distribution $\chi_{\sigma_{\mathsf{BSK}}}$.*

Algorithm 9 outputs a ciphertext $\mathsf{CT}_{\mathsf{out}} \in \mathsf{GLWE}_{\vec{S}}(P_f \cdot X^{-m})$ where m is the input message.

In a PBS, CT_f is a trivial encryption of P_f (Definition 7). In this context, the variance of the noise in $\mathsf{CT}_{\mathsf{out}}$ is:

$$\begin{aligned}\mathsf{Var}(\mathsf{PBS}) = n \cdot \ell \cdot (k+1) \cdot N \cdot \frac{\mathfrak{B}^2 + 2}{12} \cdot \mathsf{Var}(\mathsf{BSK}) + \\ + n \cdot \frac{q^2 - \mathfrak{B}^{2\ell}}{24\mathfrak{B}^{2\ell}} \cdot \Big(1 + kN \cdot \Big(\mathsf{Var}(s_i) + \mathbb{E}^2(s_i)\Big)\Big) + \frac{nkN}{8} \cdot \mathsf{Var}(s_i) \\ + \frac{n}{16} \cdot (1 - kN \cdot \mathbb{E}(s_i))^2\end{aligned} \tag{2.23}$$

The cost of the blind rotate is:

$$\mathsf{Cost}\left(\mathsf{BlindRotate}\right)^{(n,\ell,k,N)} = n \cdot \mathsf{Cost}\left(CMUX\right)^{(\ell,k,N)} \tag{2.24}$$

Proof 13 (Theorem 13) *The proof of Equation* (2.23) *is given in Appendix A.2. The proof extends to the case where* CT_f *is not a trivial encryption.*

$\square$

Remark 7 (Interest of the Modulus Switch in the PBS) *To make the blind rotate work (Algorithm 9), we need* $2N$ *to be equal to the ciphertext modulus of the input ciphertext, which is impractical for* $q = 2^{64}$*. As the cost of the blind rotate depends on* N*, we want* N *as small as possible. To this end, we perform a modulus switch before a blind rotate to reduce the ciphertext modulus. This operation adds a lot of noise but allows to choose polynomial sizes way smaller than the size of* q*.*

Programmable Bootstrap (PBS).

We have introduced in the previous parts all the building blocks needed to explain the PBS. We call Programmable Bootstrap [Chi+20a; Chi+20b; CJP21], or PBS, an FHE algorithm that enables to reset the noise in a ciphertext to a fixed level (when certain conditions are fulfilled) and to homomorphically evaluate, at the same time, a lookup-table on the encrypted message. Such an operator takes as input an LWE ciphertext encrypting a message m, a bootstrapping key BSK (i.e., a list of GGSW ciphertexts encrypting the elements of the LWE secret key used to encrypt the message m), an (trivial) encryption of a r-redundant lookup table P_f (Definition 12), and outputs an LWE ciphertext with a fixed level of noise encrypting the message $P_f[m]$ when the process is successful. The algorithm outputs the correct output up to a failure probability that can be estimated using the variance of the noise and that we note p_{fail}.

The PBS is composed of three major steps: the modulus switch (Algorithm 6 and Theorem 10), the blind rotate (Algorithm 9 and Theorem 13) and the sample extract (Algorithm 5 and Theorem 9).

Theorem 14 (Programmable Bootstrap (PBS)) *Let* $\vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n$ *be a binary LWE secret key. Let* $\vec{S}' = \left(S'_0, \ldots, S'_{k-1}\right) \in \mathfrak{R}_{q,N}^k$ *be a GLWE binary secret key such that* $S'_i = \sum_{j=0}^{N-1} s'_{i \cdot N+j} \cdot X^j$*, and* $\vec{s}' = (s'_0, \cdots, s'_{kN-1})$ *be the corresponding binary*

LWE secret key. Let P_f be an r-redundant LUT for a function $f : \mathbb{Z}_q \to \mathbb{Z}_q$ as defined in Definition 12.

Then Algorithm 10 takes as input an LWE ciphertext $\mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}}(\widetilde{m}) \in \mathbb{Z}_q^{n+1}$ with noise distribution $\chi_{\sigma_{\mathsf{in}}}$ and with $\widetilde{m}$ an encoded value of a message $m \in \mathbb{Z}_p$ such that $\widetilde{m} = \Delta_{\mathsf{in}} \cdot m$ with $\Delta_{\mathsf{in}} = \frac{q}{2p}$, a bootstrapping key $\mathsf{BSK} = \left\{\overline{\overline{\mathsf{CT}}}_i \in \mathsf{GGSW}^{\mathfrak{B},\ell}_{\vec{S}'}(s_i)\right\}_{i=0}^{n-1}$ from $\vec{s}$ to $\vec{S}'$ and a (possibly trivial) GLWE encryption of P_f, and returns an LWE ciphertext $\mathsf{ct}_{\mathsf{out}}$ under the secret key $\vec{s'}$, encrypting $\widetilde{f(m)}$ i.e., an encoded value of the message $f(m)$ with a probability $1 - p_{\mathsf{fail}}$ if and only if

- *the input noise has variance $\sigma_{\mathsf{in}}^2 < \frac{\Delta_{\mathsf{in}}^2}{4 \cdot z^*(p_{\mathsf{fail}})^2} - \frac{q^2}{12w^2} + \frac{1}{12} - \frac{nq^2}{24w^2} - \frac{n}{48}$, where $z^*(p_{\mathsf{fail}})$ is the standard score (Definition 22) associated to the failure probability p_{fail} and $w = 2N$,*
- *for an arbitrary f, we need $m < \frac{q}{2}$.*

The output noise after the PBS is estimated by the formula:

$$\begin{aligned}\mathsf{Var}(\mathsf{PBS}) = n\ell(k+1)N\frac{\mathfrak{B}^2+2}{12}\mathsf{Var}(\mathsf{BSK}) + n\frac{q^2-\mathfrak{B}^{2\ell}}{24\mathfrak{B}^{2\ell}}\left(1+\frac{kN}{2}\right) \\ + \frac{nkN}{32} + \frac{n}{16}\left(1-\frac{kN}{2}\right)^2\end{aligned} \tag{2.25}$$

The cost of Algorithm 10 is:

$$\begin{aligned}\mathsf{Cost}\left(GenPBS\right)^{(n,\ell,k,N)} &= \mathsf{Cost}\left(ModulusSwitching\right)^{(n)} + \mathsf{Cost}\left(BlindRotate\right)^{(\ell,k,N)} \\ &+ \mathsf{Cost}\left(SampleExtract\right)^{(N)}\end{aligned} \tag{2.26}$$

The size (in bits) of the traditional bootstrapping key $\mathsf{Size}(\mathsf{BSK})$ is computed with the following formula:

$$\mathsf{Size}(\mathsf{BSK}) := n \cdot \ell_{\mathsf{PBS}} \cdot (k+1)^2 \cdot N \cdot \lceil \log_2(q) \rceil$$

Proof 14 (Theorem 14) *A more generic proof of Equation* (2.25) *is detailed in Appendix A.2. Taking $\varkappa = 0$ and $\vartheta = 0$ gives the claimed formula.*

□

Remark 8 (Negacyclic Function) *The second condition in Theorem 14 which states that we need $m < \frac{q}{2}$ can be removed if the function f has the following property:*

$$f\left(x + \frac{q}{2}\right) = -f(x), \forall x \in \mathbb{Z}_q \tag{2.27}$$

In this case, the function f is said to be negacyclic.

For odd message modulus (see Definition 13), we can also remove this condition.

Algorithm 10: $\mathsf{ct_{out}} \leftarrow \mathsf{PBS}\left(\mathsf{ct_{in}}, \mathsf{BSK}, \mathsf{CT}_f, \right)$

Context:
$$\begin{cases} \vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n : \text{ the input LWE secret key} \\ \vec{s}' = (s'_0, \cdots, s'_{kN-1}) \in \mathbb{Z}_q^{kN} : \text{ the output LWE secret key} \\ \vec{S}' = \left(S'_0, \ldots, S'_{k-1}\right) \in \mathfrak{R}_q^k : \text{ a GLWE secret key} \\ \forall 0 \le i \le k-1,\ S'_i = \sum_{j=0}^{N-1} s'_{i\cdot N+j} X^j \in \mathfrak{R}_q \\ P_f \in \mathfrak{R}_q : \text{an } r\text{-redundant LUT for } x \mapsto f(x) \\ f : \mathbb{Z} \to \mathbb{Z} : \text{a function} \end{cases}$$

Input:
$$\begin{cases} \mathsf{ct_{in}} \in \mathsf{LWE}_{\vec{s}}(\widetilde{m}) = (a_0, \cdots, a_{n-1}, a_n = b) \in \mathbb{Z}_q^{n+1} \\ \mathsf{BSK} = \left\{\overline{\overline{\mathsf{CT}}}_i \in \mathsf{GGSW}_{\vec{S}'}^{\beta,\ell}(s_i)\right\}_{i=0}^{n-1} : \text{ a bootstrapping key from } \vec{s} \text{ to } \vec{S}' \\ \mathsf{CT}_f \in \mathsf{GLWE}_{\vec{S}'}(P_f) \in \mathfrak{R}_q^{k+1} \end{cases}$$

Output: $\mathsf{ct_{out}} \in \mathsf{LWE}_{\vec{s}'}\left(\widetilde{f(m)}\right)$ if we respect the requirements of Theorem 14

```
1 begin
      /* modulus switching (Algorithm 6)                          */
2     $\mathsf{ct_{MS}} \leftarrow \mathsf{ModulusSwitch}(\mathsf{ct_{in}})$
      /* blind rotate of the LUT (Algorithm 9)                    */
3     $\mathsf{CT} \leftarrow$ BlindRotate $(\mathsf{CT}_f, \mathsf{ct_{MS}}, \mathsf{BSK})$ ;
      /* sample extract the constant term (Algorithm 5)           */
4     $\mathsf{ct_{out}} \leftarrow \mathsf{SampleExtract}(\mathsf{CT}, 0)$
5 end
```

Remark 9 (Key Switch and PBS) *More often than not, we apply a key switch before a PBS. Later in this manuscript (see Figure 4.6), we will examine in detail the optimal way to combine these two algorithms. For simplicity, we will sometimes refer to the sequential combination of a key switch and a PBS as KS-PBS.*

2.3.4 Other LUT Evaluation Algorithms

In the previous part, we introduced the PBS (Algorithm 10 and Theorem 14) and its building blocks (Algorithms 5, 6 and 9 and Theorems 9, 10 and 13). The key features of a PBS is to reduce the noise and to evaluate a lookup table simultaneously.

In the literature, other algorithms also allow to perform a noise reduction and a lookup table evaluation. In this section, we describe some of them.

Circuit Bootstrap

In [Chi+17, Alg. 6], a technique called *circuit bootstrapping* was introduced. Let $\vec{s}$ be an LWE secret key, $\vec{S'}$ be a GLWE secret key and $\vec{s'}$ the flattened version of $\vec{S'}$ (Definition 11). It takes as input an LWE ciphertext (Definition 4) $\mathsf{ct} \in \mathsf{LWE}_{\vec{s}}(\widetilde{m})$ and converts it into a GGSW ciphertext (Definition 9) $\overline{\overline{\mathsf{CT}}} \in \mathsf{GGSW}^{\mathfrak{B},\ell}_{\vec{S'}}(m)$, and reduces the noise at the same time.

The circuit bootstrapping is composed of a series of ℓ PBS steps to create ℓ ciphertexts $\{\mathsf{ct}_i\}_{0\leq i\leq \ell-1}$ encrypting $m \cdot \frac{q}{\mathfrak{B}^{i+1}}$ for $0 \leq i \leq \ell - 1$ under the key $\vec{s'}$, the flattened key of $\vec{S}$ (Definition 11). Then, we perform $k + 1$ private key switches on each ct_i to obtain $\{\mathsf{CT}_{i,j}\}^{0\leq j\leq k}_{0\leq i\leq \ell-1}$ such that $\mathsf{CT}_{i,j}$ is encrypting $-m \cdot \frac{q}{\mathfrak{B}^{i+1}} \cdot S'_j$ with the key $\vec{S'}$ and with $S'_k = -1$. The collection of those ciphertexts is a GGSW ciphertext encrypting m.

The circuit bootstrap is described in Algorithm 11. The PBSs are computed using Algorithm 10 and Theorem 14 and the private key switch using Algorithm 3 and Theorem 7. $(\mathfrak{B}_{PBS}, \ell_{PBS})$ represents the decomposition parameters for the PBS, $(\mathfrak{B}_{KSK}, \ell_{KSK})$ for the private key switch and $(\mathfrak{B}_{CB}, \ell_{CB})$ for the final GGSW ciphertext (Definition 9).

Using Theorems 7 and 14, we can find the noise variance after the circuit bootstrap.

Vertical & Horizontal Packing

The authors of TFHE also proposed two operators to evaluate LUTs in a leveled way i.e. without reducing the noise, called *horizontal and vertical packing.* They both take as input a p-bit message msg, encrypted as a list of p GGSW ciphertexts each encrypting one of its bits. They also take as input α LUTs $L_0 = [l_{0,0}, \cdots, l_{0,2^p-1}], \dots L_{\alpha-1} = [l_{\alpha-1,0}, \cdots, l_{\alpha-1,2^p-1}]$: the goal is to compute the result of the evaluation of the LUTs on the input message, i.e., return encryptions of $l_{0,\mathsf{msg}}, \dots, l_{\alpha-1,\mathsf{msg}}$. Both operators use cmuxes (Algorithm 8 and Theorem 12), either as a tree or in a blind rotation, to compute the LWE result (that can be a GLWE in horizontal packing). Horizontal packing

Algorithm 11: $\overline{\overline{\mathsf{CT}}}_{\mathsf{out}} \leftarrow \mathsf{CircuitBootstrap}\left(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \{\mathsf{KSK}_j\}_{j=0}^{k}\right)$

Context:

$$\begin{cases} \vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n : \text{ the input LWE secret key} \\ \vec{s}' = (s'_0, \cdots, s'_{kN-1}) \in \mathbb{Z}_q^{kN} : \text{ the output LWE secret key} \\ \vec{S}' = \left(S'_0, \ldots, S'_{k-1}\right) \in \mathfrak{R}_q^k : \text{ a GLWE secret key} \\ \forall 0 \le i \le k-1,\ S'_i = \sum_{j=0}^{N-1} s'_{i\cdot N+j} X^j \in \mathfrak{R}_q \\ P_{f_i} \in \mathfrak{R}_q : \text{an } r\text{-redundant LUT for } x \mapsto f_i(x) \\ f_i : \mathbb{Z} \to \mathbb{Z} : \text{a function s.t. } f_i(x) = x \cdot \frac{q}{\mathfrak{B}_{CB}^{i+1}}, \forall 0 \le i < \ell_{CB} \\ \mathsf{CT}_{f_i} \in \mathsf{GLWE}_{\vec{S}'}\left(P_{f_i}\right) \subseteq \mathfrak{R}_q^{k+1}\text{: a trivial encryption of } P_{f_i} \\ \forall 0 \le j \le k, g_j : x \mapsto S'_j \cdot x \text{ with } S'_k = -1; \end{cases}$$

Input:

$$\begin{cases} \mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}}(\widetilde{m}) = (a_0, \cdots, a_{n-1}, a_n = b) \in \mathbb{Z}_q^{n+1} \\ \mathsf{BSK} = \left\{\overline{\overline{\mathsf{CT}}}_{bsk,i} \in \mathsf{GGSW}_{\vec{S}'}^{\mathfrak{B}_{PBS},\ell_{PBS}}\left(s_i\right)\right\}_{i=0}^{n-1} : \text{ a bootstrapping key from } \vec{s} \text{ to } \vec{S}' \\ \{\mathsf{KSK}_j\}_{j=0}^{k} = \left\{\left\{\overline{\mathsf{CT}}_{ksk,i} \in \mathsf{GLev}_{\vec{S}'}^{\mathfrak{B}_{ksk},\ell_{ksk}}\left(g_j(s'_i)\right)\right\}_{0 \le i \le n}\right\}_{j=0}^{k} : \text{ a private key switching key} \end{cases}$$

Output: $\overline{\overline{\mathsf{CT}}}_{\mathsf{out}} \in \mathsf{GGSW}_{\vec{S}'}\left(m\right)$ if we respect the requirements of Theorem 14

```
1  begin
       /* $\ell_{CB}$ PBS (Algorithm 10) */
2      for $0 \le i < \ell_{CB}$ do
3          $\mathsf{ct}_i \leftarrow \mathsf{PBS}\left(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \mathsf{CT}_{f_i}\right)$
4      end
       /* $(k+1) \cdot \ell_{CB}$ private key switch (Algorithm 3) */
5      for $0 \le i < \ell_{CB}$ do
6          for $0 \le j \le k$ do
7              $\mathsf{CT}_{i,j} \leftarrow \mathsf{PrivateKS}\left(\mathsf{ct}_i, \mathsf{KSK}_j\right)$
8          end
9      end
10     $\overline{\overline{\mathsf{CT}}}_{\mathsf{out}} \leftarrow \{\mathsf{CT}_{i,j}\}_{0 \le i < \ell_{CB}, 0 \le j \le k}$
11 end
```

Algorithm 12: $\mathsf{CT}_{\mathsf{out}} \leftarrow \mathsf{CMUX{-}Tree}\left(\left\{\overline{\overline{\mathsf{CT}}}_i\right\}_{i=0}^{\log_2(p)-1}, \{l_i\}_{i=0}^{p-1}\right)$

```
Context: $\begin{cases} \vec{S} \in \mathfrak{R}_{q,N}^k : \text{the input GLWE secret key} \\ \ell \in \mathbb{N} : \text{ the number of levels of the decomposition} \\ \mathfrak{B} \in \mathbb{N} : \text{ the base of the decomposition} \\ p = 2^{\log_2(p)} \in \mathbb{N} : p \text{ is a power of two} \end{cases}$
Input: $\begin{cases} \forall 0 \le i \le \log_2(p) - 1, \overline{\overline{\mathsf{CT}}}_i \in \mathsf{GGSW}_{\vec{S}}^{\mathfrak{B},\ell}(m_i) \text{ s.t. } m = \sum_{i=0}^{\log_2(p)-1} m_i \cdot 2^i \\ \{L_i\}_{i=0}^{p-1} : \text{ a collection of lookup tables} \end{cases}$
Output: $\mathsf{CT}_{\mathsf{out}} \in \mathsf{GLWE}_{\vec{S}}(L_m)$
/* Initilization                                                         */
1 for $0 \le i \le p-1$ do
2     $L_i^{(0)} \leftarrow L_i$
/* CMUX tree with Algorithm 8                                             */
3 for $0 \le h < \log_2(p)$ do
4     for $0 \le i \le 2^{\log_2(p)-h-1} - 1$ do
5         $L_i^{(h+1)} \leftarrow \mathsf{CMUX}\left(L_{2i}, L_{2i+1}, \overline{\overline{\mathsf{CT}}}_h\right)$
6 $\mathsf{CT}_{\mathsf{out}} \leftarrow L_0^{\log_2(p)}$
7 return $\mathsf{CT}_{\mathsf{out}}$
```

Algorithm 13: $\mathsf{CT}_{\mathsf{out}} \leftarrow \mathsf{VPLut}\left(\left\{\overline{\overline{\mathsf{CT}}}_i\right\}_{i=0}^{p-1}, \{l_i\}_{i=0}^{2^p-1}\right)$

```
Context: $\begin{cases} \vec{S} \in \mathfrak{R}_{q,N}^k : \text{the input GLWE secret key} \\ \ell \in \mathbb{N} : \text{ the number of levels of the decomposition} \\ \mathfrak{B} \in \mathbb{N} : \text{ the base of the decomposition} \\ N : \text{polynomial size of } \mathfrak{R}_{q,N} \\ p : \text{a power of two} \end{cases}$
Input: $\begin{cases} \forall 0 \le i \le \log_2(p) - 1, \overline{\overline{\mathsf{CT}}}_i \in \mathsf{GGSW}_{\vec{S}}^{\mathfrak{B},\ell}(m_i) \text{ s.t. } m = \sum_{i=0}^{\log_2(p)-1} m_i \cdot 2^i \\ \{L_i\}_{i=0}^{\frac{p}{N}-1} : \text{ a collection of lookup table} \end{cases}$
Output: $\mathsf{CT}_{\mathsf{out}} \in \mathsf{GLWE}_{\vec{S}}(L_m)$
/* CMUX tree (Algorithm 12) on part of the input GGSW ciphertexts        */
1 $\mathsf{CT}_{\mathsf{tmp}} \leftarrow \mathsf{CMUX} - \mathsf{Tree}\left(\left\{\overline{\overline{\mathsf{CT}}}\right\}_{i=0}^{\log_2(p)-N-1}, \{L_i\}_{i=0}^{\frac{p}{N}-1}\right)$
/* Final Blind Rotate (Algorithm 9)                                       */
2 $\mathsf{CT}_{\mathsf{out}} \leftarrow \mathsf{BlindRotate}\left(\left(2^0, 2^1, \cdots, 2^{\log_2(N)-1}, 0\right), \left\{\overline{\overline{\mathsf{CT}}}\right\}_{i=\log_2(p)-N}^{\log_2(p)-1}, \mathsf{CT}_{\mathsf{tmp}}\right)$
3 return $\mathsf{CT}_{\mathsf{out}}$
```

is interesting when many LUTs should be evaluated in parallel, while vertical packing is interesting when a single (large) LUT needs to be evaluated. They are two extremes of a trade-off for the evaluation of homomorphic LUTs and a mixed solution has been proposed generalizing both of them.

In Algorithm 12, we explain how to compute a tree of cmuxes. We assume the message m to be in $\mathbb{Z}_p$ i.e., m can be represented by $\log_2(p)$ bits. Intuitively, a cmux tree is used to select one of the input lookup tables using a list of GGSW ciphertexts encrypting the bits of the message m. On each layer, we select half of the lookup tables with cmuxes using one of the GGSW ciphertexts.

With a toy example, we show how to use a cmux tree to evaluate the identity function on a message in $\mathbb{Z}_4$. For the sake of simplicity, let us assume that the lookup tables have only one non zero coefficient on the constant term and that $L_i = i$ for $0 \leq i \leq p-1$ i.e., each lookup table represents an element of the message space $\mathbb{Z}_p$. Let us assume that $p = 4, m = 2$ i.e., $m_0 = 0$ and $m_1 = 1$. On the first layer, we will select the right lookup tables with a GGSW ciphertext encrypting m_0. We compute $\mathsf{CMUX}\left(L_0, L_1, \mathsf{GGSW}_S^{\mathfrak{B},\ell}(m_0)\right)$, we obtain a GLWE ciphertext CT_0 encrypting $L_{m_0} = L_0 = 0$. We do the same with L_2 and L_3 and obtain another GLWE ciphertext CT_1 of $L_2 = 2$. On the next layer, we compute $\mathsf{CMUX}\left(\mathsf{CT}_0, \mathsf{CT}_1, \mathsf{GGSW}_S^{\mathfrak{B},\ell}(m_1)\right)$ and we obtain CT_1 which is an encryption of 2. Thus, at the end of the cmux tree, we have applied the identity function on the input message m. If one wants to evaluate an arbitrary function, the only modification to do is to encode different values in the lookup tables.

In Algorithm 13, we present the vertical packing algorithm, which is faster than a simple cmux tree under certain conditions. Intuitively, instead of evaluating the cmux tree with every GGSW ciphertexts encrypting bits of the message, we do it on a subset of the GGSW ciphertexts. After the cmux tree, we have an encrypted lookup table and we need to select the right coefficient in this lookup table with a blind rotate and the rest of the GGSW ciphertexts.

Regarding the noise, in a cmux tree, the noise behaves in the same way as in a chain of cmuxes. Therefore, to estimate the noise of the vertical packing, one can directly use the analysis of Appendix A.2 to come up with the formula.

Multi-Output PBS [CIM19]

A multi-output version of the PBS is described in [CIM19] allowing the evaluation of multiple (negacyclic) functions $\{f_i\}_i$ over one encrypted input. Each function f_i is encoded as a LUT in a polynomial P_i. One can find a shared polynomial Q such that we can decompose each P_i as $Q \cdot P_i'$. This operation is called $\mathsf{PolyFactor}$ and so we have $(Q, \{P_i'\}) = \mathsf{PolyFactor}(\{P_i\})$. Then we compute $\mathsf{CT}_{\mathsf{out}} \leftarrow \mathsf{PBS}(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, Q)$ and multiply $\mathsf{CT}_{\mathsf{out}}$ by each P_i' and sample extract the resulting ciphertexts. One would have obtained the evaluation of each function.

One drawback of this method is that the noise inside the i-th output ciphertext depends on P_i' and is bigger than the output noise of a bootstrap. We will see later on (Chapter 4) how to compare FHE algorithms that have different output noises. To find the variance of the noise in Algorithm 14, we can use the formula in Theorem 14 and in Theorem 4. To prevent having a dependency on the lookup table in the variance, we can compute an upper bound of the values in P_i' as done in [CIM19].

Remark 10 (Encrypted Lookup Table) *The PBS (Algorithm 10 and Theorem 14) works on an encrypted GLWE ciphertext CT_f that can be a trivial encryption (Definition 7) or a classical GLWE ciphertext (Definition 5).*

For Algorithm 14, it is not possible to use an encrypted LUT. The algorithm only works with a trivial encryption of Q and P_i'.

Tree-PBS [GBA21]

A recent paper [GBA21] revisits the bootstrapping. It gives two algorithms, the tree-PBS and the chain-PBS and a few optimizations to perform programmable bootstrapping on large-precision ciphertexts encrypting one message decomposed in a certain base. Those algorithms could be used to homomorphically compute multivariate functions if we call them with the appropriate lookup tables.

The key idea in those algorithms is to split the message inside several ciphertexts using a radix encoding (see Section 2.4). We assume that an encoded value $\widetilde{m}$ of a message $m \in \mathbb{Z}_p$ is defined as $\widetilde{m} = \Delta_{\mathsf{in}} \cdot m$ with $\Delta_{\mathsf{in}} = \frac{q}{2p}$. Then, it uses the [CIM19]-PBS in combination with packing key switch and bootstrap to evaluate the function over the message. The algorithm in the special case where the message is only split in two is summarized in Algorithm 15. The variance of the noise can be found by composing the variance of Theorems 7 and 14.

Algorithm 14: $\{\mathsf{ct}_{\mathsf{out},i}\}_{i=0}^{\alpha-1} \leftarrow \mathsf{CIM19{-}PBS}\left(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \{P_i\}_{0\leq i\leq \alpha-1},\right)$

Context: $\begin{cases} \vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n : \text{ the input LWE secret key} \\ \vec{s'} = (s'_0, \cdots, s'_{kN-1}) \in \mathbb{Z}_q^{kN} : \text{ the output LWE secret key} \\ \vec{S'} = \left(S'_0, \ldots, S'_{k-1}\right) \in \mathfrak{R}_q^k : \text{ a GLWE secret key} \\ \forall 0 \leq i \leq k-1,\ S'_i = \sum_{j=0}^{N-1} s'_{i\cdot N+j} X^j \in \mathfrak{R}_q \end{cases}$

Input: $\begin{cases} \mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}}(\widetilde{m}) = (a_0, \cdots, a_{n-1}, a_n = b) \in \mathbb{Z}_q^{n+1} \\ \mathsf{BSK} = \left\{\overline{\overline{\mathsf{CT}}}_i \in \mathsf{GGSW}_{\vec{S'}}^{\beta,\ell}(s_i)\right\}_{i=0}^{n-1} : \text{ a bootstrapping key from } \vec{s} \text{ to } \vec{S'} \\ \{P_i\}_{0\leq i\leq \alpha-1} \in \mathfrak{R}_{q,N}^{\alpha} : \text{ polynomial of encoded values} \end{cases}$

Output: $\mathsf{ct}_{\mathsf{out},i} \in \mathsf{LWE}_{\vec{s'}}(P_i[m]), \forall 0 \leq i \leq \alpha-1$ if we respect the requirements of Theorem 14

```
1 begin
      /* Factorize the lookup table                                      */
2     $Q, \{P'_i\}_{i=0}^{\alpha-1} \leftarrow \mathsf{PolyFactor}\left(\{P_i\}_{i=0}^{\alpha-1}\right)$
      /* PBS with Algorithm 10                                           */
3     $\mathsf{CT}_{\mathsf{tmp}} \in \mathsf{GLWE}_{\vec{S'}}(Q) \leftarrow \mathsf{PBS}(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, (0, \cdots, 0, Q))$
      /* Leveled multiplications and sample extract (Algorithm 5)        */
4     for $0 \leq i \leq \alpha-1$ do
5         $\mathsf{CT}_{\mathsf{out},i} \in \mathsf{GLWE}_{\vec{S'}}(Q \cdot P'_i) \leftarrow \mathsf{CT}_{\mathsf{tmp}} \cdot P'_i$
6         $\mathsf{ct}_{\mathsf{out},i} \leftarrow \mathsf{SampleExtract}(\mathsf{CT}_{\mathsf{out},i}, 0)$
7     end
8 end
```

Algorithm 15: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{Tree{-}PBS}\left(\{\mathsf{ct}_{\mathsf{in},0}, \mathsf{ct}_{\mathsf{in},1}\}, \mathsf{BSK}, \mathsf{CT}_f, \mathsf{KSK}\right)$

```
Context: $\begin{cases} \vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n : \text{ the input LWE secret key} \\ \vec{s'} = (s'_0, \cdots, s'_{kN-1}) \in \mathbb{Z}_q^{kN} : \text{ the output LWE secret key} \\ \vec{S'} = \left(S'_0, \ldots, S'_{k-1}\right) \in \mathfrak{R}_q^k : \text{ a GLWE secret key} \\ \forall 0 \le i \le k-1,\ S'_i = \sum_{j=0}^{N-1} s'_{i \cdot N + j} X^j \in \mathfrak{R}_q \\ P_{f_i} \in \mathfrak{R}_q : \text{an } r\text{-redundant LUT for } x \mapsto f_i(x) \\ m \in \mathbb{Z}_p \end{cases}$
Input: $\begin{cases} \mathsf{ct}_{\mathsf{in},i} \in \mathsf{LWE}_{\vec{s}}(\widetilde{m_i}) \subseteq \mathbb{Z}_q^{n+1}, \forall i \in \{0,1\} \text{ s.t. } m = m_1 || m_0 \\ \mathsf{BSK} = \left\{\overline{\overline{\mathsf{CT}}}_i \in \mathsf{GGSW}_{\vec{S'}}^{\beta,\ell}(s_i)\right\}_{i=0}^{n-1} : \text{ a bootstrapping key from } \vec{s} \text{ to } \vec{S'} \\ \mathsf{KSK} = \left\{\overline{\mathsf{CT}}_i \in \mathsf{GLev}_{\vec{S'}}^{\mathfrak{B}_{ksk},\ell_{ksk}}(s_i)\right\}_{0 \le i \le n} : \text{ a key switching key from } \vec{s'} \text{ to } \vec{S'} \\ \forall 0 \le i \le p-1, P_{f_i} \in \mathfrak{R}_{q,N} : \text{ a collection of lookup tables} \end{cases}$
Output: $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s'}}\left(\widetilde{f(m)}\right)$ if we respect the requirements of Theorem 14
1 begin
      /* First layer of CIM-PBS with Algorithm 14                    */
2     $\{\mathsf{ct}_{\mathsf{tmp},i}\}_{i=0}^{p-1} \leftarrow \mathsf{CIM9{-}PBS}\left(\mathsf{ct}_{\mathsf{in},0}, \mathsf{BSK}, \{P_{f_i}\}_{i=0}^{p-1}\right)$
      /* packing the output of first layer                           */
3     $\mathcal{L} \leftarrow \left(\underbrace{\mathsf{ct}_{\mathsf{tmp},0}, \cdots, \mathsf{ct}_{\mathsf{tmp},0}}_{\frac{N}{p} \text{ elements}}, \cdots, \underbrace{\mathsf{ct}_{\mathsf{tmp},p-1}, \cdots, \mathsf{ct}_{\mathsf{tmp},p-1}}_{\frac{N}{p} \text{ elements}}\right)$
4     $\mathsf{CT}_L \leftarrow \mathsf{PackingKS}\left(\mathcal{L}, \{0, \cdots, N-1\}, \mathsf{KSK}\right)$
      /* second layer with several PBS (Algorithm 10)                */
5     $\mathsf{ct}_{\mathsf{out},i} \leftarrow \mathsf{PBS}\left(\mathsf{ct}_{\mathsf{in},1}, \mathsf{BSK}, \mathsf{CT}_L\right)$
6 end
```

To use Algorithm 15 with more than two inputs, one can still use [CIM19]-PBS for the first layer. The PBS will extract the constant term i.e. the coefficient that is of interest. Then we can use the packing key switch to build p lookup tables. Next, we use the PBS with an encrypted lookup table to select one coefficient for each LUT. Finally, we can use the packing key switch again to build a lookup table and apply a last PBS with an encrypted lookup table. For efficiency reasons, it could be useful to replace the PBS of the second layer by a circuit bootstrap (Algorithm 11) and several external products.

[**LMP21**]-WoP-PBS

One of the constraints of the PBS (second condition in Theorem 14) is to have a message smaller than $\frac{q}{2}$. With this constraint, it is harder to use the PBS with real use cases. This was for long a big restriction. We introduce in Section 6.1 and Algorithms 27 and 28 the first known Without-Padding PBS (WoP-PBS) i.e. a PBS working without condition 14. After our first construction, a WoP-PBS was also introduced in [LMP21] that we describe in Algorithm 16. The variance of the noise after the WoP-PBS of [LMP21] is the same as the variance after a PBS (Theorem 14).

In this thesis, we compare [LMP21]-WoP-PBS with another WoP-PBS that we describe in Section 6.2 and Algorithm 30.

2.3.5 TFHE's Limitations

In the previous sections, we introduced the main building blocks of TFHE. A wide range of operations can be performed using those algorithms but some limitations must be acknowledged. Let us summarize the different existing limitations.

Limitation 1 (Padding Bit in PBS) *We saw in Theorem 14 (Condition 2) that we need the encrypted plaintext to have its Most Significant Bit (MSB) set to zero (or at least known) to perform a correct bootstrap (Algorithm 10). The only exceptions are when the univariate function evaluated is negacyclic or when the message modulus is odd as explained in Remark 8. This limitation prevents us from leveraging the native modulo q and sustains a consistent modular arithmetic over the messages.*

Limitation 2 (Maximal Precision in PBS) *One cannot bootstrap efficiently a message with a large precision (e.g., more than 8 bits). The number of bits of the message we bootstrap is strictly related to the dimension N of the ring chosen for the PBS. This means*

```
Algorithm 16: $\mathsf{ct_{out}} \leftarrow \mathsf{LMP22{-}PBS}\left(\mathsf{ct_{in}}, \mathsf{BSK}, \mathsf{KSK}, f\right)$
  Context:
    $\begin{cases} \vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n : \text{ the input LWE secret key} \\ \vec{s'} = (s'_0, \cdots, s'_{kN-1}) \in \mathbb{Z}_q^{kN} : \text{ the output LWE secret key} \\ \vec{S'} = \left(S'_0, \ldots, S'_{k-1}\right) \in \mathfrak{R}_q^k : \text{ a GLWE secret key} \\ \forall 0 \le i \le k-1,\ S'_i = \sum_{j=0}^{N-1} s'_{i \cdot N + j} X^j \in \mathfrak{R}_q \\ m \in \mathbb{Z}_p \\ \Delta = \frac{q}{p} \\ \beta : \text{ a correction for the noise s.t. } \beta \le \frac{\Delta}{4} \\ f_0 : \mathbb{Z}_{2q} \to \mathbb{Z}_{2q} : \text{ a function s.t. } f_0(x) = \left(q \left\lfloor \frac{x}{q} \right\rfloor - \frac{q}{2}\right) \mod 2q \\ f_1 : \mathbb{Z}_{2q} \to \mathbb{Z}_{2q} : \text{ a function s.t. } f_1(x) = \begin{cases} \alpha f\left(\left\lfloor \frac{x}{\Delta} \right\rceil\right) & \text{if } x < q \\ -\alpha f\left(\lfloor (2q-x)/\Delta \rceil\right) & \text{otherwise} \end{cases} \\ \mathsf{CT}_{f_0} \in \mathsf{GLWE}_{\vec{S'}}(P_{f_0}) : \text{ an encryption of a LUT (Definition 12)} f_0 \\ \mathsf{CT}_{f_1} \in \mathsf{GLWE}_{\vec{S'}}(P_{f_1}) : \text{ an encryption of a LUT } f_1 \end{cases}$
  Input:
    $\begin{cases} \mathsf{ct_{in}} \in \mathsf{LWE}_{\vec{s}}(\widetilde{m}) = (a_0, \cdots, a_{n-1}, a_n = b) \in \mathbb{Z}_q^{n+1} \\ f : \mathbb{Z}_p \to \mathbb{Z}_p : \text{ a function we want to evaluate} \\ \mathsf{BSK} = \left\{\overline{\overline{\mathsf{CT}}}_i \in \mathsf{GGSW}_{\vec{S'}}^{\beta,\ell}(s_i)\right\}_{i=0}^{n-1} : \text{ a bootstrapping key from } \vec{s} \text{ to } \vec{S'} \\ \mathsf{KSK} = \left\{\overline{\mathsf{CT}}_i \in \mathsf{GLev}_{\vec{s}}^{\mathfrak{B}_{ksk}, \ell_{ksk}}(s'_i)\right\}_{0 \le i \le kN-1} : \text{ a key switching key from } \vec{s'} \text{ to } \vec{s} \end{cases}$
  Output: $\mathsf{ct_{out}} \in \mathsf{LWE}_{\vec{s'}}\left(P_f[m]\right)$ if we respect the requirement 14 of Theorem 14
1 begin
      /* Noise correction */
2     $\mathsf{ct_{tmp}} \leftarrow \mathsf{ct_{in}} + (0, \cdots, 0, \beta)$
      /* To increase the message/ciphertext space */
3     $\mathsf{ct_{tmp}} \leftarrow \mathsf{ct_{tmp}} \mod 2q$
      /* PBS with Algorithm 10 */
4     $\mathsf{ct_{tmp}} \leftarrow \mathsf{ct_{tmp}} - \mathsf{PBS}\left(\mathsf{ct_{tmp}}, \mathsf{BSK}, \mathsf{CT}_{f_0}\right)$
      /* Noise correction */
5     $\mathsf{ct_{tmp}} \leftarrow \mathsf{ct_{tmp}} + \beta - \frac{q}{2}$
      /* Keyswitch with Algorithm 1 */
6     $\mathsf{ct_{tmp}} \leftarrow \mathsf{Keyswitch}\left(\mathsf{ct_{tmp}}, \mathsf{KSK}\right)$
      /* PBS with Algorithm 10 */
7     $\mathsf{ct_{out}} \leftarrow \mathsf{PBS}\left(\mathsf{ct_{tmp}}, \mathsf{BSK}, \mathsf{CT}_{f_1}\right)$
8 end
```

that the more we increase the precision, the more we have to increase the parameter N, and the slower the computation ends up.

Limitation 3 (Multi-threaded PBS) *The PBS algorithm is not easily parallelizable. Indeed, it is a series of loops working on an accumulator. Some variants of the PBS [Zho+18; JP22] offer ways to have a multi-threaded PBS but as a drawback, the bootstrapping key size increases exponentially and the noise after this bootstrapping is bigger than with a traditional PBS (Algorithm 10).*

Limitation 4 (Native LWE Multiplication) *There exists no native multiplication between two LWE ciphertexts. Currently, there are two approaches to multiply LWE ciphertexts:*

- *use two programmable bootstrappings (Algorithm 10) to evaluate the function $x \mapsto \frac{x^2}{4}$ so we can build the multiplication $x \cdot y = \frac{(x+y)^2}{4} - \frac{(x-y)^2}{4}$;*
- *use 1 or more TFHE circuit bootstrappings (Algorithm 11) in order to convert one of the inputs into a GGSW ciphertext (if not given as input) and then performing an external product (Algorithm 7).*

Since both techniques make use of the PBS, they both suffer from Limitations 1 and 2.

Limitation 5 (Homomorphic Decomposition) *Because of Limitations 1 and 2, it seems impractical, in an efficient manner, to homomorphically split a message contained in a single ciphertext into several ciphertexts containing smaller chunks of the original message (apart from the trivial, but inefficient, binary case).*

Limitation 6 (Several LUTs in one PBS) *The PBS can evaluate only a single function per call. Using the [CIM19]-PBS (Algorithm 14), we can evaluate multiple lookup tables at the same time, but the output will have an additional amount of noise which depends on the function evaluated.*

Limitation 7 (Gate Bootstrapping) *TFHE's gate bootstrapping represents a very easy solution for evaluating homomorphic Boolean circuits. However, this technique requires a PBS for each binary gate, which results in a costly execution. Furthermore, when we want to apply a similar approach to the arithmetic circuit with larger integers (more than 1 bit), TFHE does not provide a solution.*

Limitation 8 (Circuit Bootstrap) *TFHE's circuit bootstrapping (Algorithm 11) requires ℓ PBSs followed by many key switches, which is significantly time consuming.*

Limitation 9 (GLWE Secret Key Size) *There is no fine-grained control over the size of a GLWE secret key. In a GLWE secret key, there are $k \cdot N$ coefficients with N a power of two and k an integer greater than one. We cannot choose to use an arbitrary number of secret key coefficients.*

Limitation 10 (Noise Plateau) *When encrypting a ciphertext, if we use a noise variance so small that when we draw random noise from this distribution, the noise is most of the time set to 0, we take the risk of having ciphertexts without noise. That is why we enforce a minimal value for the variance of the noise which depends on a given security level and a given ciphertext modulus q (see Section 3.1 for the details).*

So when one increases n (or kN), they eventually reach a plateau in terms of noise variance i.e. there is an LWE dimension $n_{\mathsf{plateau}} \in \mathbb{N}$ such that for any LWE dimension greater than n_{plateau}, we still need to take the same noise variance if we want the security to hold.

In this thesis, we successfully removed the limitations listed above by introducing new algorithms, new tools and new encodings to make TFHE more practical.

2.4 Encodings

In this section, we introduce several ways to encode a message before encrypting it. Given a message, we can encode it into one or several values. First, we explain how to encode a message to be encrypted in a single LWE ciphertext. Then, we show how to encode a single message into several encoded values using the CRT and the radix encodings.

2.4.1 Modular Arithmetic with a Single LWE ciphertext

In [Chi+20a], the authors extensively explained how to use TFHE with Boolean messages, and noticed that it also supports small integer messages i.e. messages with at most 10 bits of precision. In this section, we explain how to generalize the encoding to support small integers.

Before applying the encryption procedure in Definitions 4 and 5, we need first to encode the message. The following definition explains how to do that.

Definition 13 (GLWE Encode & Decode) *Let $q \in \mathbb{N}$ be a ciphertext modulus, and let $p \in \mathbb{N}$ a message modulus, and $\pi \in \mathbb{N}$ the number of bits of padding*[3]*. We have $2^\pi \cdot p \leq q$ and $2^\pi \cdot p$ is the plaintext modulus. Let $M \in \mathfrak{R}_p$ be a message. We define the encoding of M as:*

$$\widetilde{M} = \mathsf{Encode}\left(M, 2^\pi \cdot p, q\right) = \lfloor \Delta \cdot M \rceil \in \mathfrak{R}_q \tag{2.28}$$

with $\Delta = \frac{q}{2^\pi \cdot p} \in \mathbb{Q}$ the scaling factor (see a visual example in Figure 2.1).

To decode, we compute the following function:

$$M = \mathsf{Decode}\left(\widetilde{M}, 2^\pi \cdot p, q\right) = \left\lfloor \frac{\widetilde{M}}{\Delta} \right\rceil \in \mathbb{Z}_{2^\pi \cdot p} \tag{2.29}$$

In practice $\widetilde{M}$ contains a small *error term $E = \sum_{i=0}^{N-1} e_i \cdot X^i \in \mathbb{Q}[X]/(X^N + 1)$, so we can rewrite $\widetilde{M} = \Delta \cdot M + E \in \mathbb{Z}_q$. The decoding algorithm fails if and only if there is at least one $i \in [\![0, N-1]\!]$ such that $|e_i| \geq \frac{\Delta}{2}$. We can note this probability as*

$$\mathsf{Pr}\left(\bigcup_{i=0}^{N-1} |e_i| \geq \frac{\Delta}{2}\right) = \mathsf{Pr}\left(\mathsf{Decode}\left(\widetilde{M}, 2^\pi \cdot p, q\right) \neq M\right). \tag{2.30}$$

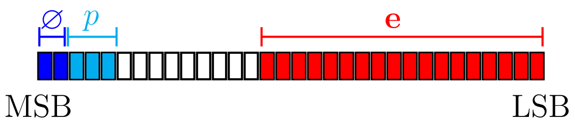


Figure 2.1: Plaintext binary representation with $p = 8 = 2^3$ (cyan), $\pi = 2$ (dark blue) such that $2^\pi \cdot p \leq q$, the error e (red). The white part is empty. The MSB are on the left and the LSB on the right.

The encoding introduced in Definition 13 can be modified in order to include a *carry space* on the left of the plaintext space. The core idea is to have enough room in a ciphertext encrypting an integer message modulo $\beta \in \mathbb{N}$ to store more than just the message but also potential carries coming from leveled operations such as addition or multiplication with a known integer.

To give an example, let us assume that we have two messages $m_0 = 3$ and $m_1 = 2$. m_0 and m_1 can be written on 2 bits as $m_0 = 11_2$ and $m_1 = 10_2$. However, the sum of m_0 and m_1 can not be written on 2 bits as $m_0 + m_1 = 101_2 = 5$. We saw in Theorem 14

3. For simplicity we use a power of 2 for the padding, but this is not a necessary condition.

that we need the message to be less than $\frac{q}{2}$ to have the correct output after a PBS. If we perform several additions, we are at risk of overwriting the padding bit and thus failing the following PBS. To mitigate this risk, we introduce a carry space, a dedicated space to absorb the additional bits of information after performing additions and multiplications (Theorem 1, Theorem 4). Once the carry space is full, we need to bootstrap in order to encode again the message and clean the carry space to provide for future additions.

In practice, we split the traditional plaintext space into three different parts: the *message subspace* storing an integer modulo $\beta \in \mathbb{Z}$ (we call β the base), the *carry subspace* containing information overlapping β, and a bit of padding (or more) often needed for bootstrapping. In this context, we refer to the *carry-message modulo* as the subspace including both the message subspace plus the carry subspace, and we note it $p \in \mathbb{N}$. Figure 2.2 shows a visual example.

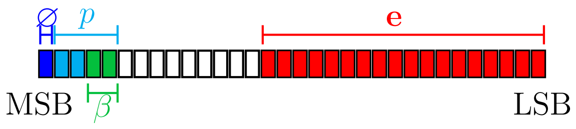


Figure 2.2: Plaintext binary representation with a base $\beta = 4 = 2^2$ (green), a carry subspace (cyan), a carry-message modulo $p = 16 = 2^{2+2}$ (cyan+green) such that $0 < \beta < p$, the error e (red), and a bit of padding is displayed in the MSB (dark blue). The white part is empty. So the plaintext modulo is $32 = 2^{2+2+1}$. This means that we have 2 bits in the carry subspace (set to 0 in a fresh ciphertext), that will contain useful data when one computes leveled operations.

Remark 11 (Boolean Encoding in TFHE) *In [Chi+20a], TFHE is described with Boolean encoding. With the Boolean encoding, we have $\beta = 2, p = 4$ and $\pi = 0$ with π the number of bits of padding as defined in Definition 13.*

The functions evaluated are all negacyclic (Remark 8) so we do not need a bit of padding to guarantee the correctness of the PBS (Theorem 14 and Algorithm 10).

Thanks to the carry bit, we can perform linear combinations between two ciphertexts before a bootstrap. With linear combinations and bootstraps, we can support every logical gate with two inputs.

In order to keep track of the worst-case message in a ciphertext – i.e., check if there is still room to perform more operations – we use a metadata that we call *degree of fullness*.

Definition 14 *The* degree of fullness*, that we note* deg*, of an LWE ciphertext* ct *encrypting a message* $0 \leq m < p$*, is equal to* $\deg(\mathsf{ct}) = \frac{\mu}{p-1} \in \mathbb{Q}$*, where* μ *is the known worst case for* m*, i.e., the largest integer that* m *can be, such that* $0 \leq m \leq \mu < p$*. To ensure correctness, the degree of fullness should always be a quantity included between* 0 *and* 1*, where* $\deg(\mathsf{ct}) = 1$ *means that the carry-message subspace is full in the worst case.*

We take advantage of the carry subspace to compute leveled operations and to avoid bootstrapping. In practice, *the carry subspace acts as a buffer* to contain the carry information derived from leveled operations (additions, multiplication by an integer) and the degree of fullness acts as a measure that indicates when the buffer cannot support additional operations: once this limit is reached the carry subspace is emptied by bootstrapping. To be able to perform a leveled operation between two LWE ciphertexts of that type, they need to have the same base β, carry-message p and ciphertext modulus q. We now list the operators one can compute over such encrypted integers:

- *homomorphic addition* between two encrypted integers (Theorem 1);
- *multiplication by a small integer constant* (Theorem 4);
- *homomorphic opposite*, requiring a correction term;
- *homomorphic subtraction*, composed as an opposite and an addition;
- *homomorphic univariate function evaluation*, computed with the PBS algorithm;
- *homomorphic multivariate function evaluation*, computed by using a trick that was already proposed in [Cle+22]. If the degrees of the ciphertexts allow it, the idea is to concatenate two messages m_1 and m_2 (or more) respectively encrypted in ct_1 and ct_2 by re-scaling the first one with constant multiplication to $\mu_2 + 1$ (where μ_2 is the worst possible value that can be reached by the m_2) and adding it to ct_2 and finally computing a PBS on the concatenation. Once the two messages are concatenated in a single ciphertext, the bivariate LUT L can be simply evaluated as a univariate LUT L' on the concatenation of m_1 and m_2. A visual example is proposed in Figure 2.3;
- *homomorphic multiplication* between two ciphertexts, computed by using the multivariate approach just described, for both the LSB multiplication (i.e. $m_1 \cdot m_2 \mod \beta$) and the MSB multiplication (i.e. $\left\lfloor \frac{m_1 \cdot m_2}{\beta} \right\rfloor$). If instead we want to compute

the multiplication without any modular reduction, we can use well known techniques in TFHE literature such as in [Chi+20b] (see Limitation 4);

- *homomorphic carry/message extraction*, computed through PBS.

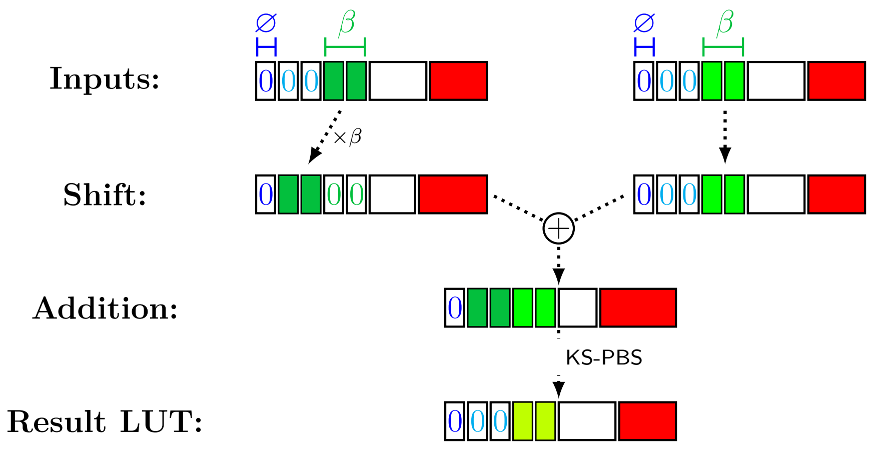


Figure 2.3: Example of a bivariate LUT evaluation with shift, key switch (KS) and PBS.

Remark 12 (Noise Growth) *Generally with FHE, one has to monitor the noise growth in order to guarantee the correctness of successive operations (Definition 13). However, with the concept of carry buffer, we can chose parameters such that the noise is always under a certain level if the degree of fullness has not reached the maximal value allowed. When we approach this value for the degree, a bootstrapping operation is performed and the noise is reduced at the same time. We give more details in Section 7.2.4.*

2.4.2 Modular Arithmetic with Several LWE ciphertexts

In the state of the art, several approaches using many ciphertexts to represent a single message are proposed. We can split these approaches into two main categories: the radix and the CRT (Chinese Reminder Theorem) representations.

The radix representation consists in decomposing a message into several chunks according to a decomposition base. It is very similar to the representation in base 10 we use in our daily lives, where to represent a large number we use several digits. Then the idea is to put each digit into a separate ciphertext and to define the new encryption of the large message as the list of these ciphertexts.

The CRT approach consists in representing a number x modulo a large integer $\Omega = \prod_{i=0}^{\kappa} \omega_i$, where the ω_i moduli are pairwise co-prime, as the list of its residues $x_i = x \bmod \omega_i$. Each residue is then encrypted into a different ciphertext and, as for the radix-based approach, the new encryption of the large message modulo Ω is the list of these ciphertexts.

In order to use these two approaches in TFHE, the digits (for the radix-based approach) and the residues (for the CRT-based approach) need to be quite small (generally less than 8 bits).

The approach of splitting a message into multiple ciphertexts has already been proposed for binary radix decomposition in FHEW [DM15] and TFHE [Chi+20a], and for other representations in [BST20], [GBA21], [KO22], [Cle+22] and [LMP21]. However, none of them takes advantage of carry buffers to make the computations more efficient between multi-ciphertext encrypted integers by avoiding bootstrapping as much as possible. In [GBA21] they propose two approaches to evaluate the PBS over these multi-ciphertexts inputs, namely the *tree-based* (Algorithm 15) and *chained-based* approaches (that we shorten as Tree-PBS and Chained-PBS). The Chained-PBS method is generalized in [Cle+22] to any function in exchange for a larger plaintext space.

The idea of using the CRT approach is mentioned in [KS21] but unfortunately no details are provided. The authors do not change the traditional TFHE encoding to fit the CRT representation. Later on, we provide detailed algorithms to describe the use of the CRT in the plaintext space with two different approaches (with or without carry buffers, along with their respective encoding). Concerning bootstrapping, they describe how to trivially construct polynomials for the blind rotation, which allows them to only evaluate a narrow set of functions (every CRT element being mapped to an output CRT element).

Breaking down a message into several ciphertexts is the first step towards larger precision messages but this method has some limitations. In particular, the radix approach does not allow to represent messages with arbitrary moduli. The modulus has to be selected as a product of a certain base (or bases). We will see in Section 7.2.1 how to remove this constraint. The CRT approach is limited on the maximal number that could be represented, because there exist a very limited amount of primes or co-prime integers smaller than 8 bits.

While arithmetic operations can be evaluated quite straightforwardly for these representations, the bootstrapping and generic LUT evaluation is very inefficient, and the only known technique is the Tree-PBS (Algorithm 15) proposed by [GBA21]. This technique becomes very inefficient as the number of ciphertexts encrypting a single large message increases.

Radix-based large integers

The radix-based approach consists in encrypting a large integer modulo $\Omega = \prod_{i=0}^{\kappa-1} \beta_i$ as a list of $\kappa \in \mathbb{N}$ LWE ciphertexts. Each of the κ ciphertexts is defined according to a pair $(\beta_i, p_i) \in \mathbb{N}^2$ of parameters, such that $2 \leq \beta_i \leq p_i < q$, which respectively corresponds to the message subspace and the carry-message subspace involved with the modular arithmetic, as described in Section 2.4.1. Figure 2.4 gives a visual representation out of a toy example.

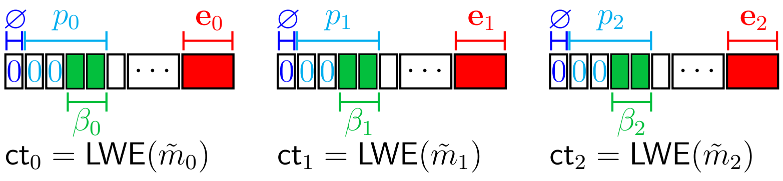


Figure 2.4: Plaintext representation of a fresh radix-based modular integer of length $\kappa = 3$ working modulo $\Omega = (2^2)^3$ with $\mathsf{msg} = m_0 + m_1 \cdot \beta_0 + m_2 \cdot \beta_0 \cdot \beta_1$. The symbol $\varnothing$ represents the padding bit needed for the PBS. For each block we have $\tilde{m}_i = \mathsf{Encode}(m_i, p_i, q)$. For all $0 \leq i < \kappa$ we have $\beta_i = 4$, $p_i = 16$, $\kappa = 3$ and $\Omega = 4^3$.

In practice, the restriction for Ω is that it has to be a product of small bases. Indeed, TFHE-like schemes do no scale well when one increases the precision, so the best practice is to keep $p_i \leq 2^8$.

Definition 15 (Radix Encoding) *To* encode *a message* $\mathsf{msg} \in \mathbb{Z}_\Omega$*, one needs to decompose it into a list of* $\{m_i\}_{i=0}^{\kappa-1}$ *such that*

$$\mathsf{msg} = m_0 + \sum_{i=1}^{\kappa-1} m_i \cdot \left(\prod_{j=0}^{i-1} \beta_j \right). \tag{2.31}$$

Then we can independently call the Encode *function (Definition 13) on each* m_i*. We have* $\tilde{m}_i = \mathsf{Encode}(m_i, 2^\pi \cdot p_i, q)$ *with* π *the number of bits of padding.*

Finally we can encrypt each $\tilde{m}_i$ *into an LWE ciphertext using Definition 4.*

To decode*, we simply recompose the integer from the* m_i *values using Equation* (2.31)*.*

To compute operations over radix-based integers, we must be sure that the messages are encoded and encrypted with the same parameters. The majority of the arithmetic operations can be computed by using a schoolbook approach (homomorphically mixing linear operations and PBS) and by keeping the degree of fullness (Definition 14) smaller than 1 in each block. When carries are full, they need to be propagated to the next block: this is done by extracting the carry and the message and adding the carry to the next block.

When it comes to computing a generic LUT over a radix-based modular integer, the only known approach from the literature is the Tree-PBS from [GBA21], which becomes more and more inefficient as the number of blocks goes increasing.

CRT-based large integers

The CRT-based approach consists in encrypting a large integer modulo $\Omega = \prod_{i=0}^{\kappa-1} \beta_i$ as a list of κ LWE ciphertexts, and we require each pair β_i and $\beta_{j\neq i}$ of bases to be co-prime. Each of the κ ciphertexts is defined according to a pair $\{\beta_i, p_i\}_{0\leq i<\kappa}$ such that $2 \leq \beta_i < p_i < q$.

Definition 16 (CRT Encoding) *Let q be a ciphertext modulus. In order to* encode *a message $\mathsf{msg} \in \mathbb{Z}_\Omega$, one needs to compute $\{m_i\}_{i=0}^{\kappa-1}$ such that*

$$\forall\ 0 \leq i < \kappa, \mathsf{msg} = m_i \mod \beta_i \tag{2.32}$$

with $2 \leq \beta_i < p_i < q$ and p_i as defined in Definition 13. We have $\mathsf{Encode}_{\mathsf{CRT}}(\mathsf{msg}) = (m_0, \cdots, m_{\kappa-1})$.

Then we can independently encode (Definition 13) and encrypt (Definition 4) each m_i into an LWE ciphertext. To decode*, we simply need to compute the modular reduction in base β_i and perform a CRT recombination.*

With this CRT encoding, we have to empty the carry buffers when they are (almost) full. Indeed, when using TFHE's PBS, the bit of padding needs to be preserved. We only need to call the message extraction algorithm described in Section 2.4.1 when needed.

All the arithmetic *operations* can be performed independently on the blocks by using the operators described in Section 2.4.1. Concerning the evaluation of a LUT, the only known way in the literature to compute it on CRT-based large integers, is the technique proposed by [KS21], which can be used only when the LUT to evaluate is CRT-friendly.

A CRT-friendly LUT is a LUT L that can be independently evaluated in each component, i.e., L such that $\mathsf{Encode}_{\mathsf{CRT}}\left(L\left(\mathsf{msg}\right)\right) = \left(L_0\left(m_0\right), \cdots, L_{\kappa-1}\left(m_{\kappa-1}\right)\right)$. For generic LUT evaluations, once again, the only technique known in the literature is the Tree-PBS by [GBA21].

Native CRT. In TFHE, we can also encode CRT integers by using no padding bit and no carry buffer (so no degree of fullness either), and by encoding the message m_i as $\left\lfloor \frac{q}{\beta_i} \cdot m_i \right\rceil$. By doing so, additions and scalar multiplications become native and do not require any PBS, except for noise reduction. To compute additions one can use the LWE addition on each residue, and to compute a scalar multiplication by α, one can decompose α with the CRT basis into smaller integers, and compute scalar multiplications with them. Without the bit of padding, the PBS can be evaluated only with a WoP-PBS algorithm. However, to evaluate a generic LUT, the problem is still open. We will provide a solution in the next sections.

PBS with p Not a Power of Two. No details, nor analysis have been provided yet in the literature about computing a PBS when the plaintext space is not a power of two. We bring some clarity to this question in this paragraph.

When one wants to compute a traditional PBS evaluating a non-negacyclic function, it is required to have a bit of padding, which forces the plaintext space to be even. However the algorithm works the same with an odd p, the only difference lies in the way the r-redundant LUT is built (Definition 12). This also brings a slight modification in the evaluation of the error probability when computing such PBS.

Theorem 15 (Non-Power-Of-Two Lookup Table) *If we use a non-power-of-two encoding, we need to modify the definition of the lookup table. Let $L : \mathbb{Z}_p \to \mathbb{Z}_{p'}$ be the lookup table we want to evaluate such that $x \mapsto L[x] = y_x$. We define the r-redundant lookup table $\tilde{L}$ represented as a polynomial as:*

$$\tilde{L} = X^{-\left\lfloor \frac{N}{2\cdot p} \right\rceil} \cdot \left(\sum_{i=0}^{N-1} \left\lfloor \frac{q}{p'} \cdot y_{\left\lfloor \frac{i\cdot p}{N} \right\rfloor} \right\rceil \cdot X^i \right) \tag{2.33}$$

Proof 15 (Theorem 15) *With such a LUT, and p not a power of 2, we end up with two possible sizes for the mega-cases of the r-redundant LUT: either $\left\lfloor \frac{N}{p} \right\rfloor$ or $\left\lceil \frac{N}{p} \right\rceil$. For the correctness study, we will take the worst case scenario, i.e., considering $\left\lfloor \frac{N}{p} \right\rfloor$. The encoding function (Definition 13) enables to have messages centered in the mega-cases* *when it comes to PBS, it means that the probability of going into the wrong mega-case during a PBS in the worst-case scenario is when the error* e_{MS} *is greater in absolute value than* $\left\lfloor \frac{N}{p} \right\rfloor$ *where* e_{MS} *is the error in the PBS after the modulus switch and before the blind rotation. Thanks to noise formulae, it is easy to estimate the variance of* e_{MS}*. Since it is close to a Gaussian distribution, we can use a confidence interval to infer the probability to get into the wrong mega-case.* □

Limitations

The radix and CRT approaches discussed in this section are a first step towards solving the precision problem in TFHE-like schemes. However, they come with some limitations that are listed below.

Limitation 11 (Limited Choice of Radix Modulus) *The radix approach is limited to the modulo* Ω *that can be expressed as a product of bases. If the modulus is for instance a large prime, no solution is known.*

Limitation 12 (Limited Choice of CRT base) *The CRT approach suffers from the CRT requirements, i.e., co-prime bases, and the precision limitation we have in practice with TFHE (Limitation 2). Indeed, there are a limited number of primes between* 2 *and* 128*. It means that this approach is good when* Ω *is composed of small enough co-prime factors but for the rest of the possible* Ω *we need other solutions.*

Limitation 13 (Generic LUT with Radix & CRT Encoding) *For both the radix and the CRT approach, the only way to evaluate a generic LUT is the Tree-PBS, which does not scale well with the number of blocks, and so it is still inefficient in practice.*

In Sections 6.2 and 7.2, we provide solutions to overcome all these limitations.

2.5 Optimization for FHE

In Section 2.2, we saw that we needed to select some parameters to encrypt ciphertexts. For instance, we need to pick a GLWE dimension k and a polynomial size N to encrypt a GLWE ciphertext (Definition 5). In Section 2.3, we introduced some FHE algorithms and some of them come with additional degrees of freedom that need to be set. For instance in Algorithm 1, we need a decomposition base $\mathfrak{B}$ and a level of decomposition ℓ. In the different theorems, we saw that those parameters influence both the cost of the operation and the noise growth (see Theorem 6). For lack of an alternative, those parameters are chosen by hand which makes it hard to guarantee correctness and efficiency.

In [Via+22], the authors distinguished four types of optimization:

Program Transformation it consists in mapping a program to an FHE program following some FHE programming rules (for instance, removing every *if-then-else* branching). In this step, we can leverage the batching capabilities of the cryptographic scheme when available.

Circuit Optimization once we have an FHE compatible graph, we can apply several transformations to it to reduce the number of some FHE operations (for instance, reducing the multiplicative depth of the circuit).

Cryptographic Optimization it consists in choosing the GLWE (or LWE) instance that we will use and more generally set every degree of freedom available inside the FHE circuit.

Target Optimization once we have a fully-parametrized circuit, we can compile it into machine code, leveraging every feature of the target machine (for instance, using AVX512 to speed up operations).

In the literature, a few compilers [Dat+20][GKT22] for FHE schemes have been proposed: they mainly optimize the circuit to make it as FHE friendly as possible i.e. they focus on *program transformation* and *circuit optimization.*

We try to solve a different problem by focusing on *cryptographic optimization* and we aim to provide a generic approach to automatically select the best parameters according to a given cost model for an arbitrary graph of FHE operators while guaranteeing correctness and security. To the best of our knowledge, no one has ever presented a result on the optimization of parameters for an FHE scheme including a bootstrapping with a flexibility towards multi-precision plaintexts before our contribution [Ber+23a].

We suggest this paper [VJH21] for more information and for a comparison of all existing FHE compilers.

2.5.1 TFHE

Cingulata [CDS15] is the first attempt at a compiler for TFHE and was proposed in 2015. It takes an arithmetic circuit described in C++ and translates it into a Boolean circuit.

Then, the Boolean circuit is converted into an FHE circuit composed of gate bootstraps as described in TFHE [Chi+20a] (we give details on this approach in Section 7.1.1). Cingulata uses the TFHE library [Chi+16b] to execute the circuit with a hardcoded parameter set. Therefore, it does not try to find optimal parameters for the circuit at hand.

Another framework called *Encrypt-Everything-Everywhere* (E3) was introduced in [Chi+18]. Like Cingulata, it takes C++ code as input and supports TFHE but does not provide any parameters. The user is responsible for selecting adequate parameters.

In [Gor+21], the authors introduce a transpiler in the same spirit as Cingulata. The C++ code is translated to a Boolean circuit and various optimization techniques to simplify it are applied. Once again, the parameters used were the one of the TFHE library [Chi+16b].

Concurrently to the work we did (see Chapter 4), in [Kle22], Klemsa proposes an approach to automatize the setup of parameters for TFHE with particular attention in efficiently using resources during the bootstrapping step. Klemsa noticed a link between the size of the public materials (key switch keys and bootstrap keys) and the running time of a computation. Using the size of the public keys as a surrogate of the execution time, he was able to reach good improvements on simple examples. This tool allows to find parameters for a graph composed of additions (Theorem 1), multiplications by integers (Theorem 4), one key switch (Theorem 6) and one PBS (Theorem 14).

2.5.2 Other Schemes

The majority of existing FHE optimizers target other homomorphic schemes than TFHE. Each scheme comes with its own set of constraints and features and those are reflected in the optimization methods.

Schemes such as BGV, B/FV or HEAAN have the tendency to avoid bootstrapping in favor of leveled operations (additions, multiplications). This is due to the fact that these schemes are parametrized with a fixed number of levels, and every multiplication consumes one of the levels. Once all the levels are consumed, no more multiplications can be performed and decryption or bootstrapping is required. The bootstrapping in those schemes is different than what we describe as a bootstrapping in TFHE, it will not reduce the noise but add more available levels. In this context, there exists a line of work aiming to reduce the multiplicative depth of the circuit to be homomorphically evaluated [CAS17; ACS20; Lee+20a] by those schemes. This is not an approach used in TFHE-like schemes, where the multiplicative depth is not a measure taken into account, since the non-linear operations are performed by using a bootstrapping.

Schemes like HEAAN [Che+17] have batching and SIMD capabilities that TFHE does not have. Informally, it allows to evaluate a circuit on different inputs in one run. Thanks to that, those schemes can achieve very good amortized execution times. Those features influence the way the optimization problem is tackled. During the optimization, one of the goal is to maximize the batching capacity to have an amortized execution as small as possible. As TFHE does not support these features, this is not a goal during the selection of the parameters.

The noise management for HEAAN is quite different than the noise management for TFHE, the former providing approximate results. By definition, HEAAN is not designed to deal with exact computations: the encoding itself does not allow exact representations of messages. Furthermore, to evaluate a function with HEAAN, one needs to approximate it with polynomials. If one can settle for approximate arithmetic, those schemes are very efficient. TFHE has been built with another objective in mind: doing efficient and exact arithmetic.

2.5.3 Limitations

To sum up this study of optimization techniques for FHE, we can define the two following existing limitations.

Limitation 14 (Automatic Parameter Selection) *No tools exists that automatically sets the degrees of freedom needed to perform computations over ciphertexts. A user needs to manually choose a (G)LWE instance and its associated parameters (GLWE dimension, polynomial size, LWE dimension). He also needs to choose the degrees of freedom available in most of the algorithms, for instance the decomposition base* $\mathfrak{B}$ *and the maximum level of the decomposition* ℓ *in Algorithm 1.*

Limitation 15 (Comparing State-of-the-Art FHE Algorithms) *In Section 2.3.4, we introduced alternatives to TFHE's PBS (Algorithm 10). Those variants have a different cost and noise trade-off than TFHE's PBS i.e., for a given set of parameters, the noise of the output ciphertext will be different than the noise after a PBS. Similarly, for the same parameters, the cost of computing a PBS or one of its variant will not be the same.*

Due to these different noise/cost trade-offs, we cannot easily compare FHE algorithms.

Chapter 3

Noise Methodology

As explained in Sections 2.1 and 2.2, a ciphertext contains some randomness called *noise* which is needed for security. In particular, the noise distribution must be carefully picked. In this thesis, we consider the noise distribution to be a *discrete centered Gaussian distribution*, therefore we only need to express the noise variance to fully characterize the noise distribution.

Originally, the LWE problem, recalled in Definition 1, was introduced with a continuous Gaussian distribution [Reg05] and later extended to discrete Gaussian distribution. Sometimes, to simplify the proofs, we use continuous Gaussian distributions instead of discrete Gaussian distributions. This holds as long as the standard deviation of each discrete Gaussian distribution is greater than the *smoothing parameters* introduced in [Duc13, Definition 3.13].

To choose the variance of the Gaussian distribution, we rely on the lattice estimator [APS15] described in Section 2.1.3. Once we have chosen this noise variance, we still need to track the noise distribution throughout an FHE computation because when performing computations over ciphertexts, the noises evolve and the value of this noise is closely linked to the correctness of the computation as explained in Definition 13.

In this chapter, we explain how to build a new tool relying on the lattice estimator to find the minimal noise variance at encryption time to guarantee a security level λ, given a ciphertext modulus q, an LWE dimension n, a noise distribution and a secret key distribution. Then, we introduce a key concept in FHE that we call a noise model (Definition 20) that allows to track the noise distribution throughout a computation. Finally, we explain how to improve the bootstrapping theoretical noise formula by taking into account the noise introduced by the use of the FFT within the algorithms.

3.1 Security Oracle

The *security* of a GLWE-based scheme depends on the distribution of the secret key (for example binary, ternary or Gaussian with a variance σ^2), the product between the GLWE dimension and the polynomial size (i.e., $n = k \cdot N$), the noise distribution, and the ciphertext modulus (often denoted by q).

To estimate the security level offered by some given parameters one can use the lattice estimator [APS15] which is the reference tool used to estimate the security of a lattice-based cryptographic scheme (see Section 2.1.3). This tool aims to help researchers to quickly find secure parameters by taking into account well-known attacks, e.g., the primal and dual attacks, the Coded-BKW attacks and the Arora-GB attack on LWE. More details on the different attacks are given in Section 2.1.

First we will give a code example that uses the lattice estimator to estimate the security of an LWE instance and explain why it is not suited for our needs. Finally, we will explain how to build what we call a security oracle (Definition 17) using the lattice estimator.

3.1.1 Motivation

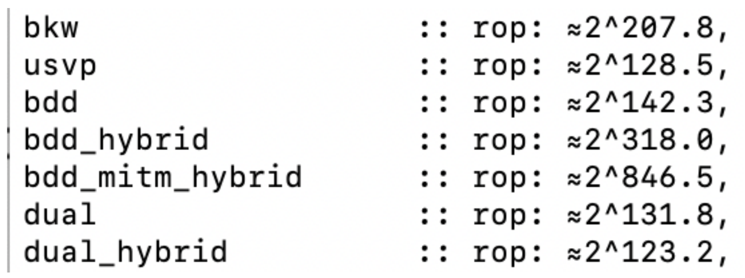
In code example 3.1, we give a simple script calling the lattice estimator to estimate the security of an LWE instance with the ciphertext modulus $q = 2^{64}$, binary secret key coefficients, a centered Gaussian noise distribution χ_σ with standard deviation $\sigma = 2^{-18} \cdot q$ and with an LWE dimension $n = 750$. Executing the code example 3.1 outputs the estimated security of the LWE instance for several attack models. The security level λ is the minimum of those predicted security levels. In the example below, we found that the given instance has 123 bits of security. The useful part of the output of the lattice estimator is given in Figure 3.1[1].

```
from estimator import *
from estimator.lwe_parameters import LWEParameters
from estimator.nd import NoiseDistribution, stddevf

TFHE_new_parameter_set = LWEParameters(
    n=750,
    q=2 ** 64,
```

1. Estimated with the commit cf36315e7718b1e2e3de271b705697943ebaecf4, 9 of May 2023

```
    Xs=NoiseDistribution.UniformMod(2),
    Xe=NoiseDistribution.DiscreteGaussian(stddev=2 ** (64 - 18))
)

LWE.estimate(TFHE_new_parameter_set, red_cost_model = RC.BDGL16)
```

Code Example 3.1: Call to the lattice-estimator to estimate the security of a given LWE instance



```
bkw                   :: rop: ≈2^207.8,
usvp                  :: rop: ≈2^128.5,
bdd                   :: rop: ≈2^142.3,
bdd_hybrid            :: rop: ≈2^318.0,
bdd_mitm_hybrid       :: rop: ≈2^846.5,
dual                  :: rop: ≈2^131.8,
dual_hybrid           :: rop: ≈2^123.2,
```

Figure 3.1: Output of code example 3.1

As said above, this tool is crafted to find the security level of a given LWE instance. Later, we will want to automatically find secure cryptographic parameters and, to do so, we need a slightly different tool. Instead of defining an LWE instance and checking its security, we want to choose a security level, partially define an instance and use the tool to set the remaining degrees of freedom such that the LWE instance has at least the targeted security level. More formally, assuming a secret key distribution (binary, for example) and a noise distribution (Gaussian, for example), the lattice-estimator provides a function f such that

$$f : (q, n, \sigma^2) \to \lambda,$$

with q, a ciphertext modulus, n, an LWE dimension, σ^2, a variance of the Gaussian noise distribution at encryption time and λ, the security level for an LWE instance with (n, σ^2, q). For our purpose, we need a function g that takes the ciphertext modulus q, the LWE dimension n and the security level λ and outputs the minimal variance σ^2 of the Gaussian noise distribution at encryption time required for the LWE instance to have at least a security level λ. Formally, we want g such that

$$g : (q, n, \lambda) \to \sigma^2.$$

We can find the right function g for different secret key and noise distributions. In the following, those functions are called security oracles.

Definition 17 (Security Oracle) *Given a ciphertext modulus q, the product $n = k \cdot N$, a level of security λ, a noise distribution and a secret key distribution, the security oracle outputs the minimal noise variance σ^2_{min} needed in GLWE encryption for it to be secure with the required level of security λ. More formally, for a given noise distribution χ_e and a secret key distribution χ_s, we define the function $\Sigma^{\chi_e,\chi_s}_{\mathsf{min}}$ such that*

$$\Sigma^{\chi_e,\chi_s}_{\mathsf{min}} : (q, k \cdot N, \lambda) \to \sigma_{\mathsf{min}}$$

When the noise distribution is a Gaussian distribution and the secret key distribution is a binary distribution, we simply note this function Σ_{min}.

When instantiating a new LWE (or GLWE) instance, we can use those security oracles to find the minimal variance needed at encryption time for a given ciphertext modulus q and LWE dimension n (respectively GLWE dimension k and polynomial size N) to guarantee the security.

Remark 13 (Relationship between the LWE Dimension and the Noise Variance) *As a general rule of thumb, to keep the same security level when increasing the product $k \cdot N$, we can decrease the minimal noise needed inside a ciphertext. Following this intuition, with a fixed ciphertext modulus q and a fixed security level λ, the function $h(n) = \Sigma_{\mathsf{min}}(q, n, \lambda)$ should be a decreasing function.*

In the subsections below, we will explain how to build a security oracle using the lattice-estimator. An open-source implementation of this method is available online[2].

3.1.2 Method

The easiest way to build Σ_{min} would be to reverse the way the lattice-estimator estimates the security of an LWE instance and find an analytic formula linking the ciphertext modulus, the LWE dimension and the minimal variance of the noise. The issue is that the output of the lattice-estimator is the result of non-trivial optimizations and there is no existing analytical formulae linking those parameters.

2. `https://github.com/zama-ai/concrete/tree/main/tools/parameter-curves`, commit *3a884f5* made on the eighth of August 2023

By doing a reasonable amount of queries to the lattice-estimator, we can create our own analytical formulae to approximate the relationship between the LWE dimension, the ciphertext modulus and the minimal noise variance at encryption time.

Acquisitions

Let λ be a target security level, q a ciphertext modulus, χ_e a noise distribution and χ_s a secret key distribution.

First, we define a set of standard deviations we want to consider. In our work, we took all the powers of two smaller than the ciphertext modulus q i.e. $\sigma \in \{2^1, 2^2, \cdots, 2^{\lfloor \log_2(q) \rfloor}\}$. For each of these standard deviations σ, we are looking for the smallest LWE dimension n such that $f(q, n, \sigma^2) \geq \lambda$.

To do so, we initialize a guess value for the LWE dimension and call the lattice-estimator to estimate the security. If the security is below the target security level, we increase the LWE dimension and start again. On the contrary, if the security is above the target security level, we decrease a bit the LWE dimension. The algorithm stops when the estimated security level is close enough to the requested λ.

We can repeat this process for different noise distributions, different secret key distributions and different security levels.

Remark 14 (Choosing the Guess Value) *If we have an outdated noise oracle and we want to correct it with the latest state-of-the-art attacks, we can still use it to find a guess value. On the contrary, if this is the first time building a noise oracle, we can pick the guess value at random or take an LWE dimension found previously for a greater or smaller standard deviation or for a different security level.*

In the code example 3.1, we found that the LWE instance is 123 bits secure. To find parameters for 128 bits of security, we increase the LWE dimension, for instance $n \leftarrow n+8$ and re-run the estimation. With this new dimension, we reach 124.5 bits of security, which is still not enough. We can continue the algorithm until we eventually reach $n = 780$ which gives $\lambda = 128.2$ bits of security.

Fitting

Once we have lots of tuples $(n, q, \lambda, \sigma^2)$, we do some data visualization to find the form of the security oracle formula we were looking for. We plot the LWE dimension n on the x-axis and $\Sigma_{\mathsf{min}}(q, n, \lambda)$ on the y-axis. Looking at the curves, we find the following formula:

$$\Sigma_{\mathsf{min}}(q, n, \lambda) = 2^{-\alpha \cdot n + \beta} \tag{3.1}$$

with α and β two real numbers that need to be set. The formula correctly models the security oracle with one notable exception: when the noise is so small that it can no longer be represented as an integer. In fact, if the noise is small enough, the security level drops rapidly to zero as a very small noise variance would lead to noise-less ciphertexts which is not secure at all. To mitigate this risk, we will later enforce the minimal standard deviation to be at least $\sigma_{\mathsf{min}} = \frac{2^2}{q}$. Therefore, we do not need to do the acquisitions of tuples $(n, q, \lambda, \sigma^2)$ where $\sigma < \sigma_{\mathsf{min}}$ (or we can remove them before doing the fitting). We choose to enforce the noise standard deviation to be greater than σ_{min} to be above the limit used by other schemes. In [GHS12], the authors use a standard deviation of $\frac{3.2}{q}$ following the analysis from [MR09a]. By setting $\sigma_{\mathsf{min}} = \frac{2^2}{q}$, we have $\frac{3.2}{q} < \sigma_{\mathsf{min}}$ which guarantees that our LWE or GLWE instances will at least be as secure as the ones chosen by other schemes.

We can then perform a *least square polynomial fitting* using NumPy [Har+20] to find the best (α, β). Formally, given a list of acquisitions $\{(n_i, q, \lambda, \sigma_i^2)\}_{i \in I}$ with I a list of indexes, we solve the following minimization problem:

$$(\alpha^*, \beta^*) = \min_{(\alpha,\beta)\in\mathbb{R}^2} \sum_{i \in I} \left(2^{-\alpha \cdot n_i + \beta} - \sigma_i\right)^2 \tag{3.2}$$

Once we find the best (α, β), we are almost done. The final formula for $\Sigma_{\mathsf{min}}(q, n, \lambda)$ needs to take into account the remark above about very small noises. It gives the following formula

$$\Sigma_{\mathsf{min}}(q, n, \lambda) = \max\left(2^{-\alpha \cdot n + \beta} \cdot q, 2^2\right) \tag{3.3}$$

with α and β defined as in Equation (3.2). We provide in Table 3.1 the best (α, β) values for $\lambda \in \{80, 112, 128, 192\}$, $q = 2^{64}$, binary secret key coefficients and Gaussian noise distribution. The fitting was done using the lattice estimator on the commit made on January 5, 2023[3].

Remark 15 *In the process above, we stop when finding an LWE dimension n such that $f\left(q, n, \sigma^2\right) \geq \lambda$. In practice, we may want to add a safety margin $\lambda + \Delta_\lambda$ to prevent current parameters from being outdated too soon. During the course of this work, the*

3. `https://github.com/malb/lattice-estimator/tree/f9f4b3c69d5be6df2c16243e8b1faa80703f020c`

λ	α	β
80	0.04045822621883835	1.7183812000404686
112	0.029881371645803536	2.6539316216894946
128	0.026599462343105267	2.981543184145991
192	0.018894148763647572	4.2700349965659115

Table 3.1: Best values of (α, β) for $\lambda = 128$, $q = 2^{64}$, binary secret key coefficient and Gaussian noise distribution

security oracles were built at least three times to take into account updates of the lattice estimator.

Verification

Once we have a security oracle fully determined thanks to the method explained above, we can run several tests to make sure that our security oracle is sound. The most straightforward test is to compare the variances predicted by the security oracles with the acquisitions obtained previously. Finally, we can define a list of standard deviations and LWE dimensions and compare the security oracle predictions against those of the lattice estimator. Those verifications can be launched manually or automatically triggered each time a commit is pushed on the lattice estimator repository.

Remark 16 (Noise Plateau (Limitation 10)) *In Limitation 10, we introduced the concept of a noise plateau. Using Equation* (3.1) *and Table 3.1, we find* $n_{\mathsf{plateau}} = 2443$ *for* 128 *bits of security and* $q = 2^{64}$.

3.2 Noise Model

In the FHE world, a small randomness called noise is embedded inside the ciphertexts to guarantee security as we saw in Section 3.1. We also saw in Definition 13 that in order to correctly decode an encoded message $\widetilde{M}$, we should guarantee the noise to be less than $\frac{\Delta}{2}$ where Δ is the scaling factor. If this inequality is not verified, the decoding algorithm fails and we are not able to recover the message. The issue is that the exact value of this noise must remain a secret to prevent an attacker from removing it from the ciphertext. If an attacker is able to do that repeatedly, the secret key can be recovered using linear algebra techniques.

On the contrary, the distribution of the noise can be publicly known without impacting the security which is why, instead of tracking the exact value of the noise, we track the distribution of this noise. To check that the noise is below the threshold, from the distribution, we can build a confidence interval and be sure that up to a given probability the noise will lie in it.

During the encryption, we use the security oracle introduced in Definition 17 to select the noise distribution. Knowing this distribution, we can easily built a confidence interval. In real use-cases, between encryption and decryption, we perform some operations, for instance additions, multiplications, LUT evaluations and so on. Almost all operations performed on a ciphertext will increase its noise. Therefore, to check that the correctness inequality is verified, we must be able to track the impact of any operation on the noise distribution.

First, we explain what a noise formula (Definition 19) and a noise model (Definition 20) are. Informally, a noise formula is associated with an FHE operator (Definition 18) and models the evolution of the output noise variance using the input noise variance(s) and some cryptographic parameters. Finally, we refine the concept of noise bound (Definition 21) which is an upper bound for the noise variance that guarantees the correctness of the computation. We give some concrete formula to compute the noise bound in the special case of the Gaussian noise distribution (Theorem 16).

3.2.1 FHE Operator & Noise Model

First, we formalize what an FHE operator is.

Definition 18 (FHE & Plain operator) *An FHE operator $\mathcal{O}$ is an implementation of an FHE algorithm, on a given piece of hardware, taking as input some ciphertexts and potentially some plaintexts and returning one or more ciphertexts.*

A plain operator is a function mapping one or several integers into an output list of one or more integers.

Any FHE operator is associated with a plain operator and the FHE operator must compute the same operation as its associated plain operator under some (noise) constraints.

As an example, the key switch is an FHE operator taking as input one ciphertext encrypted with a key $\vec{s_1}$ and outputs a ciphertext encrypted with a different key $\vec{s_2}$. The plain Operator associated with the Keyswitch is the identity function. The PBS, the Sample Extract or the CMUX are other examples of FHE Operators.

Definition 19 (Noise Formula) *An FHE operator is associated with a noise formula which models the noise evolution between its input and output ciphertexts. A noise formula for a given homomorphic operator takes as input the variances of the input ciphertexts noise distributions, some cryptographic parameters involved in the computation, as well as the plaintext values used in the operator and outputs the variance of the output ciphertexts.*

The encryption operator has a special noise formula that we call security oracle (Definition 17). Most of the time, the noise formula of an FHE Operator depends on the LWE (or GLWE) instance i.e., on the LWE dimension n (respectively on the GLWE dimension k and the polynomial size N). It can also depend on other parameters that are specific to this operator. For example, in the key switch (Algorithm 1), the noise also depends on the decomposition parameters β and ℓ.

Definition 20 (Noise Model) *A noise model is a collection of noise formulae associated with their FHE operators (Definition 18). The noise formulae inside a noise model can be freely combined to estimate the noise of sequences of FHE operators. In particular, those formulae take the input noise variances and some parameters and estimate the output noise variances. The formulae of a noise model heavily depend on the hypothesis made during the proofs. For instance, in some cases it can be sound to assume independence between some random variables as long as the dependency is negligeable (if it exists at all). A Noise model can be a* distribution-based *model or a* bound-based *model. A* distribution-based *model contains formulae giving the exact (or a good-enough approximation) of the noise distribution whereas a* bound-based *model gives upper and lower bounds on the noise value.*

Remark 17 (Average and Worst Case) *In the literature, the* distribution-based *model is often called* average case *and the* bound-based *model* worst case*.*

Using the noise model, we can define confidence intervals for the noise i.e., intervals where noise values lie with a given probability. When we are working with a *bound-based* noise model, building a confidence interval with a probability of 1 is easy since we know explicit bounds on the noise, we can just take the interval defined by those bounds. Now, if we want to work with a *distribution-based* noise model, we need to carefully build the confidence interval and for that, we introduce the *noise bound* (Definition 21).

Remark 18 (Hamming Weight and Bound-based Model) *For efficiency, we want to have confidence intervals as tight as possible. In fact, the smaller the confidence interval* *is, the smaller some parameters will be. With a bound-based noise model, we must assume the worst case (in terms of noise) for the secret key coefficients i.e., assuming that every coefficient is a one. In order to tighten the confidence interval, we can fix the Hamming weight of the secret key. If we were to enforce half of the coefficients to be zeros and the other half to be ones, the confidence interval predicted by a bound noise model would be tighter than the ones without fixing the Hamming weight as we would know exactly the number of zeros and ones.*

3.2.2 Noise Bound

Thanks to a noise model (Definition 20), we can track the noise distributions throughout a computation. To guarantee correctness, we saw in Definition 13 that we need to enforce the noise to be smaller than $\frac{\Delta}{2}$ with Δ the scaling factor. As we only have access to the noise variances, we need to transform this inequality between the actual noise values and the scaling factor in an inequality between a noise variance and the scaling factor.

Definition 21 (Noise Bound) *Let $\mathsf{CT} \in \mathsf{GLWE}_{\vec{S}}\left(\widetilde{M}\right)$ a GLWE ciphertext of an encoding $\widetilde{M}$ of M with a message modulus p and π padding bits. Let the noise inside CT have a standard deviation σ and mean zero. The noise bound $t_\alpha(\pi, p)$ for a given failure probability α is the largest integer satisfying:*

$$\sigma \leq t_\alpha(\pi, p) \Rightarrow \mathsf{Pr}\left(\mathsf{Decode}\left(\widetilde{M}, 2^\pi \cdot p, q\right) \neq M\right) \leq \alpha$$

In Definition 21, the noise bound depends on the number of bits of padding π and on the precision p but it can also depend on other values, for instance it could depend on the degree of fullness (Definition 14) later defined in this thesis.

Remark 19 (Centered Noise Distribution) *Definition 21 assumes that the ciphertext has a centered noise distribution. It must be noticed that having a centered noise distribution helps to keep the variance as tight as possible all along the computation. For instance, during the proofs, we must estimate the variance of product of independent random variables. Let us take X, Y_1 and Y_2, three random variables, with X following an arbitrary distribution with 0 mean, Y_1 following a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ and Y_2 following a Gaussian distribution $\mathcal{N}(0, \sigma^2)$ with $\mu \in \mathbb{R} \setminus \{0\}$ and $\sigma^2 \in \mathbb{R}^+$. Let $i \in \{1, 2\}$ and let us assume that X and Y_i are independent. We have*

$$\mathsf{Var}(X \cdot Y_i) = \mathsf{Var}(X) \cdot \mathsf{Var}(Y_i) + \mathsf{Var}(X) \cdot \mathbb{E}(Y_i)^2 + \mathsf{Var}(Y_i) \cdot \cancel{\mathbb{E}(X)^2}$$

As $\mathbb{E}(Y_1)^2 > 0 = \mathbb{E}(Y_2)^2$ *, we have* $\mathsf{Var}(X \cdot Y_1) > \mathsf{Var}(X \cdot Y_2)$ *which proves our point.*

Sometimes, we face non-centered noise distribution due to the algorithms themselves. In this case, we can subtract the mean of the noise distribution to the ciphertext to center the distribution back to zero.

It must also be noticed that everything described in this thesis could be easily adapted to non-centered distributions.

In Definition 21, we gave a generic definition of the noise bound. In practice, with TFHE, the noise is almost always following a Gaussian distribution or can be approximated by a Gaussian distribution. The theorem below gives the formula to compute the noise bound for a Gaussian noise distribution.

Definition 22 (Standard Score) *Let* $A \hookleftarrow \mathcal{N}(0, \sigma^2)$ *(centered normal distribution), let* p_{fail} *be a failure probability and let* erf *be the error function* $\mathsf{erf}(z) \mapsto \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2}\mathsf{dt}$. *We define the* standard score z^* *for* p_{fail} *as* $z^*(p_{\mathsf{fail}}) = \sqrt{2} \cdot \mathsf{erf}^{-1}(1 - p_{\mathsf{fail}})$ *and we have:* $\mathsf{Pr}(A \notin] - z^*\sigma, z^*\sigma[) \leq p_{\mathsf{fail}}$.

Let $t \in \mathbb{R}$, *we have* $z^*(p_{\mathsf{fail}}) \cdot \sigma \leq t \Rightarrow \mathsf{Pr}(A \notin] - t, t[) \leq p_{\mathsf{fail}}$

Theorem 16 (Gaussian Noise & Noise Bound) *Let* CT *a GLWE ciphertext that contains a noise polynomial* $E = \sum_{i=0}^{N-1} e_i X^i \in \mathfrak{R}_q$ *such that* $\forall\ 0 \leq i \leq N-1, e_i \sim \mathcal{N}(0, \sigma^2)$. *We have an explicit formula for the noise bound* $t_\alpha(\pi, p) = \frac{\Delta}{2 \cdot \kappa}$ *with* $\kappa = z^*(p_{\mathsf{fail}})$, *the standard score (Definition 22) for* $p_{\mathsf{fail}} = 1 - \sqrt[N]{1 - \alpha}$ *i.e.,*

$$\sigma \leq t_\alpha \Rightarrow \mathsf{Pr}\left(\mathsf{Decode}\left(\widetilde{M}, 2^\pi \cdot p, q\right) \neq M\right) \leq \alpha$$

When the input ciphertext is an LWE ciphertext i.e., $N = 1$, *we have* $p_{\mathsf{fail}} = 1 - \sqrt[N]{1 - \alpha} = \alpha$.

Proof 16 (Theorem 16) *Let* E, t_α *and* p_{fail} *as defined in Theorem 16. Let us assume* $\sigma \leq t_\alpha$. *Immediately using Definition 22 and Equation 2.30, we have* $\mathsf{Pr}\left(|e_i| \geq \frac{\Delta}{2}\right) \leq$

$p_{\mathsf{fail}} = 1 - \sqrt[N]{1-\alpha}$. *Thus*

$$\begin{aligned}
\mathsf{Pr}\left(\mathsf{Decode}\left(\widetilde{M}, 2^{\pi}\cdot p, q\right) \neq M\right) &= \mathsf{Pr}\left(\bigcup_{i=0}^{N-1} |e_i| \geq \frac{\Delta}{2}\right) \\
&= 1 - \mathsf{Pr}\left(\bigcap_{i=0}^{N-1} |e_i| < \frac{\Delta}{2}\right) \\
&= 1 - \prod_{i=1}^{N} \mathsf{Pr}\left(|e_i| < \frac{\Delta}{2}\right) \quad \textit{by indep. of } \{e_i\}_{i\in[\![1,N]\!]} \\
&= 1 - \prod_{i=1}^{N}\left(1 - \mathsf{Pr}\left(|e_i| \geq \frac{\Delta}{2}\right)\right) \\
&\leq 1 - (1 - p_{\mathsf{fail}})^{\frac{1}{N}} = \alpha
\end{aligned}$$

□

Remark 20 (Arbitrary Noise Distribution & Noise Bound) *With TFHE, most of the time, the output distribution of an FHE Operator is a Gaussian distribution (or can be approximated by it). If we were to manipulate arbitrary distribution, we could still build confidence intervals but they can not be always as tight as the one we built assuming Gaussian distribution. For instance, we can use Chebyshev's inequality: for a random variable E of variance σ^2 and 0 mean, for $\kappa \in \mathbb{R}^*$ we have $\mathsf{Pr}\left(|E| \geq \kappa\cdot\sigma\right) \leq \frac{1}{\kappa^2}$.*

To conclude, using the noise bound (Definition 21), we can guarantee a correct decoding up to a given probability using only the distribution of the noise which can be publicly estimated. The tightness of the noise model (Definition 20) is crucial to build tight confidence intervals.

3.3 FFT-related Noise

To guarantee the correctness of a computation, we rely on a tight noise model (Definition 20) and on the noise bound (Definition 21). For this to work, we need the noise formulae to reflect exactly the behavior of the noise throughout the computation. In theory, we could settle for looser formulae, as long as the bounds (in the case of a *bound* noise model) or the variances of the distributions (in the case of a *distribution* noise model) are conservative. We cannot settle for optimistic formulae i.e., formulae that give a lower bound on the noise as this could impact the correctness.

One major example of mismatch between theoretical formulae and experimental observations is the case of the external product (Algorithm 7) which is one of the main building blocks of the PBS (Algorithm 10). During an external product, we need to perform polynomial multiplications and additions in $\mathfrak{R}_q$. If we were using a schoolbook multiplication method, the cost of these operations would be in $\mathcal{O}\left(N^2\right)$ with N the polynomial size. We could also use the Karatsuba algorithm which has a cost in $\mathcal{O}\left(N^{\log_2(3)}\right)$. Instead, the external product is implemented using a Fast Fourier Transform (FFT) which allows to compute a polynomial multiplication in $\mathcal{O}\left(N \ \log_2 N\right)$. The use of the FFT in the PBS is one of the reasons that makes it the fastest known bootstrapping algorithm. However, the speed of the FFT comes with some drawbacks: the noise formula of the external product does not match real-life experiments because of the noise introduced by the FFT. We identified three issues with the way the FFT is used.

In this section, we explain the issues with the FFT in the context of the TFHE bootstrapping and explain how to build an experimental corrective formula to correctly model the noise growth inside a PBS.

3.3.1 Issues with the FFT

First, by definition, floating-point arithmetic is approximate. During the FFT computation, we accumulate some errors due to the fact that we are limited by machine precision to represent the twiddle factors[4] and intermediate values throughout the Fourier transform. It must be noted that the order of the operations performed during the FFT influences the error because we loose associativity and distributivity between additions and multiplications when composing floating-point operations.

Then, when casting the coefficients of a GLWE ciphertext with $q = 2^{64}$ (represented by 64-bits unsigned integers) to floating-point numbers float64 (or double-precision floating-point), a modulus switching of sorts happens. As a matter of fact, a float64 is composed of 53 bits for the mantissa, 1 bit for the sign and 11 bits for the exponent [20]. When casting an integer on 64 bits as a floating point number, under the hood, a rounding happens and discards the 11 least significant bits of the input integer. This loss of information can be represented by an error added to the ciphertext. This error is very similar to the noise added by a modulus switch.

Finally, once we have operands in the Fourier domain, we perform the modular multi-

4. A twiddle factor is a constant coefficient that is multiplied to the data during an FFT.

plications and several modular additions. All these operations happen in the FFT domain, so we cannot perform the reduction modulo q needed to discard the useless MSB and keep the LSB. In fact, the opposite happens, due to the floating-point arithmetic, the MSB will be kept and we will loose the LSB. When we leave the FFT domain, we will discard those MSB but the LSB are lost. This loss of information can be seen as a new, deterministic error added to the ciphertext.

A thorough analysis of the FFT and of the computations happening in the Fourier domain could give us a corrective formula to patch the external product formula and make it match the experiments. In our work, we decided to use experiments to come up with a corrective formula which would guide us in our theoretical analysis. The theoretical analysis is still an ongoing subject and will not be discussed here.

3.3.2 Experimental Noise Formula

The first step to build this corrective formula is to estimate the noise after an external product on a wide range of parameters. For instance, we can choose the GLWE dimension k in $[\![1, 6]\!]$, the polynomial size in $\{2^8, 2^9, \cdots, 2^{17}\}$, the maximum level of the decomposition ℓ in $[\![1, 64]\!]$ and the logarithm of the decomposition base in $[\![1, 64]\!]$. We can define the Cartesian product of each of those sets and called it $\mathcal{P}$. At this stage, we can choose to work on the full set $\mathcal{P}$ or on a random subset of it. We set the encryption variance for the input GLWE ciphertext and for the GGSW ciphertext using the noise oracle introduced in Definition 17.

For every set of parameters in $\mathcal{P}$, we compute a given number of external products and compute the experimental variance. For some parameters, we do not need to compute the experiments. If the theoretical variance is greater or equal to the variance of an uniform distribution, we can remove this set of parameters from $\mathcal{P}$ as this set of parameters will never be used. The parameters, number of samples, theoretical and experimental variances are stored in a text file to be exploited later.

Once we have everything, we apply almost the same process as during the building of the noise oracles (Section 3.1). Basically, we do some data visualizations to have an intuition on the formula which gives us the following formula where ω is a value that must be found:

$$\mathsf{FftError}\left(k, N, \beta, \ell\right) = 2^{\omega} \cdot \ell \cdot \beta^2 \cdot N^2 \cdot (k+1)$$

Then, we apply a least square fitting to find the best ω. We find that the best value is $\omega \approx 19.4$. Intuitively, the noise added by the FFT should be scaled in the same way as the noise introduced by the casting from 64-bit unsigned integers (with $q = 2^{64}$) to 64-bit floating-point numbers. As the number of bits for the mantissa in the floating point representation is 53, we expect the variance of the noise to be scaled around $2 \cdot (64 - 53) = 22$. With $\omega = 19.4 = 22 - 2.6$, we conclude that the intuition is close to the reality.

Remark 21 (Implementation) *We used the Concrete Framework[5] written in Rust to perform the noise acquisitions and we used Python to do the fitting and find the best ω.*

Notice that the formula is only valid for this FFT implementation.

We can add some margins by rounding up ω. Using the corrective formula, we can verify with experiments that the predicted noise is now at least as big as the experimental noise. Now that we have a noise model that matches the experiments, we can safely use our noise model to find correct parameters. This will be the whole topic of Chapter 4.

5. `https://github.com/zama-ai/concrete`

Chapter 4

Optimization for FHE

We introduced in Chapter 3 the definition of an FHE operator which is an implementation of an FHE algorithm on a given piece of hardware (Definition 18). We also gave the definition of a noise formula, a formula associated with an FHE operator that takes as input the cryptographic parameters and the input noise distribution and predicts the noise distributions of the output ciphertexts (Definition 19). A collection of noise formulae is called a noise model (Definition 20) and is used to track the noise throughout an FHE computation. The variance of the noise in a ciphertext must not exceed the noise bound (Definition 21 and Theorem 16), a threshold that guarantees the correctness of a computation up to a given failure probability.

Most of the time, there is a trade-off between the time needed to perform an FHE operation and the amount of noise in a ciphertext: the larger the parameters are, the slower the execution is and the smaller the noise variance is. Finding the sweet spot where we have an execution as fast as possible while still guaranteeing a noise variance below the noise bound is tricky. In this chapter, we introduce the blueprint for an optimizer framework that automatizes the search for cryptographic parameters according to this trade-off.

First, we formalize the optimization problem as a minimization under constraints problem, introduce some key concepts and the guarantees we want to have during the parameter selection. Then, we explain how to simplify this problem in order to efficiently solve it. To do so, we introduce the concept of atomic pattern types (Definition 28). Using all this, we explain how to truly compare FHE operators that compute the same plain operator (Definition 18). This comparison method will be used in the next chapters to prove the interest of the new algorithms introduced in this thesis. Finally, we give additional applications using our optimizer framework. For instance, we introduce an efficient way to deal with several evaluation keys (bootstrapping keys, key switching keys) and to insert PBS evaluation inside sequences of leveled operations.

4.1 Optimization Problem

To compute over ciphertexts, one needs to select cryptographic parameters. If the computation involves LWE ciphertexts, we need to set the LWE dimension n and when we use GLWE ciphertexts we need to set their polynomial size N and their GLWE dimension k. In this thesis, those parameters are called *macro-parameters*. In addition, some FHE operators (Definition 18) come with degrees of freedom that one also needs to set. As an example, to perform a key switch and to generate the key switching key, we need to choose a base $\mathfrak{B}$ and a level ℓ. Those parameters are called *micro-parameters*, because they are only used locally, inside an FHE operator.

Micro and macro parameters have an impact on the cost and on the noise added during the evaluation of an FHE operator, so they need to be carefully picked. In practice, one needs to find parameters not only for one FHE operator (see definition later on), but for a graph of FHE operators. We focus on a special case of graph, a *directed acyclic graph* (DAG). A vector containing values for both the micro and macro parameters needed for a given DAG is called a *parameter set*. The more degrees of freedom in a DAG there are, the larger the parameter set is and the harder the parameter search is. The question we answer in this chapter is:

For a given graph of FHE operators, how to find parameters such that the evaluation is the fastest while preserving both correctness and security?

To answer this question, first, we introduce the concept of a cost model (Definition 23), we use it to estimate the total cost of a graph of FHE operators (Definition 24). The cost model serves as a surrogate of the execution time using algorithmic complexities. Then, we detail the guarantees of our optimization framework: security, correctness and efficiency. Finally, we introduce some key concepts to build our framework, for instance the noise feasible set (Definition 26) and formalize the optimization problem as a minimization problem under constraints (Equation (4.1)).

4.1.1 Cost Model

One of our main goals is to find parameters that would lead to an efficient FHE computation. To do so, one needs to compare different parameter sets in terms of execution time and select the best one. This requires a way to predict the execution time according to a given parameter set. A trivial way would be to benchmark the execution of the circuit with every possible set of parameters in order to rank them. While this method would give us an exact metric to compare parameter sets, it would be very costly as there are lots of parameters to consider, especially if the circuit is large. Another downside is that the estimated cost would heavily depend on the machine on which we made the experiments. If we were working with another machine, we would need to run again all the benchmarks. Even on the same machine, benchmarks are precise up to a certain point, due to the inherent variability of any computation and the values we would obtain would be good estimates but not perfect ones.

Luckily, we have a more efficient method to rank parameter sets according to their cost. Instead of using the execution time as a metric, we can build a surrogate of the execution time that will give us the possibility to rapidly rank parameter sets on any machine. In this context, a surrogate is an approximation of the real metric we want to study but is simpler to evaluate and good enough to yield satisfying results. We call this surrogate a *Cost Model*.

Definition 23 (Cost Model) *An FHE operator (Definition 18) is associated with one or several cost formulae. A cost formula is a surrogate for the metric one wants to minimize, it could be an approximation of the execution time, the power consumption, or the price (of running the FHE operator in an on-demand cloud computing platform). The cost model is the collection of cost formulae for a given metric. A cost function is noted* $\mathsf{Cost}(\cdot)$*.*

For all the experiments and benchmarks in this thesis, we will consider a *cost model* that approximates the execution time on a single thread machine using the algorithmic complexities of each FHE operator (Definition 18). As some FHE operators have a negligible cost compared to other operators, sometimes we assume they are free. It is the case of the sample extract (Theorem 9) in a PBS (Theorem 14).

It means that we count the number of additions, multiplications, cast operations between integer types, and the asymptotic cost of the FFT in each algorithm and use those as a surrogate of the execution time. For instance, the operation $\sum_{i=1}^{\alpha} M_i \cdot \mathsf{CT}_i$, where $M_i \in \mathfrak{R}_q$ are polynomials and $\mathsf{CT}_i = (A_i, B_i) \in \mathfrak{R}_q^2$ are RLWE ciphertexts, will have a cost of:

$$\underbrace{3\ \alpha\ N\log(N)}_{\text{to FFT domain}} + \underbrace{2\ \alpha\ N}_{\text{complex }\times} + \underbrace{(\alpha-1)\ N}_{\text{complex }+} + \underbrace{2\ N\log(N)}_{\text{to standard domain}}$$

In the first term, we have 3α forward FFT (α polynomials and α RLWE ciphertexts composed of two polynomials) and the asymptotic cost of the FFT is $N\log_2(N)$ which gives a cost of $3\alpha N\log_2(N)$. Once all polynomials are in the Fourier domain, we need to perform $2\alpha N$ multiplications and $(\alpha-1)N$ additions. Finally, we perform a backward FFT on the RLWE ciphertext. With this cost model, we assume the cost of a multiplication between complex numbers and integers to be the same than the cost of an addition between integers and between complex numbers. While this hypothesis is false, in practice, it is close enough to provide efficient parameter sets.

4.1.2 Guarantees

As explained in the introduction of this chapter, we want to create a tool that automatically finds the best trade-off between the cost and the noise of an FHE DAG to overcome Limitations 14 and 15.

Our framework takes as input a graph of FHE operators, a level of security and a correctness probability, and outputs a parameter set that will guarantee:

1. the desired *level of security*,
2. the desired *correctness* probability,
3. the smallest *cost* possible.

The first guarantee is easy to reach using the security oracle introduced in Definition 17. We can build it using the lattice estimator [APS15] as explained in Section 3.1. To reach the required security level for a given LWE dimension (or GLWE dimension and polynomial size), a given key distribution and a given ciphertext modulus, one can always increase the amount of noise at encryption time (or evaluation and public key generation). Using this, one does not need to find the best encryption noise, one can simply look for the best LWE dimension (or GLWE dimension and polynomial size) and take the minimal encryption noise given by the security oracle. In the end, one is sure to provide enough security as the noise is chosen with respect to other ciphertext parameters.

To guarantee the correctness of a computation (guarantee 2), one needs to rely on the noise model (Definition 19) of each FHE operator in the graph. We explained in Definition 13 and in Section 3.2 that the noise inside a ciphertext must remain below some threshold in order to ensure correct decoding. If the noise grows too much, the message will be tampered by the noise and the decryption algorithm will not yield the correct result. In order to guarantee the correctness, one needs to track the noise at each step of the computation using the noise model (Definition 20) and choose parameters in a way that the noise remains small enough.

The last guarantee is to have a cost as small as possible. For that, one needs to use the cost model introduced in Definition 23 and select the parameters that minimize this cost among the ones that satisfy guarantees 1 and 2. Naturally, the more realistic the cost model is, the better the parameters will be in practice.

4.1.3 Foundations of the Optimization Framework

We start by explaining the core ideas to ensure the three aforementioned guarantees.

As said above, one needs to choose the *macro-parameters* among a set of possible values. For example, the polynomial size N is often chosen as a power of 2. One wants to narrow it down to a finite set, and a practical yet wide enough space for TFHE could be $\mathcal{P}_N = \{2^8, 2^9, \cdots, 2^{17}\}$ and we call it the *search space* of N. In the same manner, the LWE dimension n could be selected in $\mathcal{P}_n = [\![256, 2048]\!]$ and the GLWE dimension in $\mathcal{P}_k = [\![1, 6]\!]$. To reduce the size of $\mathcal{P}_n$, we can consider a subset of it, for instance, we could use $\mathcal{P}_n = \{i \in [\![256, 2048]\!] \mid i \mod 4 = 0\}$.

In the definition Definition 24, we introduce the concept of FHE DAG. A *directed acyclic graph* (DAG) is a directed graph with no cycles.

Definition 24 (FHE DAG) *Let $\mathcal{G} = (V, L)$ be a DAG of FHE operators (Definition 18). We define $V = \{\mathcal{O}_i\}_{1 \le i \le \alpha}$ as the set of vertices, each of them being an FHE operator. We define L as the set of edges, each of them associated with the message modulus $p \in \mathbb{N}$ of the encrypted message i.e. $L \subset \{\{x, y, p\} \,|(x, y) \in V^2, p \in \mathbb{N}\}$. When there is no possible confusion regarding L, we will simply write $\mathcal{G} = V$. We note $\mathsf{Cost}(\mathcal{G}, x)$ the cost of running the FHE graph $\mathcal{G}$ with the parameter set x.*

For a given FHE DAG $\mathcal{G}$, one also needs to set the *micro parameters.* For example, the decomposition base $\mathfrak{B}$ for a KS or a PBS can be selected in $\mathcal{P}_{\mathfrak{B}} = \{\mathfrak{B}_i \mid \mathfrak{B}_i = 2^i, \forall i \in [\![1, \lfloor \log_2(q) \rfloor]\!]\}$ and the level of the decomposition ℓ in $\mathcal{P}_\ell = [\![1, \lfloor \log_2(q) \rfloor]\!]$. As $(\mathfrak{B}, \ell)$ are used to do a radix decomposition of each integer composing the input ciphertext as explained in Section 2.3.2, we have in practice that $\ell \cdot \log_2(\mathfrak{B}) \le \log_2(q)$ so we will consider $(\mathfrak{B}, \ell)$ as one unique variable in $\mathcal{P}_{\mathfrak{B},\ell} = \{(\mathfrak{B}, \ell) \mid \ell \cdot \log_2(\mathfrak{B}) \le \log_2(q), \forall (\log_2(\mathfrak{B}), \ell) \in [\![1, \lfloor \log_2(q) \rfloor]\!]^2\}$.

In the end, one needs to choose a set of parameters in the Cartesian product of the search spaces of all the micro and macro parameters of a graph $\mathcal{G}$. This space is noted $\mathcal{P}_\mathcal{G}$ and is called the *search space* of $\mathcal{G}$.

Definition 25 (Search Space) *Let $\mathcal{G}$ be an FHE DAG with its associated micro and macro parameters $\{x_i\}_{i\in I}$ and a set of indexes I. Each parameter x_i must be picked in a dedicated search space $\mathcal{P}_i$.*

The search space of $\mathcal{G}$ is defined as the cartesian product of these search spaces i.e.,

$$\mathcal{P}_\mathcal{G} = \prod_{i\in I} \mathcal{P}_i.$$

In the rest of the paper, this set is simply called $\mathcal{P}$ when there is no ambiguity on the graph.

Every ciphertext involved during the evaluation of an FHE DAG must have a noise variance smaller than its associated noise bound (Definition 21) in order to guarantee the correctness of the computation. With these constraints, we define the *noise feasible set*, a subset of the search space $\mathcal{P}$ where every set of parameters will guarantee a correct computation.

Definition 26 (Noise Feasible Set) *Let $\mathcal{G}$ be an FHE DAG such that $\mathcal{G} = (V, L)$ with $L = \{(\cdot,\cdot,p_i)\}_{i\in[\![1,|L|]\!]}$, and let $\alpha \in [0,1]$ be a failure probability. Let $\{\sigma_i : \mathcal{P} \mapsto \mathbb{R}\}_{1\leq i\leq |L|}$ be the standard deviation of the noise in the ciphertexts transiting on every edge of $\mathcal{G}$. Let $\{t_\alpha(p_i)\}_{1\leq i\leq |L|}$ be the noise bounds associated to each precision. For every edge i, we must have $\sigma_i(x) \leq t_\alpha(p_i)$. This defines a subset of the search space $\mathcal{P}$: $\mathcal{S}_i = \{x \in \mathcal{P} | \sigma_i(x) \leq t_\alpha(p_i)\}$. The intersection of all those sets is the noise feasible set $\mathcal{S}$: the set of parameter sets that will lead to a correct computation. We have:*

$$\mathcal{S} = \bigcap_{i\in I} \mathcal{S}_i = \{x \in \mathcal{P} | \forall i \in [\![1, |L|]\!], \sigma_i(x) \leq t_\alpha(p_i)\}$$

By choosing a set of parameters that is in the noise feasible set, we are sure to satisfy the correctness (Guarantee 2). In this noise feasible set, we want to find the set of parameters minimizing the cost of the FHE DAG. Formally, we want:

$$\arg\min_{x\in\mathcal{P}} \mathsf{Cost}(\mathcal{G}, x) \text{ s.t. } x \in \mathcal{S}(\mathcal{G}) \tag{4.1}$$

The problem of finding efficient and correct FHE parameters is then a minimization problem under constraints. We can naturally use optimization techniques to solve it. The issue is that the complexity of the problem is dependent on the size of the FHE DAG which can rapidly become unrealistic for large DAGs. In the next section, we introduce several non trivial simplifications before even starting the optimization.

Remark 22 (Other Feasible Sets) *As we defined a feasible set for the noise, we can also define other feasible sets for other constraints. For instance, we could define a feasible set to limit the size of the evaluation keys (key switching keys, bootstrapping keys, ...), to limit the size of the required bandwidth (size of the ciphertexts) or even to add some constraints between parameters.*

Remark 23 (MINLP Optimization) *It is far from being simple to solve the optimization problem described in Equation* (4.1)*. In practice, the noise variances are represented by real values because their domain of definition is too big to be represented with integers (when $q \geq 2^{64}$) and the cost is represented by an integer. The cost formulae we use are quite simple but the noise formulae are not and are highly non-linear. This means that the optimization problem is a* mixed-integer non-linear problem *and it cannot be solved with an analytical approach as it is NP-hard [KM78].*

Remark 24 (Multi-Objective Optimization) *We described the optimization problem where we want to minimize one metric, the cost, under security and correctness constraints. We could want to add another metric, for instance the size of the public material and do a* multi-objective optimization *to extract interesting trade-off between the cost and the size of the public material. The user can then chose the trade-off that best suits their needs.*

4.2 Solving the FHE-to-TFHE Translation Problem

In this section, we explain how to solve the optimization problem introduced in Equation (4.1). We introduce a way to re-write the graph with higher level building blocks (see Definition 28) that we can compare before beginning the optimization (Definition 29). This helps decreasing the complexity of the resolution. Then, we study the feasible set of a simple graph composed of a dot product (Theorem 5), a key switch (Theorem 6) and a PBS (Theorem 14) and explain in detail how to further simplify the optimization by removing decomposition parameters $(\mathfrak{B}, \ell)$ that are always part of sub-optimal solutions. Next, we introduce the full-fledge optimization problem, taking as input a plain DAG (Definition 31), doing the translation to the FHE world and outputting a fully parametrized FHE DAG. Finally, we explain how to extend the concept of failure probability to a whole graph, in order to guarantee the correctness of a computation not at an FHE operator level but at the level of the whole graph.

4.2.1 Graph Transformations

To simplify the optimization problem, we present an analysis that can be applied on any FHE DAG. The idea is to subdivide it into subgraphs with the constraint that to compute the noise distribution of a ciphertext in one of these subgraphs, we do not need to know the noise distribution of a ciphertext in another subgraph. The starting point is to note that there are some FHE operators that output ciphertexts with a noise independent of the input noise for some well-chosen parameters and with some success probability. This motivates us to distinguish those FHE operators from the rest:

Definition 27 (FHE Operator Categories) *We divide the FHE operators (Definition 18) into two categories regarding their respective noise formulae:*

1. *an operator which outputs a noise independent of the input noise with a given probability, such as the PBS in our context;*
2. *an operator which adds some noise to the input noise, such as a KS or a dot product;*

Using this distinction, for any FHE DAG, we can identify subgraphs that are independent from others. Now that we have several independent subgraphs, we want to find a way to compare them together. To do so, we define the notion of *atomic pattern types* to regroup subgraphs of FHE operators called *atomic patterns* that we know how to compare. For instance, two atomic patterns of the same type can represent the same subgraph but with different message moduli p or different numbers of inputs.

Definition 28 (Atomic Pattern Type) *An Atomic Pattern (AP) type $\mathcal{A}^{(\cdot)}$ corresponds to a subgraph of FHE operators that outputs one or several ciphertexts with a noise independent of the input noise.*

An Atomic Pattern A is a particular instance of an AP type $\mathcal{A}^{(\cdot)}$. When an AP $A \in \mathcal{A}^{(\cdot)}$ is instantiated with a parameter set x, we write $A(x)$. From $A(x)$, one can estimate its *total cost using a cost model and one can also estimate the amount of noise at any edge of its FHE subgraph. From these noise distributions, we can estimate the probability of correctly computing the atomic pattern with the parameter set x.*

Once we have identified the atomic pattern types in a graph $\mathcal{G} = (V, E)$, we can build an FHE DAG $\mathcal{G}' = (V', E')$ such that each FHE operator in V' is an atomic pattern i.e., $V' = \{A_i(\cdot)\}_{i \in [\![1, |V'|]\!]}$. This new graph is equivalent to the input graph and now we can express the feasible set of $\mathcal{G}'$ with the feasible sets of each atomic patterns. Formally, we have $\mathcal{S}(\mathcal{G}) = \bigcap_{i \in [\![1,|V'|]\!]} \mathcal{S}(A_i(\cdot))$. We leverage the fact that we can compare the noise between atomic patterns of the same type to efficiently find the atomic patterns that have the smallest feasible sets. We will describe this procedure for a noise feasible set, but this can be extended to another kind of feasible set, for instance, the evaluation key sizes.

As we can compare two AP of the same type even without a given set of parameters, we can introduce the notion of *domination between AP*.

Definition 29 (AP Domination) *Let $\mathcal{G}$ be an FHE DAG. Let A and A' be two atomic patterns of $\mathcal{G}$. The AP A dominates the AP A' if any $x \in \mathcal{P}(\mathcal{G})$ satisfying the noise constraints of A also satisfies the constraints of A'. More formally, we have $\mathcal{S}(A) \subset \mathcal{S}(A')$ i.e., $\mathcal{S}(A) \bigcap \mathcal{S}(A') = \mathcal{S}(A)$. A' is said to be dominated by A.*

For all AP types in a graph $\mathcal{G}$, for all APs of the same type, we can simply keep the ones that are not dominated by any other AP. Indeed, we can discard the APs that are dominated because their constraints will be satisfied if the constraints of one of their dominant AP is satisfied.

With TFHE, we mainly use three FHE operators: the homomorphic dot product (DP), the key switch and the programmable bootstrapping. The key switch is generally computed before the PBS (as in [CJP21]). Naturally, we define our first concrete atomic pattern type $\mathcal{A}^{(\mathrm{CJP21})}$ with these three FHE operators.

Definition 30 ($\mathcal{A}^{(\mathbf{CJP21})}$ Atomic Pattern Type) *We define a first atomic pattern type $\mathcal{A}^{(CJP21)}$ as a subgraph composed of a dot product (DP, Theorem 5), followed by a key switch (Algorithm 1 and Theorem 6) and a final PBS (Algorithm 10 and Theorem 14). This subgraph is illustrated in Figure 4.1.*

In the dot product, we assume every input to be the output of a bootstrapping. In Theorem 5, we saw that the 2-norm ν and the input variance are sufficient to compute the output noise of a dot product if every input ciphertext has the same Gaussian noise

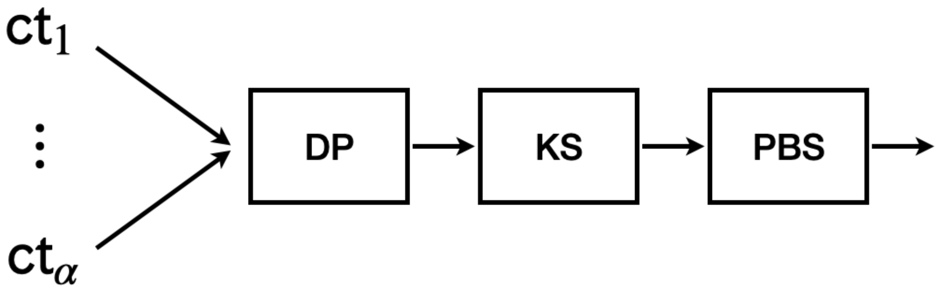


Figure 4.1: $\mathcal{A}^{(\mathrm{CJP21})}$ atomic pattern type, composed of a dot product (DP), a key switch (KS) and a programmable bootstrap (PBS)

distribution. Hence, an atomic pattern A_1 of type $\mathcal{A}^{(CJP21)}$ is entirely characterized by two values: the 2-norm ν and its noise bound t. We will denote it by $A_1 = A(\nu, t)$. When the atomic pattern A_1 is instantiated with a parameter set x, we write $A_1(x) = A(\nu, t)(x)$.

Remark 25 (Freshly Encrypted Input in DP) *In Definition 30, we do not consider the fact that some of those inputs could be freshly-encrypted ciphertexts and not output of a bootstrap. Everything we describe below can be easily modified to take that into account.*

Remark 26 (Cost of $\mathcal{A}^{(\mathrm{CJP21})}$) *To simplify the problem, we assume the cost of the dot product to be negligible compared to the other FHE operators. Here, we assume the cost of an atomic pattern A to be the sum of the cost of every FHE operator inside it, i.e., the cost of a PBS and the cost of a KS.*

4.2.2 Pre-Optimization

It is easy to compare the noise in atomic patterns of type $\mathcal{A}^{(\mathrm{CJP21})}$ using the following property which is a special case of Definition 29.

Theorem 17 (AP Domination) *Let's consider A_1, A_2 two APs of a type $\mathcal{A}^{(CJP21)}$ that includes a homomorphic DP, ν_1, ν_2 two 2-norms such that $\nu_1 \leq \nu_2$ and t_1, t_2 two noise bounds where $t_2 \leq t_1$. We have $A_1 = A(\nu_1, t_1)$ and $A_2 = A(\nu_2, t_2)$.*

Then, we have:

$$\mathcal{S}(A(\nu_2, t_2)) \subset \mathcal{S}(A(\nu_1, t_1))$$

i.e.,

$$\mathcal{S}(A(\nu_2, t_2)) \bigcap \mathcal{S}(A(\nu_1, t_1)) = \mathcal{S}(A(\nu_2, t_2))$$

A_1 is said to be dominated by A_2.

Proof 17 (Theorem 17) *A_1 and A_2 share the same type. Let $x^* \in \mathcal{S}\left(A(\nu_2, t_2)\right)$, we have*

$$\nu_2^2\sigma_{BR}^2(x^*) + \sigma_{KS}^2(x^*) + \sigma_{MS}^2(x^*) \leq t_2^2$$

with σ_{BR}, σ_{KS} and σ_{MS} the standard deviations after the blind rotate, the key switch and the modulus switch. Immediately, we have

$$\nu_1^2\sigma_{BR}^2(x^*) + \sigma_{KS}^2(x^*) + \sigma_{MS}^2(x^*) \leq \nu_2^2\sigma_{BR}^2(x^*) + \sigma_{KS}^2(x^*) + \sigma_{MS}^2(x^*) \leq t_2^2 \leq t_1^2$$

So, $x^* \in \mathcal{S}\left(A(\nu_1, t_1)\right)$.

□

Given a graph $\mathcal{G} = \{A_i\}_{i\in I}$ of atomic patterns of type $\mathcal{A}^{(\text{CJP21})}$, we can apply Theorem 17 to simplify the construction of $\mathcal{S}(\mathcal{G})$. In fact, we do not need to build each $\mathcal{S}(A_i)$ as some of them are included in others. From our input graph $\mathcal{G}$, we construct a new graph $\mathcal{G}_{\mathsf{pareto}} = \mathsf{Pareto}(\mathcal{G}) = \{A'_i\}_{i\in I_{\mathsf{pareto}}}$ containing only non-dominated atomic patterns using the same theorem (Pareto comes from *Pareto front*, a well known concept in optimization). It follows that $\mathcal{G}_{\mathsf{pareto}}$ contains at most as many atomic patterns as there are different noise bounds in the graph.

An interesting property of $\mathcal{G}_{\mathsf{pareto}}$ is that $\mathcal{S}(\mathcal{G}) = \mathcal{S}(\mathcal{G}_{\mathsf{pareto}})$ i.e., if one solves the optimization problem (Equation (4.1)) using $\mathcal{S}(\mathcal{G}_{\mathsf{pareto}})$ instead of $\mathcal{S}(\mathcal{G})$, we will get the same optimal solution. This is interesting because to compute $\mathcal{S}(\mathcal{G}) = \bigcap_{i\in I}\mathcal{S}(A_i)$ we need to build $|I|$ search spaces and with $\mathcal{G}_{\mathsf{pareto}}$, we only need to build $|I_{\mathsf{pareto}}|$ search spaces and most of the time $|I| \gg |I_{\mathsf{pareto}}|$.

Another useful observation is that in an atomic pattern of type $\mathcal{A}^{(\text{CJP21})}$ (and in most of the atomic pattern types defined in this manuscript), the noise is strictly increasing until the end of the modulus switching step in the final PBS. As the noise bound is assumed to be constant inside one atomic pattern, we do not need to check that the noise satisfies the noise bound t after the dot product or after the key switching, we only need to do it after the modulus switch. If we note $\sigma_{\mathsf{MS},1}$, the standard deviation of the noise after the modulus switching in an atomic pattern A_1, we have $\mathcal{S}(A_1) = \{x \in \mathcal{P} | \sigma_{\mathsf{MS},1}(x) \leq t\}$.

As we assume the cost of a dot product to be negligible, the cost of an atomic pattern of type $\mathcal{A}^{(\text{CJP21})}$ can be computed from the cost of the key switch (Theorem 6) and of the bootstrap (Theorem 14). As those costs only depend on the cryptographic set of parameters and not on (ν, t), it means that the cost of an atomic pattern of type $\mathcal{A}^{(\text{CJP21})}$ is the same for any (ν, t).

For a graph $\mathcal{G} = \{A_i\}_{i\in I}$, we have $\mathsf{Cost}\left(\mathcal{G}, x\right) = \sum_{i\in I} \mathsf{Cost}\left(A_i, x\right)$ for x a solution in the search space $\mathcal{P}$ and we know that for any $(i,j) \in I^2, \mathsf{Cost}\left(A_i, x\right) = \mathsf{Cost}\left(A_j, x\right)$, so instead of minimizing the cost of running the total graph $\mathcal{G}$, we can settle for minimizing the cost of one atomic pattern of type $\mathcal{A}^{(\text{CJP21})}$.

To sum up, for a given graph $\mathcal{G}$, instead of solving equation 4.1, we can build a new graph $\mathcal{G}_{\mathsf{pareto}}$ as described above and solve the following problem which will give us the same value but will be easier to compute.

$$\arg\min_{x\in\mathcal{P}} \mathsf{Cost}\left(\cdot, x\right) \text{ s.t. } x \in \mathcal{S}\left(\mathcal{G}_{\mathsf{pareto}}\right) \tag{4.2}$$

The above problem is greatly simplified but still depends on the input graph $\mathcal{G} = \{A\left(\nu_i, t_i\right)\}_{i\in I}$. When we first tried to apply our optimization framework on real-life use cases, we implemented a prototype in Python. The resolution was taking too much time (around half an hour), so we looked for ways to pre-compute parameter sets that work for a wide range of applications. Given a graph $\mathcal{G}$, we will be able to immediately select the best set of parameters in those pre-computed sets. In the following, we explain how to efficiently pre-compute good parameter sets.

A simple way to do that is to introduce another special graph $\mathcal{G}_{\mathsf{worst}}$, that we call the *worst-case* atomic pattern. It is defined as $\mathcal{G}_{\mathsf{worst}} = \{A\left(\max_{i\in I}\nu_i, \min_{i\in I} t_i\right)\}$. This graph is reduced to only one atomic pattern that may or may not be present in the input graph $\mathcal{G}$. Using Theorem 17, we know that $\mathcal{S}\left(\mathcal{G}_{\mathsf{worst}}\right) \subset \mathcal{S}\left(\mathcal{G}\right)$. So if we solve equation 4.2 on $\mathcal{G}_{\mathsf{worst}}$, we end up with a feasible solution for $\mathcal{G}$. Using this new graph, we are able to pre-compute sets of cryptographic parameters for different values of (ν, t). Given a graph $\mathcal{G}$, we will select the set of parameters for the worst case atomic pattern $\mathcal{G}_{\mathsf{worst}}$ of $\mathcal{G}$.

Above, we found a feasible solution and intuitively, this solution is close to the optimal one. To have bounds on the optimality of the solution for a graph $\mathcal{G} = \{A_i\}_{i\in I}$, we can use another particular graph $\mathcal{G}_{\mathsf{best}}$ defined as $\mathcal{G}_{\mathsf{best}} = \{A(\nu^*, t^*)\}$ with $t^* = \min_{i\in I} t_i$ and $\nu^* = \max\left\{\nu_i | A\left(\nu_i, t^*\right) \subset \mathcal{G}\right\}$ i.e., a graph composed of the atomic pattern of the graph $\mathcal{G}$ that has the smallest noise bound and the highest 2-norm for this noise bound. If the worst-case atomic pattern is the same as the best-case atomic pattern, the method described above yields an optimal solution as $\mathcal{S}\left(\mathcal{G}_{\mathsf{pareto}}\right) = \mathcal{S}\left(\mathcal{G}_{\mathsf{worst}}\right) = \mathcal{S}\left(\mathcal{G}_{\mathsf{best}}\right)$. If they are different, we can deduce a bound of optimality: as $\mathcal{G}_{\mathsf{best}} \subset \mathcal{G}$, we know that $\mathcal{S}\left(\mathcal{G}\right) \subset \mathcal{S}\left(\mathcal{G}_{\mathsf{best}}\right)$. Solving equation 4.2 for $\mathcal{G}_{\mathsf{best}}$ gives us a lower bound on the cost of the optimal solution of equation 4.2 for $\mathcal{G}$ and solving equation 4.2 for $\mathcal{G}_{\mathsf{worst}}$ gives us an upper bound.

4.2.3 CJP Atomic Pattern: Further Simplifications

Let us look more closely at the feasible set of type $\mathcal{A}^{(\mathrm{CJP21})}$.

Study of the Feasible Set

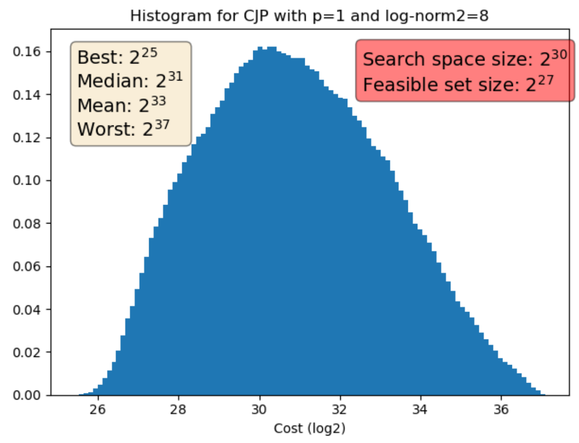
For this experiment, we use the following ranges for each micro and macro parameters. The polynomial size N is taken in $\mathcal{P}_N = \{2^8, 2^9, \cdots, 2^{17}\}$. The LWE dimension n is selected in $\mathcal{P}_n = [\![512, 1024]\!]$ and the GLWE dimension in $\mathcal{P}_k = [\![1, 6]\!]$. The logarithm of the decomposition base $\log_2(\mathfrak{B})$ and the maximum level of decomposition ℓ are defined in $[\![1, 64]\!]$.

For an atomic pattern A of type $\mathcal{A}^{(\mathrm{CJP21})}$, the search space $\mathcal{P}(\mathcal{G})$ has an approximate size of 2^{39}. Now, let us consider the decomposition parameters as a couple as described above i.e., with $(\log_2(\mathfrak{B}), \ell) \in \{(\log_2(\mathfrak{B}), \ell) \in [\![1, 64]\!]^2 | \log_2(\mathfrak{B}) \cdot \ell \leq \log_2(q)\}$ with $q = 2^{64}$, the ciphertext modulus. With this additional constraint, the search space is smaller, with an approximate size of 2^{30}.

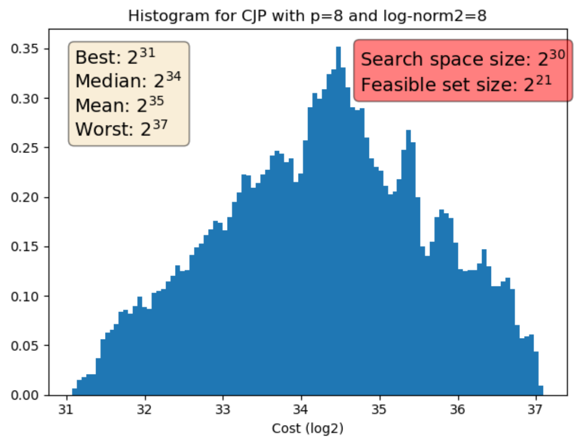
We conducted an experiment for two simple graphs $\mathcal{G}_1 = \{A(p_1, \nu_1)\}$ and $\mathcal{G}_2 = \{A(p_2, \nu_2)\}$ with $p_1 = 2^1$, $p_2 = 2^8$ and $\nu_1 = \nu_2 = 2^8$. For every parameter set in $\mathcal{P}(\mathcal{G}_1)$ (respectively $\mathcal{P}(\mathcal{G}_2)$), we test if it is in the feasible set $\mathcal{S}(\mathcal{G}_1)$ (respectively $\mathcal{S}(\mathcal{G}_2)$). Then, for every set of parameters that is feasible, we estimate the cost of the graph using our cost model (Definition 23). We display the results in Figure 4.2.

As we can see, for $\mathcal{G}_2$, thanks to the optimizer, we find parameters that give a cost 16 times smaller than the average cost for every feasible parameter sets and 8 times faster than the median cost. The feasible set $\mathcal{S}(\mathcal{G}_2)$ represents approximately $\frac{1}{512}$ of $\mathcal{P}(\mathcal{G}_2)$ i.e., 0.2% of $\mathcal{P}(\mathcal{G}_2)$. For $\mathcal{G}_1$, we find parameters that give a cost 256 times smaller than the average cost for every feasible parameter sets and 64 times faster than the median cost. The feasible set $\mathcal{S}(\mathcal{G}_1)$ represents approximately $\frac{1}{8}$ of $\mathcal{P}(\mathcal{G}_1)$ i.e., 12.5% of $\mathcal{P}(\mathcal{G}_1)$.

This study justifies the interest of having an FHE Optimizer. The larger the precision is, the harder it is to find feasible parameters. The smaller the precision is, the greater the gain is between an average solution and the optimal one.



(a) Feasible set of $\mathcal{G}_1 = \{A(p_1, \nu_1)\}$



(b) Feasible set of $\mathcal{G}_2 = \{A(p_2, \nu_2)\}$

Figure 4.2: Study of the feasible set of $\mathcal{G}_1 = \{A(p_1, \nu_1)\}$ and $\mathcal{G}_2 = \{A(p_2, \nu_2)\}$ with $p_1 = 2^1$, $p_2 = 2^8$ and $\nu_1 = \nu_2 = 2^8$ and $p_{\mathsf{fail}} \approx 2^{-14}$. The cost (in $\log_2$) is displayed on the x-axis.

Pareto Front

To further reduce the size of the search space, we can remove some pairs $(\mathfrak{B}, \ell)$ from the search space.

Let's study the special case of the key switch. Note that everything explained here is adaptable to the PBS. Looking at the noise formula (Definition 19) of the key switch, we learn that the key switch adds a term of noise to the input ciphertext. This new error term does not depend on the input noise but it depends on various parameters: the macro-parameters and the decomposition parameters. Informally, we only want to keep the decomposition parameters that satisfy a different cost and noise trade-off.

More formally, for every set of macro-parameters (k, N, n) and for every level, we iterate over the decomposition base and compute the noise and the cost of the key switch. If the decomposition micro-parameters $(\mathfrak{B}, \ell)$ yield a noise smaller than the noises computed previously but has a greater cost, we can keep the pair as it satisfies a different noise-cost trade-off. If the decomposition parameters $(\mathfrak{B}, \ell)$ yield a cost smaller than the costs computed previously but has a bigger noise, in the same way, we can also keep the pair. After repeating this procedure for every possible macro parameter, we have a list of decomposition parameters each of them satisfying a different trade-off. There are 278 possible pairs in $\{(\log_2(\mathfrak{B}), \ell) \in [\![1, 64]\!]^2 | \log_2(\mathfrak{B}) \cdot \ell \leq \log_2(q)\}$ with $q = 64$ and approximately 50 pairs are left after this filtering.

Branch and Bound

To solve the optimization problem introduced in Equation (4.1), we can use a branch and bound algorithm. One can see the branch and bound as a smart exhaustive search. To make it work, we need to define noise and cost oracles. Given a subset of a search space, the noise oracle estimates the probability of finding a feasible solution in this set. It cannot guarantee that there is a feasible solution but it can guarantee that there is no feasible solution. In this scenario, there is no need to iterate over the parameter sets in this space as none of them will be feasible. The noise oracle is useful to quickly prune part of the search space. In practice, the noise oracle computes a lower bound on the noise for every parameters in the set at hand and compares this lower bound with the noise bound (Definition 21). If the lower bound of the noise does not meet the constraint, we are sure that there are no feasible solution in this set.

In the same way, we can build a cost oracle that takes as input a subset of the search space and computes a lower bound on the cost. If this lower bound is greater than the cost of previously found solutions, we can remove this set from the search as it would yield sub-optimal solutions.

Thanks to those two oracles, we divide the search space in smaller spaces at each step, apply the oracles on each of them and discard the ones that only contain sub-optimal or non-feasible parameter sets.

Concretely, building the cost oracle is easy as the cost formulae are increasing functions of all their parameters. We can then easily compute a minimum of the cost of a set by taking the minimal values of each variable and applying the cost formula. It is trickier to build efficient noise oracles as the formulae are non-monotonic. We build naive oracles by using the following property with f and g two positive functions:

$$\min_{x\in\mathbb{R}^\alpha} (f(x) + g(x)) \geq \min_{x\in\mathbb{R}^\alpha} f(x) + \min_{x\in\mathbb{R}^\alpha} g(x)$$

and

$$\min_{x\in\mathbb{R}^\alpha} f(x) \cdot g(x) \geq \left(\min_{x\in\mathbb{R}^\alpha} f(x)\right) \cdot \left(\min_{x\in\mathbb{R}^\alpha} g(x)\right)$$

A prototype of branch-and-bound following the blueprint explained in this section has been developed in Python.

4.2.4 Full-fledge problem

In Definition 24, we introduced the notion of FHE DAG. Such structure is filled with nodes symbolizing FHE operators or subgraphs of FHE operators. In real-life applications, one owns a graph of computations and wants to deploy an FHE scheme to compute the same graph but over encrypted data. It means that the problem we eventually address is way more complicated than what we previously explained in Section 4.1. Instead of taking a graph of FHE operators, we take as input a crypto-free graph and find simultaneously, the best FHE DAG and its associated cryptographic parameters which guarantee that it behaves as the input DAG but over encrypted inputs. The following definition formalizes this notion.

Definition 31 (Plain DAG) *Let $\overline{\mathcal{G}} = \left(\overline{V}, \overline{E}\right)$ be a DAG of plain operators (Definition 18). We define $\overline{V} = \left\{\overline{\mathcal{O}}_i\right\}_{1\leq i\leq\alpha}$ as the set of vertices, each of them being a plain operator that can be additions, subtractions, multiplications, LUT evaluation, etc. We define $\overline{E}$ as the set of edges, each of them associated with a given message modulus as well as a label which is either* private *or* public*. Private means that the associated message should be encrypted and public means that the associated message can be publicly known. When the set of edges, denoted by $\overline{E}$, is not relevant, we will represent the graph simply by its set of vertices, using the notation $\overline{\mathcal{G}} = \overline{V}$. This implies that in such contexts, the graph $\overline{\mathcal{G}}$ is understood solely through its vertices.*

We note $\mathcal{S}_{\mathsf{FHE}}\left(\overline{\mathcal{G}}\right)$ the set of all possible FHE graphs computing the same functionality than the plain DAG $\overline{\mathcal{G}}$.

Our optimization framework takes as input a plain DAG $\overline{\mathcal{G}}$, a level of security, and a correctness probability. It outputs an FHE DAG $\mathcal{G}$ as well as a parameter set x for $\mathcal{G}$. Remember that most of the FHE operators in $\mathcal{G}$ introduce some cryptographic parameters, for instance a local polynomial size $N \in \mathbb{N}$ or a local base $\mathfrak{B} \in \mathbb{N}$. It implies that the total number of possible parameter sets is exponentially huge and we want x to be the best of them all. Also remember that for a same plain operator, for instance a homomorphic multiplication, there are several possible strategies to translate it into an FHE subgraph. In this example, we could use the leveled multiplication that will be introduced later (Algorithm 22) or two PBS as in [Chi+20b] (see Limitation 4). Those different ways to translate a plain operator into an FHE one make the problem more complex as the size of $\mathcal{S}_{\mathsf{FHE}}\left(\overline{\mathcal{G}}\right)$ depends on the list of known translation rules between plain and FHE operators.

The output of our optimization framework is an FHE DAG $\mathcal{G}$ and its associated parameter set x and must fulfill Guaranties 1 (security), 2 (correctness), 3 (efficiency) in addition to a last one:

4. the FHE DAG computes the plain functionality described in the plain DAG.

The full-fledge problem we want to solve is the following:

$$\left(\widehat{\mathcal{G}}, \widehat{x}\right) = \text{argmin }_{\mathcal{G},x} \mathsf{Cost}\left(\mathcal{G}, x\right) \;\; s.t. \begin{cases} \mathcal{G} \in \mathcal{S}_{\mathsf{FHE}}\left(\overline{\mathcal{G}}\right) \\ x \in \mathcal{S}\left(\mathcal{G}\right) \end{cases} \tag{4.3}$$

To build $\mathcal{S}_{\mathsf{FHE}}\left(\overline{\mathcal{G}}\right)$, we use simple translation rules between plain operators and FHE operators (Definition 18). To compute a lookup table or a function, we can use a KS and a PBS from [Chi+20a] if the precision of the message is between 1 and 8 bits or the WoP-PBS described later in Algorithm 30 for larger precisions. A complete comparison on the efficiency of the different algorithms to perform a LUT evaluation is available in Section 4.3 and in Section 6.2. A multiplication between ciphertexts can be replaced by one of the methods previously introduced. A DP can be replaced by the same DP working over ciphertexts. We will explain in 4.4.1 how to do a better transformation by optimally inserting PBS in the DP.

4.2.5 Failure Probability: From the AP to the Entire Graph

In the previous section, all failure probabilities were associated with one single FHE operator (at least those where there is effectively a risk such as a PBS). We want to extend our framework to work directly with the failure probability of the entire graph.

Observe that it is easy to have an upper bound on the graph failure probability given individual AP failure probabilities. Let $\mathcal{G} = \{A_i\}_{i \in I}$ be an FHE DAG and let's assume, without loss of generality, that $\mathcal{G}$ has an unique output $\mathsf{ct}\left(\widetilde{m_{\mathsf{out}}}\right)$ and that inside each atomic pattern there is only one place where the noise is at its highest. For every $i \in I$, let $\mathsf{ct}\left(\widetilde{m_i}\right)$ be the ciphertext for which the noise is the highest in A_i. Then, the failure probability for the whole graph is bounded by:

$$p_{\mathsf{fail}}\left(\mathcal{G}\right) = \mathsf{Pr}\left(\mathsf{Decode}\left(\mathsf{ct}\left(\widetilde{m_{\mathsf{out}}}\right)\right) \neq m_{\mathsf{out}}\right) \leq 1 - \prod_{i \in I}\left(1 - p_{\mathsf{fail}(A_i)}\right) \tag{4.4}$$

with $p_{\mathsf{fail}}\left(A_i\right) = \mathsf{Pr}\left(\mathsf{Decode}\left(\mathsf{ct}\left(\widetilde{m_i}\right)\right) \neq m_i\right)$, $\forall i \in I$. By applying the *domination* concept presented in Theorem 17, when an atomic pattern A_1 is dominated by an atomic pattern A_2, we can find a relationship between $p_{\mathsf{fail}}\left(A_1\right)$ and $p_{\mathsf{fail}}\left(A_2\right)$.

We start by exposing a simpler case where two APs are compared, before generalizing the method to a whole graph. Let α be the failure probability that we want to guarantee for every atomic pattern, and let $\kappa = z^*\left(\alpha\right)$ be its associated standard score. Let $\left(\nu_1, \nu_2\right) \in \mathbb{R}^2$ s.t. $\nu_1 \leq \nu_2$, and let $\left(p_1, p_2\right)$ be two precisions s.t. $p_1 \leq p_2$, which means that $t_1 = \frac{q}{2^{1+p_1+1} \cdot \kappa} = \frac{\Delta_1}{2\kappa} \geq t_2 = \frac{q}{2^{1+p_2+1} \cdot \kappa} = \frac{\Delta_2}{2\kappa}$. Let $A_1 = A\left(\nu_1, t_1\right)$ and $A_2 = A\left(\nu_2, t_2\right)$ be two atomic patterns of type $\mathcal{A}^{(\mathrm{CJP21})}$.

As $\nu_1 \leq \nu_2$ and $t_1 \geq t_2$, A_1 is dominated by A_2 (Theorem 17). It means that if we have $|e_2| < \frac{\Delta_2}{2}$, we know that with high probability $|e_1| < \frac{\Delta_1}{2}$ where $e_1 \hookleftarrow \mathcal{N}\left(0, \sigma_1^2\right)$ (respectively $e_2 \hookleftarrow \mathcal{N}\left(0, \sigma_2^2\right)$) is the maximal noise in A_1 (respectively A_2).

Following Definition 21, we know that $\sigma_2 \leq t_2 \Rightarrow \mathsf{Pr}\left(|e_2| \geq \frac{\Delta_2}{2} = \kappa \cdot t_2\right) \leq \alpha$. It follows that, if this inequality is verified, we will also have $\mathsf{Pr}\left(|e_1| \geq \frac{\Delta_1}{2}\right) = p_{\mathsf{fail}}\left(A_1\right) \leq \alpha$ (Theorem 17).

At this point, we look for an estimation of the failure probability $p_{\mathsf{fail}}\left(A_1\right)$ as a function of $p_{\mathsf{fail}}\left(A_2\right)$. The noise inside an atomic pattern of type $\mathcal{A}^{(\mathrm{CJP21})}$ is maximal after the modulus switching. With our noise model we have that:

$$\forall i \in I, \sigma_i^2 = \sigma_{\mathsf{BR}}^2 \cdot \nu_i^2 + \sigma_{\mathsf{KS}}^2 + \sigma_{\mathsf{MS}}^2$$

With $\sigma_2 \leq t_2$ and $\sigma_1 \leq t_1$, we have:

$$\sigma_1^2 \leq t_2^2 - \left(\nu_2^2 - \nu_1^2\right) \cdot \sigma_{\mathsf{BR}}^2 \leq t_1^2$$

With the previous inequality, we have found a noise bound tighter than before and we can compute its associated standard score κ_1.

Let us assume that there exists a real number $D \in \mathbb{R}^*$ such that, for every possible set of parameters x, $\sigma_{\mathsf{BR}}^2\left(x\right) \geq D$, i.e., $D = \min_x \sigma_{\mathsf{BR}}^2\left(x\right)$. Using the previous inequality, we have:

$$\begin{aligned} \sigma_1^2 &\leq t_2^2 - \left(\nu_2^2 - \nu_1^2\right) \cdot \sigma_{\mathsf{BR}}^2 \\ &\leq t_2^2 - \left(\nu_2^2 - \nu_1^2\right) \cdot D \\ &= \left(\frac{\Delta_1}{2 \cdot \kappa_1}\right)^2 \leq t_1^2 \end{aligned}$$

and so we have:

$$\kappa_1 = \frac{\Delta_1}{2} \cdot \left(D \cdot (\nu_1^2 - \nu_2^2) + \left(\frac{\Delta_2}{2\kappa} \right)^2 \right)^{-\frac{1}{2}}. \tag{4.5}$$

Using Definition 22, we have $p_{\mathsf{fail}}(A_1) \leq 1 - \mathsf{erf}\left(\frac{\kappa_1}{2}\right)$. To find an adequate D, one can iterate over every possible set of parameters x and find the minimal value for $\sigma_{\mathsf{BR}}(x)$. In particular, if $\nu_1 = \nu_2$, we have $\kappa_1 \geq \frac{\Delta_1}{\Delta_2} \cdot \kappa$.

Using the relationship above, we have a simple algorithm to find parameters that satisfy a given failure probability for a whole graph $\mathcal{G} = \{A_i\}_{i \in I}$. Let $p_{\mathsf{fail}}(\mathcal{G})$ be the failure probability we want to guarantee for the whole graph, and let δ_{fail} be its associated tolerance. The algorithm will output a failure probability $p_{\mathsf{fail}}(A)$ that can be used as described in the sections above and we are sure to achieve a failure probability $\widetilde{p_{\mathsf{fail}}(\mathcal{G})}$ such that $\left|\widetilde{p_{\mathsf{fail}}(\mathcal{G})} - p_{\mathsf{fail}}(\mathcal{G})\right| < \delta_{\mathsf{fail}}$.

First, we build $\mathcal{G}_{\mathsf{pareto}} = \{A_i\}_{i \in I'} \leftarrow \mathsf{Pareto}(\mathcal{G})$, as defined in Section 4.2.2. Then, we set $p_{\mathsf{fail}}(A_{\mathsf{dominant}}) \leftarrow 1 - (1 - p_{\mathsf{fail}}(\mathcal{G}))^{\frac{1}{|I'|}}$. At this stage, we can apply what is described above to find the probability of failure of the dominated atomic patterns i.e., compute $\forall i \in I \setminus I', p_{\mathsf{fail}}(A_{\mathsf{dominated},i})$. Then, using equation 4.4, we can compute

$$\widetilde{p_{\mathsf{fail}}(\mathcal{G})} \approx 1 - (1 - p_{\mathsf{fail}}(A_{\mathsf{dominant}}))^{|I'|} \cdot \prod_{i \in I \setminus I'} (1 - p_{\mathsf{fail}}(A_{\mathsf{dominated},i}))$$

If $\left|\widetilde{p_{\mathsf{fail}}(\mathcal{G})} - p_{\mathsf{fail}}(\mathcal{G})\right| > \delta_{\mathsf{fail}}$, we need to decrease or increase $p_{\mathsf{fail}}(A_{\mathsf{dominant}})$ and to repeat the rest of the algorithm until we meet the condition.

4.3 Comparison of FHE Operators

The atomic pattern types give us a powerful tool to compare several variants of the bootstrap existing in the FHE literature and more generally, to compare different ways to perform the same plain operation. As different bootstrapping techniques have different cost-noise trade-offs, it is hard to compare them. Indeed, it is not possible to analytically compare the cost and the noise formulae of different bootstraps as they do not share any common ground (number, types and values of parameters).

In practice, we want to execute arithmetic circuits with a lot of leveled operations (addition, multiplication by an integer) and lookup table evaluations. We can see an arithmetic circuit as a graph of atomic patterns. Most of those atomic patterns will have input ciphertexts coming from other atomic patterns and not freshly encrypted ciphertexts. The naive way to compare different bootstrapping techniques is to define a set of those circuits and find which variant is the best one to efficiently compute them in FHE. As there is too many circuits to test, this technique is not realistic. In practice, we want to define a few simple circuits and compare the different techniques on those.

Following the observation above, we define an atomic pattern type for each bootstrapping technique such that each atomic pattern type performs the same plain operations. Then, for each atomic pattern type, we define simple graphs only composed of one atomic pattern. To simulate the fact that the atomic pattern is part of a bigger graph composed of chains of atomic patterns, we add the constraint that *the input noise of each atomic pattern is defined as the output noise of the atomic pattern.* With this trick, we can truly compare the cost of different atomic patterns that can perform the same plain operations and decide which technique is faster.

In this section, we compare different ways to perform lookup table evaluations and we study the optimal place of the key switch in an atomic pattern. Other comparisons using the optimization framework will be detailed throughout this thesis.

4.3.1 LUT Evaluation for Different Precisions

In Section 2.3, we recalled the tree-PBS introduced in [GBA21]. In Section 2.4, we explained that this algorithm was the only known solution to apply arbitrary functions on messages encoded with a radix or a CRT encoding. Let us see how this algorithm scales with the precision of the message. To compare it against the PBS, we need to define another atomic pattern type $\mathcal{A}^{(\mathrm{GBA21})}$.

Definition 32 ($\mathcal{A}^{(\mathbf{GBA21})}$ Atomic Pattern Type) *We define a second atomic pattern type $\mathcal{A}^{(GBA21)}$ as a subgraph composed of a dot product (Theorem 5), a key switch (Theorem 6) and the tree-PBS (Algorithm 15) introduced in [GBA21]. As in Definition 30, we assume that every input of the dot product are ciphertexts with independent noises. An atomic pattern AP of type $\mathcal{A}^{(GBA21)}$ is entirely characterized by two values: the 2-norm ν and its noise bound t.*

To compare the PBS of [Chi+20a] in $\mathcal{A}^{(\mathrm{CJP21})}$ and the tree-PBS in $\mathcal{A}^{(\mathrm{GBA21})}$, we solve Equation (4.2) for the two types of atomic patterns on for 4 distinct 2-norms and for message modulus in $\{2^1, \cdots, 2^{24}\}$, and finally plot the results in Figure 4.3. The padding bit is not included in the message precision.

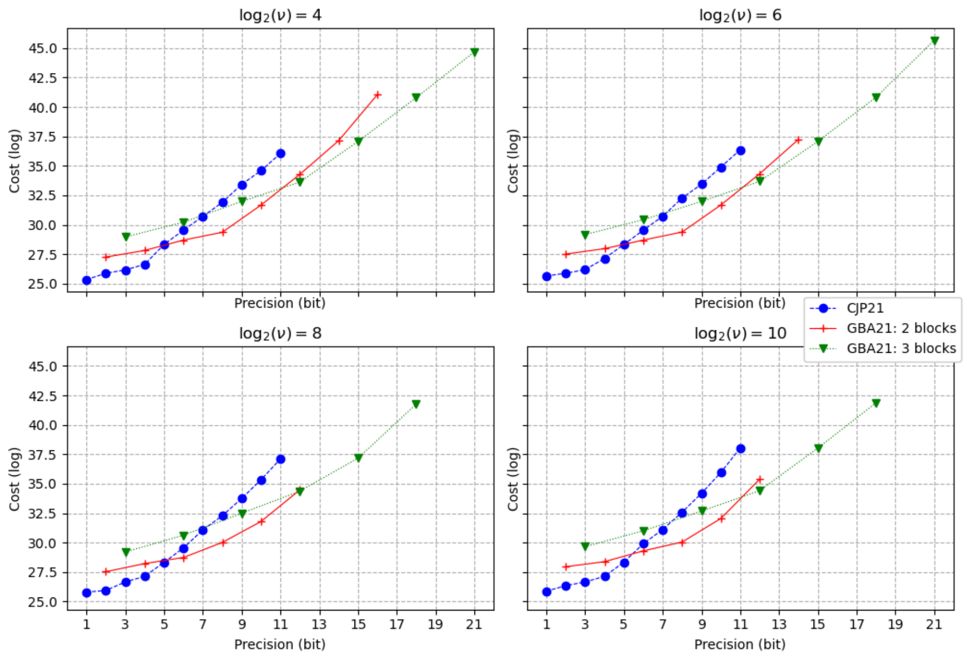


Figure 4.3: Comparison of the cost of AP type $\mathcal{A}^{(\text{CJP21})}$ and AP of type $\mathcal{A}^{(\text{GBA21})}$ with 2 and 3 blocks.

In this experiment, we choose $\mathcal{P}(N) = \{2^1, \cdots, 2^{18}\}$, the search space of the polynomial size N. We set $q = 2^{64}$ and we used a probability of failure $p_{\mathsf{fail}} \approx 2^{-35}$ and one bit of padding (i.e. $\pi = 1$).

Remark 27 (Noise Bound) *For $\mathcal{A}^{(CJP21)}$, the noise bound (Definition 21) is defined as $t(p, 1) = \frac{q}{2^{1+1} \cdot p \cdot z^*(p_{\mathsf{fail}})}$.*

For $\mathcal{A}^{(GBA21)}$, the noise bound needs to be computed differently because this AP with 2 blocks (respectively 3 blocks) involves η_2 (respectively η_3) PBS, all sources of potential failures.

$$\eta_i = i \cdot \frac{p^{i-1} - 1}{p - 1} + 1, \text{ with } i \text{ the number of blocks}$$

To guarantee a global failure probability for one $\mathcal{A}^{(GBA21)}$, the noise bound needs to be computed from the number η_i of PBS. We start by computing the failure probability needed for one PBS defined as $p'_i = 1 - (1 - p_{\mathsf{fail}})^{\frac{1}{\eta_i}}$ and from it we can finally compute the noise *bound for each PBS* $t\left(p,1\right) = \frac{q}{2^{1+1}\cdot p\cdot z^*\left(p_i'\right)}$. *A generalization of this approach is explained in Section 4.2.5.*

The first takeaway is that TFHE's bootstrapping (in atomic pattern $\mathcal{A}^{(\mathrm{CJP21})}$, blue/● curve) can only handle messages up to 11 bits of precision. By using these parameters sets, the cost of this atomic pattern with regards to the precision is an exponential function in two parts. For precisions above 4 to 5 bits (padding bit not included), adding a bit of precision more than doubles the cost, indeed the polynomial size doubles for every additional bit of precision. TFHE PBS does not scale well with the precision, to maximize efficiency, it should not be used when the messages have more than 5 bits of precision.

For $\mathcal{A}^{(\mathrm{GBA21})}$, we used on the first layer the multi-value PBS (Algorithm 14) introduced in [CIM19] and we used PBS over encrypted lookup tables [Chi+20a] on the other layers. The tree-PBS of [GBA21] takes as input a vector of ciphertexts each containing part of the message as explained in Section 2.4. The red/+ curve (respectively green/▼ curve) represents the cost to compute a tree-PBS over 2 ciphertexts (respectively 3 ciphertexts) each one containing a chunk of the message. Using this, we can reach precisions that are not feasible with the bootstrapping from [Chi+20a]. Above 11 bits, we cannot find parameters that will guarantee the correctness of $\mathcal{A}^{(\mathrm{CJP21})}$. Regarding the tree-PBS with 2 blocks, it becomes interesting in terms of cost with 6 bits of precision or more, and offers parameters up to 16 bits of precision. For higher precisions, no feasible solution could be found. The tree-PBS with 3 blocks provides a way to go above that and we found solutions for precisions up to 21 bits. It is more efficient than the other two starting at 10 bits of precision. It is important to notice that even if solutions exist, computing $\mathcal{A}^{(\mathrm{GBA21})}$ over message of 21 bits costs more than 2^{20} times the cost of the [Chi+20a] PBS over Boolean messages.

To conclude this comparison, [Chi+20a]'s bootstrapping used as in [CJP21] (i.e., with a KS before and not after) is the best way to apply a function over messages of small precision (1 to 5 bits). For precision above 11 bits, we have to use the tree-PBS in [GBA21]. But as we can see in the figures, we need an algorithm more efficient than [GBA21] when it becomes too expensive, i.e., above 9 bits, especially if one wants to build efficient operations over larger homomorphic integers with TFHE and still being able to compute LUTs on them. We introduce a new more efficient algorithm to perform LUT evaluations over large integers in Section 6.2 using Algorithm 30.

4.3.2 Keyswitch Position in an Atomic Pattern

In some contexts it is possible to analytically compare two AP types before beginning the optimization, i.e., for all suitable sets of parameters, one of the AP types is always better than the other. For instance, in the gate bootstrapping described in [Chi+20a], an unlimited number of sequences of PBS, KS and DP is described. However one can analytically prove that it is always best to have the KS right before the PBS. We introduce a new atomic pattern type $\mathcal{A}^{(\text{CGGI20})}$.

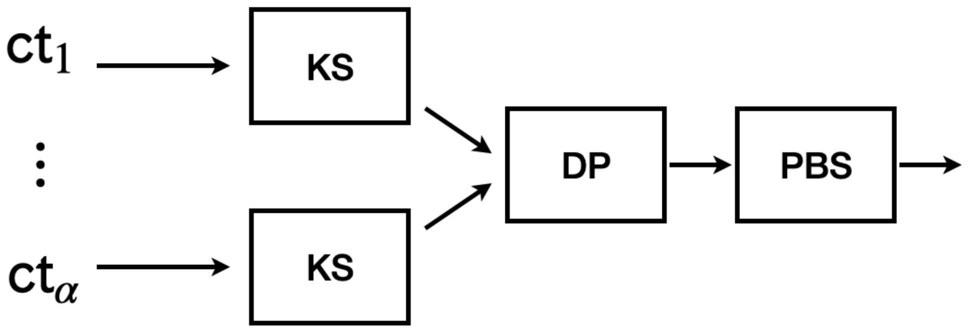


Figure 4.4: $\mathcal{A}^{(\text{CGGI20})}$ atomic pattern type, composed of a layer of key switches (KS), a dot product (DP) and a programmable bootstrap (PBS)

Definition 33 ($\mathcal{A}^{(\text{CGGI20})}$ Atomic Pattern Type) *We define a third atomic pattern type $\mathcal{A}^{(CGGI20)}$ as a subgraph composed of a layer of key switches (Theorem 6), a dot product (Theorem 5) and a PBS (Theorem 14). This subgraph is illustrated in Figure 4.4.*

As in Definitions 30 and 32, we assume that the inputs of the dot product are ciphertexts with independent noises. We can describe an atomic pattern of type $\mathcal{A}^{(CGGI20)}$ by the pair (ν, t) with ν, the 2-norm and t the noise bound (Definition 21).

The following theorem formalizes the comparison between an atomic pattern of type $\mathcal{A}^{(\text{CJP21})}$ and an atomic pattern of type $\mathcal{A}^{(\text{CGGI20})}$.

Theorem 18 (Relation Between $\mathcal{A}^{(\text{CJP21})}$ and $\mathcal{A}^{(\text{CGGI20})}$) *We consider two 2-norms $\nu_1, \nu_2 \in \mathbb{R}^+$ such that $\nu_1 \leq \nu_2$, two noise bounds $t_1, t_2 \in \mathbb{N}$ such that $t_2 \leq t_1$ and two AP: $A_1 \in \mathcal{A}^{(CJP21)}$ and $A_2 \in \mathcal{A}^{(CGGI20)}$. We have $\mathcal{S}\left(A_2\left(\nu_2, t_2\right)\right) \subseteq \mathcal{S}\left(A_1\left(\nu_1, t_1\right)\right)$.*

Proof 18 (Theorem 18) *Let's start with the observation that $\mathcal{A}^{(CJP21)}$ and $\mathcal{A}^{(CGGI20)}$ share the same search space $\mathcal{P}$ because they are built with the same operators. For a* *parameter set $x \in \mathcal{P}$, the maximum noise variance in an AP of type $\mathcal{A}^{(CJP21)}$ is $\sigma^2_{\mathsf{out},1}(x) = \sigma^2_{\mathsf{in}}(x) \cdot \nu^2 + \sigma^2_{\mathsf{KS}}(x) + \sigma^2_{\mathsf{MS}}(x)$ where $\sigma^2_{\mathsf{KS}}(x)$ is the noise added by the KS and $\sigma^2_{\mathsf{MS}}(x)$ is the noise added by the MS. Similarly, the maximum noise variance in an AP of type $\mathcal{A}^{(CGGI20)}$ is $\sigma^2_{\mathsf{out},2}(x) = (\sigma^2_{\mathsf{in}}(x) + \sigma^2_{\mathsf{KS}}(x)) \cdot \nu^2 + \sigma^2_{\mathsf{MS}}(x)$.*

We consider $\bar{x} \in \mathcal{S}(A_2(\nu_2, t_2))$, so we have $\sigma^2_{\mathsf{out},2}(\bar{x}) = (\sigma^2_{\mathsf{in}}(\bar{x}) + \sigma^2_{\mathsf{KS}}(\bar{x})) \cdot \nu_2^2$ and $\sigma^2_{\mathsf{out},2}(\bar{x}) \leq t_2^2$. The simplest non-trivial DP possible is when we only have one input ciphertext multiplied by 1, so we have $1 \leq \nu$. Since variances are positive and $1 \leq \nu_1 \leq \nu_2$, we have:

$$\begin{aligned} t_1^2 \geq t_2^2 \geq \sigma^2_{\mathsf{out},2}(\bar{x}) &= \sigma^2_{\mathsf{in}}(\bar{x}) \cdot \nu_2^2 + \sigma^2_{\mathsf{KS}}(\bar{x}) \cdot \nu_2^2 + \sigma^2_{\mathsf{MS}}(\bar{x}) \\ &\geq \sigma^2_{\mathsf{in}}(\bar{x}) \cdot \nu_2^2 + \sigma^2_{\mathsf{KS}}(\bar{x}) + \sigma^2_{\mathsf{MS}}(\bar{x}) \\ &\geq \sigma^2_{\mathsf{in}}(\bar{x}) \cdot \nu_1^2 + \sigma^2_{\mathsf{KS}}(\bar{x}) + \sigma^2_{\mathsf{MS}}(\bar{x}) = \sigma^2_{\mathsf{out},1}(\bar{x}) \end{aligned}$$

So we have $t_1^2 \geq \sigma^2_{\mathsf{out},1}(\bar{x})$, meaning that $\bar{x} \in \mathcal{S}(A_1(\nu_1, t_1))$, so $\mathcal{S}(A_2(\nu_2, t_2)) \subseteq \mathcal{S}(A_1(\nu_1, t_1))$. □

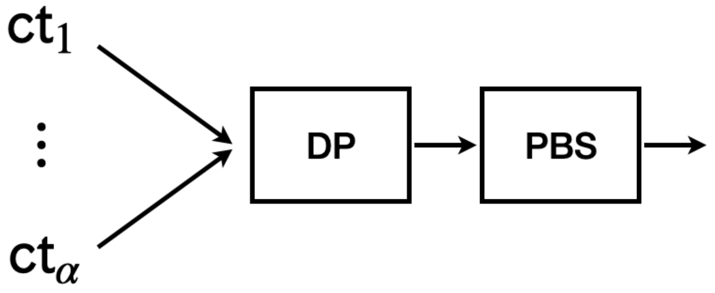


Figure 4.5: $\mathcal{A}^{(\text{KS-free})}$ atomic pattern type, composed of a dot product (DP) and a programmable bootstrap (PBS)

The same result can be found by solving Equation (4.1) for atomic pattern types $\mathcal{A}^{(\text{CJP21})}$ and $\mathcal{A}^{(\text{CGGI20})}$. The curves are shown on Figure 4.6 for precisions up to 11 bits and for four different 2-norm ν. In this figure, we also added the comparison with $\mathcal{A}^{(\text{KS-free})}$, the atomic pattern type composed of a DP and a bootstrapping from [Chi+20a] (without any key switch as illustrated in Figure 4.5). The cost of an atomic pattern of type $\mathcal{A}^{(\text{KS-free})}$ is plotted as the green/▼ curve. The blue/● curve represents the cost of $\mathcal{A}^{(\text{CJP21})}$ and the red/+ curve is the cost of $\mathcal{A}^{(\text{CGGI20})}$.

As we can see, the smallest cost is the one of $\mathcal{A}^{(\text{CJP21})}$ which confirms what we found theoretically by comparing $\mathcal{A}^{(\text{CJP21})}$ and $\mathcal{A}^{(\text{CGGI20})}$ in Theorem 18. We also notice that for precisions larger than 8 bits, there are no feasible parameters for $\mathcal{A}^{(\text{CGGI20})}$. This is due to the fact that the noise of the key switch is amplified by the DP in $\mathcal{A}^{(\text{CGGI20})}$ whereas it is not in $\mathcal{A}^{(\text{CJP21})}$. On the contrary, for very small precisions, $\mathcal{A}^{(\text{CJP21})}$ and $\mathcal{A}^{(\text{CGGI20})}$ are very similar in terms of efficiency. The difference increases as soon as we increase the 2-norm factor ν. It means that for Boolean TFHE (gate bootstrapping described in [Chi+20a]), having the key switch after the bootstrap does not worsen the cost by much, but it is not the case for larger precisions. This figure also illustrates the usefulness of the key switching as $\mathcal{A}^{(\text{KS-free})}$ is always the worst atomic pattern type in terms of cost. Furthermore, for $\mathcal{A}^{(\text{KS-free})}$ no solution is found for precisions larger than 7 bits. To conclude, one always wants to compute the KS right before the PBS as in AP of type $\mathcal{A}^{(\text{CJP21})}$.

Remark 28 (Mixing Different AP Types in an FHE Graph) *We consider an FHE graph $\mathcal{G}$ containing two types of AP: type $\mathcal{A}^{(CJP21)}$ and type $\mathcal{A}^{(CGGI20)}$. We can apply the AP domination (Theorem 17) on every atomic pattern of each type. At the end of this procedure, we end up with a few AP of type $\mathcal{A}^{(CJP21)}$ and a few AP of type $\mathcal{A}^{(CGGI20)}$. To further simplify the problem, we can use Theorem 18 to compare the remaining atomic patterns of $\mathcal{A}^{(CJP21)}$ with the atomic patterns of $\mathcal{A}^{(CGGI20)}$.*

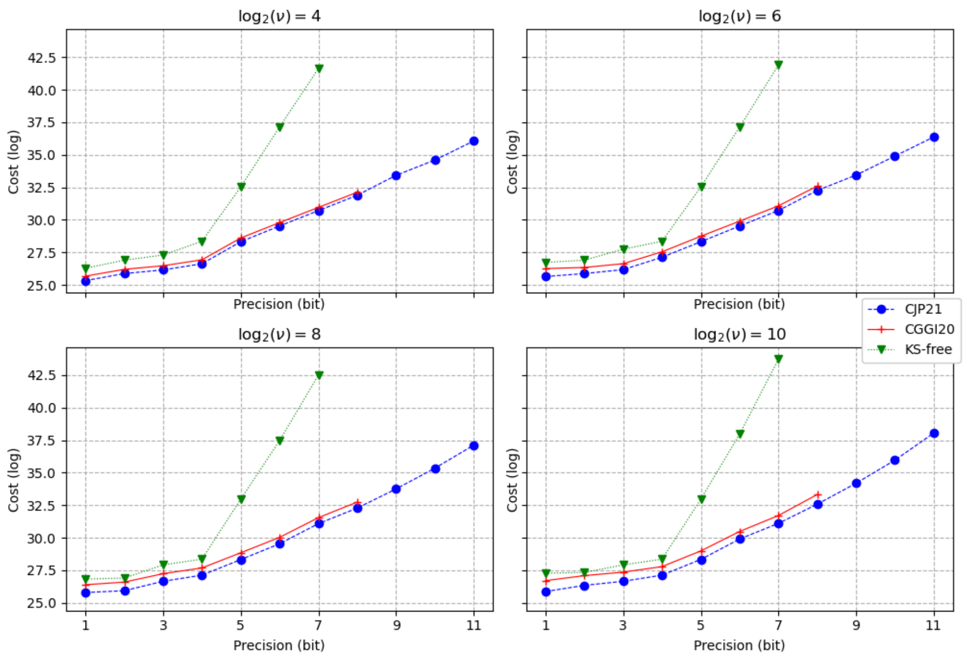


Figure 4.6: Comparison of the AP types $\mathcal{A}^{(\text{CJP21})}$, $\mathcal{A}^{(\text{CGGI20})}$ and $\mathcal{A}^{(\text{KS-free})}$.

4.4 Other applications

In this section, we describe several improvements we designed for our optimization framework. First, we explain how to easily insert programmable bootstraps inside dot products. Then, we explain how to mitigate the stochastic property of the FFT to build a Consensus-friendly TFHE i.e., a way to instantiate TFHE which guarantees that a circuit executed on different hardware targets but with the same input ciphertexts and public material produces the same output ciphertexts. Finally, we offer an efficient way to find several keyswitching keys for a given circuit.

4.4.1 Optimal PBS Insertion within a Dot Product

In Section 4.2.4, we suggested to translate a plain DP into an FHE DP. Here, we explain how to automatically insert a PBS during a DP wherever it is interesting with regards to the cost model. It can sound counterintuitive as the PBS can be a very costly operator, hence inserting PBSs will increase the total cost of the computation. However when the 2-norm of a DP operator is high, the parameters must be large enough to still guarantee the correctness of the computation and those large parameters will have an impact on the cost. We need our framework to choose whether it is interesting to split the DP operator or not to end up with more PBSs but with smaller cryptographic parameters.

The following theorem explores this approach.

Theorem 19 (DP Splitting) *Let $A(\nu, t)$ be an AP of type $\mathcal{A}^{(CJP21)}$ with a noise bound t and including a DP of 2-norm ν. Let $\widetilde{A}(\nu, t, d)$ the same AP than A but where its DP is split into $d+1$ sub-DP of approximately the same 2-norm and connected together with PBS. It actually breaks an AP into $d+1$ APs of the same type organised in two layers (d followed by a last one connecting them all).*

Let $\mathcal{G} = \{A(\nu_i, t)\}_{0\leq i<Y}$ such that $\nu_0 < \nu_1 < ... < \nu_{Y-1}$. Let $\vec{d^*} = (d^*_0, \cdots, d^*_{Y-1})$ and $\vec{d'} = \left(d'_0, \cdots, d'_{Y-1}\right)$ be two different possible splitting solutions. We define the following two FHE graphs: $\mathcal{G}^* = \left\{\widetilde{A}(\nu_i, t, d^*_i)\right\}_{0\leq i<Y}$ and $\mathcal{G}' = \left\{\widetilde{A}(\nu_i, t, d'_i)\right\}_{0\leq i<Y}$.

If every coordinate of $\vec{d^*}$ is inferior or equal (coordinate wise) to the ones from $\vec{d'}$ and $\mathcal{S}(\mathcal{G}^*) = \mathcal{S}(\mathcal{G}')$, then, d' cannot be the optimal solution.

Proof 19 (Theorem 19) *As every coordinate of $\vec{d^*}$ is inferior or equal to the ones from $\vec{d'}$, there are strictly more atomic patterns in $\mathcal{G}'$ than in $\mathcal{G}^*$. It means that $\forall x \in \mathcal{S}(\mathcal{G}^*) = \mathcal{S}(\mathcal{G}'), \mathsf{Cost}(\mathcal{G}^*, x) < \mathsf{Cost}(\mathcal{G}', x)$.*

Thus, we have

$$\min_{x\in\mathcal{S}(\mathcal{G}^*)} \mathsf{Cost}\left(\mathcal{G}^*, x\right) < \min_{x\in\mathcal{S}(\mathcal{G}')} \mathsf{Cost}\left(\mathcal{G}', x\right)$$

To conclude, $\mathcal{G}'$ cannot be an optimal solution as $\mathcal{G}^*$ is a strictly better one.

$\square$

Let us study how to choose d, the additional number of atomic patterns after the splitting. We consider a graph $\mathcal{G} = \{A(\nu_i, t)\}_{0\leq i<\alpha}$ composed of AP of type $\mathcal{A}^{(\text{CJP21})}$ which involves a DP operator. We introduce a new AP type and translate every AP of type $\mathcal{A}^{(\text{CJP21})}$ into this new AP type $A^* \in \mathcal{A}^{(\text{CJP21}^*)}$ which has the exact same subgraph than AP type $\mathcal{A}^{(\text{CJP21})}$ except that the DP is split into several sub-DP connected with PBS as explained in Theorem 19. This AP has a new parameter d_i describing the splitting of the DP for the i-th AP, i.e., how many sub-DP we will have. Note that a graph $\mathcal{G}^*$ with fixed values d_i can be viewed as a graph $\widetilde{\mathcal{G}^*}$ of AP type $\mathcal{A}^{(\text{CJP21})}$.

Formally, Equation (4.3) can be written again:

$$\left(\widehat{\mathcal{G}}, \widehat{x}\right) = \underset{\vec{d}\in\mathcal{D}}{\arg\min}\, \mathsf{Cost}\left(\mathcal{G}^*_{\vec{d}}, x^*\right) \;\; s.t. \begin{cases} \mathcal{G}^*_{\vec{d}} = \{A^*(\nu_i, t, d_i)\}_{0\leq i<\alpha} \\ x^* \text{ solution of Equation (4.1) on } \mathcal{G}^*_{\vec{d}} \end{cases} \tag{4.6}$$

Several ways can be imagined to split a DP. One could be to group public weights of the DP into d_i sets such that their 2-norm is approximately the same. This will yield the best result if we keep neglecting the cost of the DP. Inserting a PBS adds extra operations to perform, but it will also reduce the 2-norm of the initial DP.

To find how to split in two a dot product with a 2-norm ν and with weights $\{\omega_i\}_{i\in J}$, we can solve the following problem

$$\min_{\omega_{i,1},\Lambda} \max(\nu_1, \nu_2) \text{ s.t. } \begin{cases} \omega_i = \omega_{i,1}\Lambda + \omega_{i,2} \\ \mathsf{GCD}\left(\omega_{i,1}, \Lambda\right) = 1 \\ \forall j \in \{1,2\}, \nu_j = \sqrt{\sum_i \omega_{j,1}^2} \end{cases} \tag{4.7}$$

This will yield two sets of weights $\{\omega_{i,1}\}_{i\in J}$ and $\{\omega_{i,2}\}_{i\in J}$ with $\nu_1^2 = \sum_{i\in J}\omega_{i,1}^2 \approx \nu_2^2 = \sum_{i\in J}\omega_{i,2}^2$. Those sets of weights can be used to optimally split a dot product as explained in Theorem 5. Those sets of weights for different values of the splitting factor d can be precomputed before beginning the optimization of the whole problem.

If the weights $\{\omega_i\}_{i\in J}$ are unknown but we know that they follow a uniform distribution between -2^p and 2^p, we have a trivial way yet very efficient (regarding the noise) to split a DP into $d+1$ DPs. We can radix-decompose as defined in Section 2.3.2 all the DP weights with the level being equal to $d+1$ and the $\log_2$ of the base $\mathfrak{B}$ is equal to $\frac{p+1}{d+1}$. Here each Λ_i is equal to a power of $\mathfrak{B}$.

4.4.2 Consensus-friendly TFHE

Two implementations of the same FHE algorithm that do not involve the FFT will output the same result as long as they operate over the same inputs (same ciphertexts and same public material). For instance, different implementations of a DP or an LWE-to-LWE KS will produce the same outputs.

However, implementations that leverage the FFT output different ciphertexts depending on the FFT algorithm used. To highlight this, we made an experiment with the traditional parameter set of TFHE-lib [Chi+16b] for the bootstrapping. We use the same secret keys, the same bootstrapping key and the same input ciphertexts, but two different implementations of the PBS with their respective FFT implementations.

We computed the difference on the resulting ciphertexts for the two different implementations, we call this value the *FFT error* of the ciphertexts, which is different from the noise needed for security in the plaintext. We observed that the ciphertexts had the same most significant bits but their least significant bits were different. We also re-run the experiment with different parameters: more levels and larger polynomials in the bootstrapping key. The messages encrypted were still correct but the ciphertexts were completely different i.e., re-randomized.

From those experiments, we can conclude that for a given parameter set and a ciphertext with a given input FFT error, the PBS with a given FFT either resets the FFT error to a minimal level or outputs the maximum amount of FFT error, i.e., a re-randomization of the ciphertext. It means that an FHE circuit containing a DP, a KS and a PBS will not output the exact same ciphertext if it is run on the same inputs with different implementations. This is not compatible with use-cases where one actually needs to guarantee reproducibility across different implementations.

Thankfully, it is possible to ensure the reproducibility by tweaking a bit our optimization framework as well as the PBS algorithm. The idea is to use a new AP type that is identical to type $\mathcal{A}^{(\mathrm{CJP21})}$ but with an extra rounding step at the end (right after the PBS). This rounding procedure aims to remove the LSB of the ciphertexts that are different from an implementation to the other. This rounding increases the noise in the plaintext and it adds a new parameter to optimize: the location of the rounding. The higher in the MSB we round, the more noise we add, but also the more FFT error we remove. Note that this rounding will either keep the same amount of FFT error (when the output FFT error is maximal, i.e., the ciphertexts are completely different) or cancel it entirely depending on the parameter for the rounding and the input FFT error.

We can add to the optimization framework a new constraint related to the maximum FFT error we want to consider. We can do that easily with an additional feasible set $\mathcal{S}_{\mathsf{other}}(\mathcal{G})$. The condition could be represented as $\mathcal{S}_{\mathsf{other}}(\mathcal{G}) = \{x \in \mathcal{P} | \forall i, \mathbb{E}_{\mathsf{PBS}_i}(x) = 0\} \subset \mathcal{P}$, with $\mathbb{E}_{\mathrm{PBS}_i}(x)$ the FFT error of the output ciphertext coefficients of a bootstrapping after the rounding. This approach relies on the fact that we have a model for the FFT error in the output of the PBS in terms of ciphertext coefficients.

Some cryptographic observations can help reduce the size of the parameter space. In particular, we must have $\frac{q}{2\mathfrak{B}_{\mathsf{PBS}}^{\ell_{\mathsf{PBS}}}} > \mathsf{error}_{FFT}(x)$ where $(\ell_{\mathsf{PBS}}, \mathfrak{B}_{\mathsf{PBS}})$ are the decomposition parameters of the PBS.

This optimization enables to set a limit in terms of FFT error in the ciphertext coefficients that an implementation of the FFT introduces. Then, we can optimize for a given FFT error model, some noise model, and some cost model targeting a common architecture for instance. The result of the optimization can be used to compute the same circuit on the same ciphertexts with the same public keys, but with a different implementation of the same FHE algorithms and we will end up with the exact same ciphertext as output.

This feature enables many miners in a blockchain for instance, to compute the same circuit on the same inputs and have a consensus without a need to decrypt anything. It guarantees that the result came out of the desired FHE DAG and not another circuit designed by an attacker.

4.4.3 Several Evaluation Keys

In previous results on atomic pattern type $\mathcal{A}^{(\mathrm{CJP21})}$, we assumed that we only have one public key material per FHE operator for the whole FHE DAG as we were only looking for one polynomial size, one GLWE dimension and one LWE dimension.

Restricting the number of public keys helps to have small total public key material. It also has an impact on the complexity of the optimization problem because as a result, parameters are shared across the entire FHE DAG. The downside is that we cannot speed up parts of the FHE DAG that have a higher noise bound or fewer leveled operations (smaller 2-norm) with smaller and faster parameters.

In this section we describe a simple optimization problem: one LWE secret key $\vec{s}$ and one GLWE secret key $\vec{S'}$ (for the PBS) that can be viewed as a bigger LWE secret key $\vec{s'}$, along with X different key switching keys going from $\vec{s'}$ to $\vec{s}$. These key switching keys can use a different base $\mathfrak{B}$ and/or a different number of levels ℓ. In this context, we want to find parameters for a graph $\mathcal{G}$ of Y APs of type $\mathcal{A}^{(\mathrm{CJP21})}$.

There are many ways to solve this problem. A naive solution is to consider different parameters for each KS and to let every KS have its own $(\mathfrak{B}, \ell)$ and to add the constraint that they can have at most X values. This approach increases exponentially the search space of the optimization problem, so we will not consider it.

A second solution, which is straightforward though not the most efficient one, is to introduce a new variable δ for each KS. This value δ stores the associated KSK identifier. This is a new parameter to optimize for each KS, we have $\forall i \in [1, Y]\, \delta_i \in [1, X]$. So our additional search space, defined by the problem of finding which KSK is used by which KS, is of size X^Y.

We designed a third solution to solve the problem. Starting from now, we will sort our KSK $(\mathsf{KSK}_0, \mathsf{KSK}_1, \cdots)$ such that a KS with KSK_i adds less noise than using KSK_{i+1} for all $0 \leq i$. Ranking the noise added by the key switching keys is useful for the following theorem.

Theorem 20 (Optimal KSK) *Let* KSK_0 *and* KSK_1 *two KS keys with decomposition parameters selected through the resolution of the optimization problem. Without loss of generality, let us assume that a KS using* KSK_0 *adds strictly less noise than the one using* KSK_1*, then the KS with* KSK_0 *will be slower than the KS with* KSK_1*.*

Proof 20 (Theorem 20) *The two keys must have different parameters for the base and/or the number of level, because the noise added is different by hypothesis. If the optimization has selected two distinct keys it means that they both satisfy a different cost/noise trade-off. Then, if a KS with* KSK_1 *is slower than a KS with* KSK_0 *and generates more noise,* KSK_1 *will always be worse (both in terms of noise and cost) than* KSK_0 *which contradict the fact that the optimization has selected those two distinct keys.* □

Let us consider the following toy example where $t \in \mathbb{N}$ is a noise bound (Definition 21) and $\mathcal{G} = \{A_0\,(\nu_0, t)\,, A_1\,(\nu_1, t)\,, A_2\,(\nu_2, t)\}$ is an FHE DAG (Definition 24) such that $\nu_0 < \nu_1 < \nu_2$. This graph has the same noise bound t for each AP and they are all of type $\mathcal{A}^{(\text{CJP21})}$. Let us have two possible key switching keys and let us assume that $\vec{\delta} = (\delta_0, \delta_1, \delta_2) = (0, 1, 0)$ is the optimal solution i.e., we use KSK_0 for A_0 and A_2, and KSK_1 for A_1. We will show that it cannot be so. We can use Theorem 17 to infer that $\mathcal{S}\left(A\left(\nu_2, t\right)\right) \subseteq \mathcal{S}\left(A\left(\nu_1, t\right)\right) \subseteq \mathcal{S}\left(A\left(\nu_0, t\right)\right)$, and Theorem 20 to infer that a KS with KSK_1 is faster than a KS with KSK_0. It is then straightforward to see that if $(0, 1, 0)$ is a solution, then $(1, 1, 0)$ (KSK_0 for A_2 and KSK_1 for A_0 and A_1) is also a solution but a faster one. This example can be extended to an arbitrary number of AP sharing the same noise bound and for an arbitrary number of key switching keys as described in the Theorem 21 below.

Theorem 21 (Several KSK) *Let $\mathcal{G} = \left\{A\left(\nu_i, t\right)\right\}_{0 \leq i < Y}$ an FHE graph only composed of AP type $\mathcal{A}^{(CJP21)}$ such that $\nu_0 < \nu_1 < \nu_2 < \cdots < \nu_{Y-1}$. We consider that we can have X KSK. The optimal $\vec{\delta} = (\delta_0, \cdots, \delta_{Y-1})$ has the property that for all $0 \leq i < Y - 1$ there is $\delta_i \geq \delta_{i+1}$.*

Proof 21 (Theorem 21) *Let us assume that there exists an i^* such that $\delta_{i^*} < \delta_{i^*+1}$ and that $\forall i \in [\![0, Y-1]\!] \setminus \{i^*\}, \delta_{i^*} \geq \delta_{i^*+1}$.*

Following the same logic as in the toy example above, we can use Theorem 17 and we find that

$$\mathcal{S}\left(A\left(\nu_{Y-1}, t\right)\right) \subseteq \cdots \subseteq \mathcal{S}\left(A\left(\nu_1, t\right)\right) \subseteq \mathcal{S}\left(A\left(\nu_0, t\right)\right).$$

Let $\vec{\delta'}$ such that $\forall i \in [\![0, Y-2]\!] \setminus \{i^*\}, \delta'_i = \delta_i$ and $\delta'_{i^*} = \delta_{i^*+1}$. As $\mathcal{S}\left(A_{i^*+1}\right) \subseteq \mathcal{S}\left(A_{i^*}\right)$, δ' is a feasible solution.

Using Theorem 20, we know that KSK_Y is faster than KSK_{Y-1} which is faster than KSK_{Y-2} and so on. It means that $\mathsf{KSK}_{\delta_{i^*+1}}$ is faster than $\mathsf{KSK}_{\delta_{i^*}}$.

To conclude $\vec{\delta'}$ is a better solution than $\vec{\delta}$, so $\vec{\delta}$ cannot be the optimal solution.

□

Using Theorem 21, we can solve the optimization problem without considering those $\vec{\delta}$ that cannot be an optimal solution of Equation (4.1).

We can consider an FHE DAG with different noise bounds, and apply what we just described for each of the different noise bounds. The same approach works to enable several BSKs in the optimization. Indeed, BSKs can also be sorted according to the amount of noise they produce in their output ciphertexts.

Conclusion. In this chapter, we introduced our optimization framework and its building blocks. We formalized the optimization problem in a way that satisfies the guarantees we want to have, we explained different simplifications to reduce the complexity of this problem and we introduced the concept of atomic pattern. Using this concept, we explained how to correctly compare bootstrapping techniques and more generally different atomic pattern types. We also introduced several additional applications that extend our optimization framework.

In the next chapter, we will introduce new FHE operators that remove some limitations explained in Section 2.3. To find parameters for those new algorithms, we will use the optimization framework described in this chapter. We will also use the method explained in Section 4.3 to compare our new methods against the state of the art.

Chapter 5

New FHE Operators

In Section 2.3, we introduced the main building blocks of TFHE. In this chapter, we describe new algorithms that improve these building blocks.

First, we introduce a generalized version of the modulus switch (Algorithm 6) on which we can build a generalized version of the PBS (see Algorithm 17). The modulus switch as described in Algorithm 6 applies by default the rounding on the most significant bits of the plaintext. With this new version of the PBS, we can choose exactly which part of the plaintext we want to extract.

In Limitation 6, we saw that the only existing technique to evaluate several lookup tables over the same input at the same time was the [CIM19]-PBS, recalled in Algorithm 14. With the help of the generalized PBS, we define a new algorithm (Algorithm 18) that can do the same without any additional cost but with a slightly bigger noise which is independent of the lookup tables compared to the PBS.

In Limitation 2, we explained that TFHE bootstrap (Algorithm 10 and Theorem 14) was very slow for messages with more than 8 bits of precision as the polynomial size N is directly constrained by the number of bits of precision. We introduce in Algorithm 19, the rounded PBS that takes as input a message with high precision (more than 8 bits), rounds it to a smaller message and evaluates a lookup table on it. This algorithm is a good alternative to the PBS when the function we want to evaluate is independent from the least significant bits of the message.

In Limitation 4, we recalled that there is no native multiplication algorithm between LWE ciphertexts. The only existing methods are very costly and based on several PBSs (Algorithm 10 and Theorem 14) or on circuit bootstraps (Algorithm 11) and external products (Algorithm 7 and Theorem 11). In this chapter, we introduce an LWE multiplication and several variants that are inspired by the GLWE multiplication algorithm from [FV12].

5.1 Generalized PBS

Algorithm 17: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{GenPBS}\left(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \mathsf{CT}_f, \varkappa, \vartheta\right)$

Context:
$$\begin{cases} \vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n : \text{ the input LWE secret key} \\ \vec{s}' = (s'_0, \cdots, s'_{kN-1}) \in \mathbb{Z}_q^{kN} : \text{ the output LWE secret key} \\ \vec{S}' = \left(S'_0, \ldots, S'_{k-1}\right) \in \mathfrak{R}_q^k : \text{ a GLWE secret key} \\ \forall 0 \le i \le k-1,\ S'_i = \sum_{j=0}^{N-1} s'_{i\cdot N+j} X^j \in \mathfrak{R}_q \\ P_f \in \mathfrak{R}_q : \text{an } r\text{-redundant LUT for } x \mapsto f(x) \\ \Delta_{\mathsf{out}} \in \mathbb{Z}_q : \text{ the output scaling factor} \\ f : \mathbb{Z} \to \mathbb{Z} : \text{a function} \\ (\beta, m') = \mathsf{PTModSwitch}_q(m, \Delta_{\mathsf{in}}, \varkappa, \vartheta) \in \{0,1\} \times \mathbb{N} \end{cases}$$

Input:
$$\begin{cases} \mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}}(m \cdot \Delta_{\mathsf{in}}) = (a_0, \cdots, a_{n-1}, a_n = b) \in \mathbb{Z}_q^{n+1} \\ \mathsf{BSK} = \left\{\overline{\overline{\mathsf{CT}}}_i \in \mathsf{GGSW}_{\vec{S}'}^{\beta,\ell}(s_i)\right\}_{i=0}^{n-1} : \text{ a bootstrapping key from } \vec{s} \text{ to } \vec{S}' \\ \mathsf{CT}_f \in \mathsf{GLWE}_{\vec{S}'}\left(P_f \cdot \Delta_{\mathsf{out}}\right) \in \mathfrak{R}_q^{k+1} \\ (\varkappa, \vartheta) \in \mathbb{Z} \times \mathbb{N} : \text{define along with } N \text{ the chunk of the plaintext to bootstrap} \end{cases}$$

Output: $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s}'}\left((-1)^{\beta} \cdot f(m') \cdot \Delta_{\mathsf{out}}\right)$ if we respect the requirements of Theorem 22

```
1 begin
      /* modulus switching                                        */
2     for $0 \le i \le n$ do
3         $a'_i \leftarrow \left[\left\lfloor \frac{a_i \cdot 2N \cdot 2^{\varkappa-\vartheta}}{q} \right\rceil \cdot 2^{\vartheta}\right]_{2N}$
4     end
      /* blind rotate of the LUT (Algorithm 9)                    */
5     $\mathsf{CT} \leftarrow$ BlindRotate $(\mathsf{CT}_f, \{a'_i\}_{i=0}^{n}, \mathsf{BSK})$ ;
      /* sample extract the constant term (Algorithm 5)           */
6     $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{SampleExtract}(\mathsf{CT}, 0)$
7 end
```

We propose a *more versatile* algorithm for the PBS (Algorithm 10) where we are able to bootstrap a precise chunk of bits of the encrypted plaintext, instead of only the MSB as described in TFHE [Chi+20a], and to also apply several function evaluations at once. We describe this generalization in Algorithm 17. We introduce two *new parameters*, $\varkappa$ and ϑ, which redefine the *modulus switching* step of the TFHE PBS. In particular, $\varkappa$ defines the number of MSB that are not considered in the PBS, while 2^{ϑ} defines the number of functions that can be evaluated at the same time in a single generalized PBS.

The two parameters $\varkappa$ and ϑ are illustrated in Figure 5.1, where *input* represents the plaintext (with noise) which is encrypted inside the input ciphertext of the modulus switching, and *output* illustrates the plaintext (with noise) that is encrypted inside the output ciphertext (after modulus switching). The first $\varkappa$ MSB will not impact the following steps of the generalized PBS and ϑ bits will be set to 0 in order to encode 2^ϑ functions in the LUT stored in P_f (see Section 5.2 and Algorithm 18 for more details). Observe that the case $(\varkappa, \vartheta) = (0, 0)$ corresponds to the original TFHE PBS.

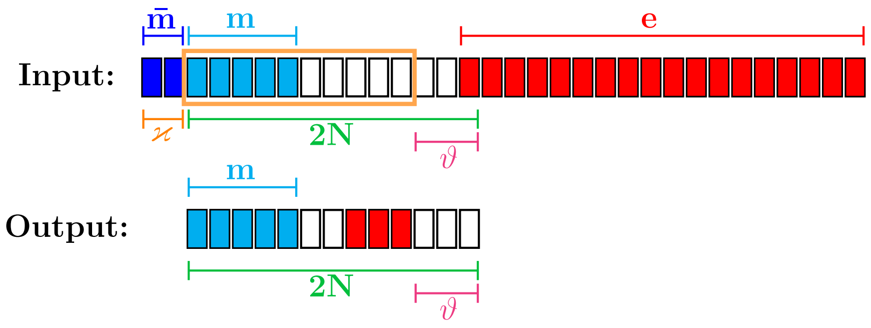


Figure 5.1: Modulus switching operation in the generalized PBS (Algorithm 17): on top of the figures we illustrate the data $(\bar{m}, m, e)$, on the bottom the dimensions ($\varkappa$, $2N$, ϑ).

We also define the *plaintext modulus switching* function written $\mathsf{PTModSwitch}$ to recover the plaintext of the encrypted output of a modulus switching algorithm. Let $m \in \mathbb{Z}_p$ be a message, $\Delta \in \mathbb{Z}_q$ its scaling factor (as defined in Definition 13), $\varkappa \in \mathbb{Z}$ and $\vartheta \in \mathbb{N}$ the parameters of a modulus switching. We define $q' = \frac{q}{\Delta 2^\varkappa}$. The case where $\varkappa \geq 0$ is illustrated in Figure 5.2. We define $(\beta, m') \leftarrow \mathsf{PTModSwitch}_q(m, \Delta, \varkappa, \vartheta) \in \{0,1\} \times \mathbb{N}$ as follows:

$$\text{If } \varkappa \geq 0: \begin{cases} m' = m \mod \frac{q'}{2} \\ \text{if } m \mod q' < \frac{q'}{2},\ \beta = 0, \text{ else } \beta = 1 \end{cases} \qquad \text{Else }: \begin{cases} m' = m \\ \beta \text{ is a random bit} \end{cases}$$

Note that for the sake of simplicity, we provide the generalized PBS noise formula *only for binary secret keys.* However, Appendix A.2 provides formulae as well as proofs for other key distributions (binary, ternary and Gaussian).

Theorem 22 (Generalized PBS) *Let $\vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n$ be a binary LWE secret key. Let $\vec{S}' = \left(S'_0, \ldots, S'_{k-1}\right) \in \mathfrak{R}_q^k$ be a GLWE binary secret key such that $s'_i = \sum_{j=0}^{N-1} s'_{i\cdot N+j} \cdot X^j$, and $\vec{s}' = (s'_0, \cdots, s'_{kN-1})$ be the corresponding binary LWE secret key. Let P_f be a r-redundant LUT for a function $f : \mathbb{Z} \to \mathbb{Z}$ (Definition 12) and Δ_{out} be the output scaling*

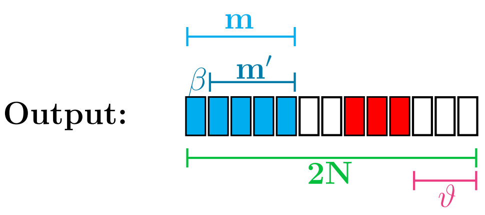


Figure 5.2: Plaintext after the modulus switching from the generalized PBS (Algorithm 17) where $\varkappa \geq 0$: on top of the figure we illustrate the data(m, β, m'), on the bottom the dimensions ($2N$, ϑ).

factor. Let $(\varkappa, \vartheta)$ be the two integer variables defining (along with N) the window size to be modulus switched, such that $\frac{q2^{\vartheta}}{\Delta_{\mathsf{in}}2^{\varkappa}} < 2N$, and let $(\beta, m') = \mathsf{PTModSwitch}_q(m, \Delta_{\mathsf{in}}, \varkappa, \vartheta) \in \{0,1\} \times \mathbb{N}$.

Then Algorithm 17 takes as input an LWE ciphertext $\mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}}(m \cdot \Delta_{\mathsf{in}}) \in \mathbb{Z}_q^{n+1}$ with noise distribution $\chi_{\sigma_{\mathsf{in}}}$, a bootstrapping key $\mathsf{BSK}=\left\{\overline{\overline{\mathsf{CT}}}_i \in \mathsf{GGSW}_{\vec{S}'}^{\mathfrak{B},\ell}(s_i)\right\}_{i=0}^{n-1}$ from $\vec{s}$ to $\vec{S}'$ and a (possibly trivial) GLWE encryption of $P_f \cdot \Delta_{\mathsf{out}}$, and returns an LWE ciphertext $\mathsf{ct}_{\mathsf{out}}$ under the secret key $\vec{s'}$, encrypting the message $(-1)^{\beta} \cdot f(m') \cdot \Delta_{\mathsf{out}}$ with probability $1 - p_{\mathsf{fail}}$ if and only if the input noise has variance $\sigma_{\mathsf{in}}^2 < \frac{\Delta_{\mathsf{in}}^2}{4 \cdot z^*(p_{\mathsf{fail}})^2} - \frac{q'^2}{12w^2} + \frac{1}{12} - \frac{nq'^2}{24w^2} - \frac{n}{48}$, where $z^*(p_{\mathsf{fail}})$ is the standard score (Definition 22) associated to the failure probability p_{fail}, $w = 2N \cdot 2^{-\vartheta}$ and $q' = q \cdot 2^{-\varkappa}$.

The output noise after the generalized PBS is estimated by the formula:

$$\begin{aligned}\mathsf{Var}(\mathsf{PBS}) = n\ell(k+1)N\frac{\mathfrak{B}^2+2}{12}\mathsf{Var}(\mathsf{BSK}) + n\frac{q^2-\mathfrak{B}^{2\ell}}{24\mathfrak{B}^{2\ell}}\left(1+\frac{kN}{2}\right) \\ + \frac{nkN}{32} + \frac{n}{16}\left(1-\frac{kN}{2}\right)^2\end{aligned} \tag{5.1}$$

The cost of Algorithm 17 is the same as the complexity of the TFHE PBS [Chi+20a], i.e.,

$$\begin{aligned}\mathsf{Cost}\left(GenPBS\right)^{(n,\ell,k,N)} = \mathsf{Cost}\left(ModulusSwitching\right)^{(n)} + n\mathsf{Cost}\left(CMUX\right)^{(\ell,k,N)} \\ + \mathsf{Cost}\left(SampleExtract\right)^{(N)}\end{aligned} \tag{5.2}$$

with

$$\mathsf{Cost}\left(ModulusSwitching\right)^{(n)} = (n+1)\mathsf{Cost}\left(Scale\&Round\right) \tag{5.3}$$

$$\begin{aligned}\mathsf{Cost}\left(CMUX\right)^{(\ell,k,N)} &= (k+1)\mathsf{Cost}\left(Rotation\right)^{(N)} + 2(k+1)N\mathsf{Cost}\left(Add\right)\\ &+ n\mathsf{Cost}\left(ExternalProduct\right)^{(\ell,k,N)}\end{aligned} \tag{5.4}$$

$$\begin{aligned}\mathsf{Cost}\left(ExternalProduct\right)^{(\ell,k,N)} &= (k+1)N\mathsf{Cost}\left(dec\right)^{(\ell)} + \ell(k+1)\mathsf{Cost}\left(FFT\right)\\ &+ (k+1)\ell(k+1)N\mathsf{Cost}\left(multFFT\right)\\ &+ (k+1)(\ell(k+1)-1)N\mathsf{Cost}\left(addFFT\right)\\ &+ (k+1)\mathsf{Cost}\left(iFFT\right)\end{aligned} \tag{5.5}$$

Proof 22 (Theorem 22) *In the proof, we compute the decryption of the resulting ciphertext, obtaining the message plus the noise so we can estimate its variance. The detailed proof of this theorem is provided in Appendix A.2.* □

In the next section, we use the generalized PBS to build a variant of the PBS that is able to evaluate several lookup tables on the same input at the same time.

5.2 Many-LUT PBS

Using the generalized PBS, we are able to extract any chunk of the encrypted plaintext with ϑ, $\varkappa$ and N (see Section 5.1). When possible, one can define a smaller chunk for the plaintext by trimming the bound in the LSB using a $\vartheta > 0$. It means that after the modulus switching there are ϑ LSB set to 0. More formally, after the modulus switching, a plaintext m^* will be of the form $m^* = m \cdot \Delta + e \cdot 2^\vartheta \in \mathbb{Z}_q$.

Thank to the ϑ LSB set to 0 in the plaintext, one can evaluate 2^ϑ functions at the cost of only one GenPBS (Algorithm 17) without increasing the noise compared to a regular TFHE PBS. The procedure is described in Algorithm 18.

The shape of the LUT polynomial is set accordingly to the ϑ parameter so that it contains up to 2^ϑ functions. As for the TFHE bootstrap, one needs to have redundancy in the LUT to remove the input noise as explained in Definition 12. Each block of functions (i.e., the sequence of $f_i, i \in [0, 2^\vartheta - 1]$ coefficients) is repeated all along the polynomial. The LUT can be built as follows:

Algorithm 18: $\mathsf{ct}_1, \ldots, \mathsf{ct}_{2^\vartheta} \leftarrow \mathsf{PBSmanyLUT}(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, P_{(f_1, \ldots f_{2^\vartheta})}, \Delta_{\mathsf{out}}, \varkappa, \vartheta)$

Context:
$$\begin{cases} \vec{s} = (s_0, \ldots, s_{n-1}) \in \mathbb{Z}_q^n \\ \vec{s}' = (s'_0, \ldots, s'_{kN-1}) \in \mathbb{Z}_q^{kN} \\ \vec{S}' = \left(S'^{(0)}, \ldots, S'^{(k-1)}\right) \in \mathfrak{R}_q^k \\ \forall 0 \le i \le k-1,\ S'^{(i)} = \sum_{j=0}^{N-1} s'_{i \cdot N + j} X^j \in \mathfrak{R}_q \\ f_0, \ldots, f_{2^\vartheta - 1} : \mathbb{Z} \to \mathbb{Z} \\ (\beta, m') = \mathsf{PTModSwitch}_q(m, \Delta, \varkappa, \vartheta) \in \{0,1\} \times \mathbb{N} \\ \mathsf{CT}_{(f_0, \ldots, f_{2^\vartheta} - 1)} \in \mathsf{GLWE}_{\vec{S}'}\left(P_{(f_0, \ldots, f_{2^\vartheta - 1})} \cdot \Delta_{\mathsf{out}}\right) \text{ (might be a trivial encryption)} \end{cases}$$

Input:
$$\begin{cases} \mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}}(m \cdot \Delta_{\mathsf{in}}) = (a_0, \cdots, a_{n-1}, a_n = b) \in \mathbb{Z}_q^{n+1} \\ \mathsf{BSK} = \left\{\mathsf{BSK}_i \in \mathsf{GGSW}_{\vec{S}'}^{(\mathfrak{B}, \ell)}(s_i)\right\}_{0 \le i \le n-1} \\ P_{(f_0, \ldots, f_{2^\vartheta - 1})} : \text{a redundant LUT for} : x \mapsto f_0(x) || \ldots || f_{2^\vartheta - 1}(x) \\ (\varkappa, \vartheta) \in \mathbb{Z} \times \mathbb{N} : \text{defined along with N the window size} \end{cases}$$

Output: $\mathsf{ct}_0, \ldots, \mathsf{ct}_{2^\vartheta - 1}$ such that $\mathsf{ct}_j \in \mathsf{LWE}_{\vec{s}'}\left((-1)^\beta \cdot f_j(m') \cdot \Delta_{\mathsf{out}}\right)$

```
1 begin
      /* modulus switching                                                */
2     for $0 \le i \le n$ do
3         $a'_i \leftarrow \left[\left\lfloor \frac{a_i \cdot 2N \cdot 2^{\varkappa - \vartheta}}{q} \right\rceil \cdot 2^\vartheta\right]_{2N}$
4     end
      /* blind rotate of the LUT (Algorithm 9)                            */
5     $\mathsf{CT} \leftarrow$ BlindRotate $\left(\mathsf{CT}_{(f_0, \cdots, f_{2^\vartheta - 1})}, \{a'_i\}_{0 \le i \le n}, \mathsf{BSK}\right)$ ;
      /* sample extract the first $2^\vartheta$ terms (coeffs. from 0 to $2^\vartheta - 1$) */
6     for $0 \le j \le 2^\vartheta - 1$ do
7         $\mathsf{ct}_j \leftarrow \mathsf{SampleExtract}(\mathsf{CT}, j)$
8     end
9 end
```

$$P_{(f_0,\ldots,f_{2^\vartheta-1})} = X^{\frac{N}{2p}} \sum_{j=0}^{p-1} X^{j\frac{N}{p}} \sum_{k=0}^{\frac{N}{p2^\vartheta}-1} X^{k\cdot 2^\vartheta} \sum_{i=0}^{2^\vartheta-1} f_i(j)X^i,$$

with $p = \frac{q}{\Delta_{\mathsf{in}}\cdot 2^{\varkappa+1}}$, a power-of-two. By doing so, one can sample-extract at the end 2^ϑ coefficients which leads to 2^ϑ output ciphertexts, one for each evaluated function. By neglecting the computational cost of the ϑ sample extracts, the cost is the same than for a PBS evaluating only one function. The noise is also not impacted.

This method is particularly efficient when the polynomial size is constrained by the desired output noise. If the polynomial size is chosen large enough, there will be bits set to zero between the modulus switching noise and the message. With the Many-LUT PBS, we can exploit these bits to compute different functions on the same input ciphertext.

Theorem 23 (Many-LUT PBS) *Let $\vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n$ be a binary LWE secret key. Let $\vec{S}' = \left(S'_0, \ldots, S'_{k-1}\right) \in \mathfrak{R}_q^k$ be a GLWE secret key such that $S'_i = \sum_{j=0}^{N-1} s'_{i\cdot N+j}X^j \in \mathfrak{R}_q$, and $\vec{s}' = (s'_0, \cdots, s'_{kN-1}) \in \mathbb{Z}_q^{kN}$ be the corresponding LWE key. Let $P_{(f_0,\cdots f_{2^\vartheta-1})} \in \mathfrak{R}_q$ be an r-redundant LUT for the functions $x \mapsto f_0(x)||\ldots||f_{2^\vartheta-1}(x)$ and $\Delta_{\mathsf{out}} \in \mathbb{Z}_q$ be the output scaling factor. Let $(\varkappa, \vartheta) \in \mathbb{Z} \times \mathbb{N}$ be the two integer variables defining (along with N) the window size to be modulus switched, such that $\frac{q2^\vartheta}{\Delta_{\mathsf{in}}2^\varkappa} < 2N$, and let $(\beta, m') = \mathsf{PTModSwitch}_q(m, \Delta_{\mathsf{in}}, \varkappa, \vartheta)$.*

Then, Algorithm 18 takes as input an LWE ciphertext $\mathsf{ct}_{\mathsf{in}} = (a_0, \cdots, a_{n-1}, a_n = b) \in \mathsf{LWE}_{\vec{s}}(m\cdot\Delta_{\mathsf{in}})$, with noise distribution $\chi_{\sigma_{\mathsf{in}}}$, a bootstrapping key $\mathsf{BSK} = \left\{\overline{\overline{\mathsf{CT}}}_i \in \mathsf{GGSW}_{\vec{S}'}^{\mathfrak{B},\ell}(s_i)\right\}_{i=0}^{n-1}$ from $\vec{s}$ to $\vec{S}'$ and a (trivial) GLWE encryption of $P_f \cdot \Delta_{\mathsf{out}}$, and returns 2^ϑ LWE ciphertexts $\{\mathsf{ct}_j\}_{j\in[0,2^\vartheta-1]}$ under the secret key $\vec{s}'$ encrypting the messages $(-1)^\beta \cdot f_j(m') \cdot \Delta_{\mathsf{out}}$ with probability $1 - p_{\mathsf{fail}}$ if and only if the input noise has variance verifying Theorem 22.

The cost of the algorithm is:

$$\mathbb{C}_{\mathsf{PBSmanyLUT}}^{(n,\ell,k,N,\vartheta)} = \mathsf{Cost}\left(GenPBS\right)^{(n,\ell,k,N)} + 2^\vartheta \mathsf{Cost}\left(SampleExtract\right)^{(N)} \tag{5.6}$$

Proof 23 (Theorem 23) *The proof is mainly the same as the one from the* GenPBS *(provided in Appendix A.2). Let $p = \frac{q}{\Delta_{\mathsf{in}}\cdot 2^{\varkappa+1}}$ be the number of possible values for each $f_i, i \in [0, 2^\vartheta - 1]$. Let $m \in [0, p-1]$ be a plaintext value. The polynomial $P_{(f_0,\cdots,f_{2^\vartheta-1})}$*

encodes the following LUT*:*

$$\underbrace{\left(..., \underbrace{f_0(m), ..., f_{2^\vartheta-1}(m), ..., f_0(m), ..., f_{2^\vartheta-1}(m)}_{N/p \text{ elements}}, \underbrace{f_0(m+1), ..., f_{2^\vartheta-1}(m+1), ..., f_0(m+1), ..., f_{2^\vartheta-1}(m+1)}_{N/p \text{ elements}}, ... \right)}_{p \text{ blocks}}$$

From the GenPBS*, all ϑ bits are set to* 0*. Then, by construction of the* LUT*, for $i \in [0, 2^\vartheta - 1]$,* $\mathsf{LUT}_{(f_0,\cdots,f_{2^\vartheta-1})}[m^* + i] = f_i(m')$*.*

□

Remark 29 (Comparison with [CIM19]) *A technique to evaluate many LUTs at the same time by performing a single TFHE bootstrapping (plus a bunch of polynomial multiplications per LUT) has been already proposed in [CIM19] (recalled in Algorithm 14) and used in [GBA21] (Algorithm 15). Their technique does not impose a strong constraint on the polynomial size used for the bootstrapping, however it results in a larger output noise, which strictly depends on the function being evaluated. If the noise constraints at the output of the bootstrapping are a problem, the technique of [CIM19] will require to increase the polynomial size.*

Our new PBSmanyLUT *is a better alternative to this technique in some situations because the output noise will be independent from the evaluated function. But this comes at the cost of enlarging the space for the evaluation of the different LUTs (i.e., ϑ bits on the modulus switching to evaluate 2^ϑ functions so a large enough polynomial size N must be chosen). If we already are working with large polynomials, there is no computation overhead nor extra noise terms when replacing a* GenPBS *by a* PBSmanyLUT*.*

Improving the Circuit Bootstrapping In TFHE [Chi+17], a technique called *circuit bootstrapping* was introduced, we recalled it in Algorithm 11. With this technique, we can convert an LWE ciphertext (Definition 4) into an GGSW ciphertext (Definition 9).

The authors of [Chi+17] observe that a GGSW ciphertext, encrypting a message $\mu \in \mathbb{Z}$ (μ is binary in their application) under the secret key $\vec{S} = (S_0, \ldots, S_{k-1}, S_k = -1)$, is composed by $(k+1)\ell$ GLWE ciphertexts encrypting $\mu \cdot S_i \cdot \frac{q}{\mathfrak{B}^j}$, for $0 \leq i \leq k$ and $1 \leq j \leq \ell$. As already mentioned in Section 2.3, the goal of circuit bootstrapping (Algorithm 11) is to build one by one all the GLWE ciphertexts composing the output GGSW. In order to do that, it performs the following two steps:

- The first step performs ℓ *independent TFHE PBS* to transform the input LWE encryption of μ into independent LWE encryptions of $\mu \cdot \frac{q}{\mathfrak{B}^j}$.

- The second step performs a list of $(k+1)\ell$ private key switchings from LWE to GLWE to multiply the messages $\mu \cdot \frac{q}{\mathfrak{B}^j}$ obtained in the first step by the elements of the secret key S_i, and so to obtain the different lines of the output GGSW.

Here, we propose a faster method based on the PBSmanyLUT algorithm (Algorithm 18). In a nutshell, the idea is to replace the ℓ PBSs of the first step by only one PBSmanyLUT (that costs exactly the same as a one of the ℓ original PBSs and do not increase the noise). Since the most costly part of the circuit boostrapping is due to the PBS part, the overall cost is then roughly reduced by a factor ℓ. In [Chi+17], $\ell = 2$, so we have an improvement of a factor 2 on the PBS part, without any impact on the noise.

Theorem 24 *Consider the circuit boostrapping algorithm as described in [Chi+17, Alg. 11] and recalled in Algorithm 11. The ℓ independent bootstrappings (line 2) can be replaced by:*

$$\begin{cases} \{\mathsf{ct}_i\}_{i\in[1,\ell]} \leftarrow \mathsf{PBSmanyLUT}(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, P \cdot X^{N/2^{\rho+1}}, 1, \varkappa = 0, \rho = \lceil \log_2(\ell) \rceil) \\ \forall i \in [1,\ell], \mathsf{ct}_i + \left(\vec{0}, \frac{q}{2\mathfrak{B}^i}\right) \end{cases}$$

$$\textit{with } P(X) = \sum_{i=0}^{\frac{N}{2^\rho}-1} \sum_{j=0}^{2^\rho-1} \frac{q}{2\mathfrak{B}^j} X^{2^\rho \cdot i + j}.$$

Proof 24 (Theorem 24) *By calling* PBSmanyLUT *with $\rho = \lceil \log_2(\ell) \rceil$, we are able to compute ℓ* PBS *in parallel. The polynomial P represents the* LUT*:*

$$\underbrace{\left(\underbrace{\frac{q}{2\mathfrak{B}^1}, \dots, \frac{q}{2\mathfrak{B}^\ell}, 0, \dots, 0}_{2^\rho \text{ elements}}, \underbrace{\frac{q}{2\mathfrak{B}^1}, \dots, \frac{q}{2\mathfrak{B}^\ell}, 0, \dots, 0}_{2^\rho \text{ elements}}, \dots, \underbrace{\frac{q}{2\mathfrak{B}^1}, \dots, \frac{q}{2\mathfrak{B}^\ell}, 0, \dots, 0}_{2^\rho \text{ elements}} \right)}_{N' = N/2^\rho \text{ elements}}$$

In the end, for $i \in [1,\ell]$, $\mathsf{ct}_i \in \mathsf{LWE}_{\vec{S}}(\pm\frac{q}{2\mathfrak{B}^i})$, where the sign depends on the plaintext value. By adding the trivial ciphertext $\left(\vec{0}, \frac{q}{2\mathfrak{B}^i}\right)$ to the ct_i, we either get $\mathsf{ct}_i \in \mathsf{LWE}_{\vec{S}}(\frac{q}{\mathfrak{B}^i})$ or $\mathsf{LWE}_{\vec{S}}(0)$, as expected.

$\square$

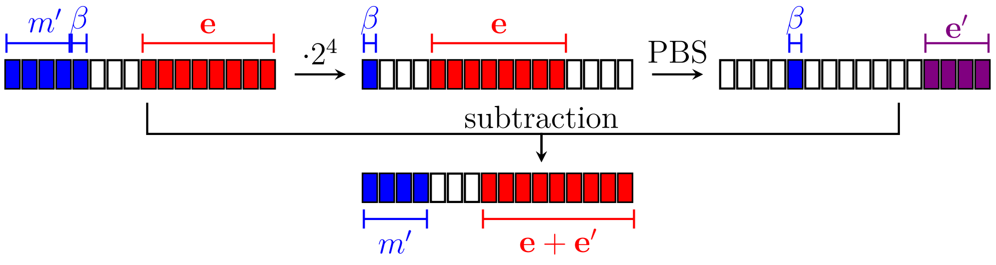


Figure 5.3: Overview of the rounded PBS (Algorithm 19).

5.3 Rounded PBS

As explained in Limitation 2 and illustrated on Figure 4.3, the PBS (Algorithm 10) does not scale well with large precisions.

Some lookup tables are independent from the least significant bits of the message. For example, to compute the sign of a message, we do not need the LSB of the message. In some use-cases, we can settle for a lookup table evaluation on a rounding of the input message, for instance, when evaluating a neural network in FHE, we do not care about exact computation. Unfortunately, with the PBS, there is a dependency between the input message precision and the polynomial size N.

In this section, we describe a way to round a message before a lookup table evaluation. By rounding the message before the lookup table evaluation, we are sure that the PBS will depend on the precision after rounding and not on the input message precision.

We introduce our new technique in Algorithm 19. In this algorithm, KS-PBS denotes a key switch followed by a PBS. It can be broken down into the following steps illustrated in Figure 5.3. First, we extract the least significant bits of the message that we want to discard. To do so, we start with the least significant bit of the message. We multiply the ciphertext by a power of 2 which puts the LSB of the input message in the MSB position. Then, we apply a PBS with the right lookup table to shift the bit back to the least significant bits. Note that, we do not need to have a bit of padding to apply the right-shift lookup table on a one-bit message. Next, we subtract this ciphertext with the original ciphertexts, which will remove the LSB of the original message. We repeat this process until we have removed all the undesired least significant bits. Finally, we apply a PBS on the rounded ciphertext to evaluate the lookup table.

Theorem 25 (Rounded-PBS) *Let $\vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n$ be a binary LWE secret key. Let $\vec{S'} = \left(S'_0, \ldots, S'_{k-1}\right) \in \mathfrak{R}_q^k$ be a GLWE binary secret key such that $S'_i = \sum_{j=0}^{N-1} s'_{i \cdot N+j} \cdot X^j$,*

Algorithm 19: $\mathsf{ct_{out}} \leftarrow \mathsf{Round\text{-}PBS}(\mathsf{ct_{in}}, \mathsf{PUB}, L)$

```
Context: { Δ_in : scaling factor for the input ciphertext ct_in
         { Δ_out : scaling factor for the input ciphertext ct_out
         { δ : bits to remove from the message in ct_in
         {      starting from Δ_in with Δ_out = 2^δ · Δ_in
         { (ϰ, ϑ) ∈ ℕ × ℕ parameters of the modulus switching in the
         {     generalized PBS (Algorithm 17)
Input:   { ct_in encrypting m_in
         { PUB : public keys required for the whole algorithm
         { L ∈ ℤ_ω : a LUT
Output: ct_out encrypting L[⌊m_in / 2^δ⌉]
1  $\mathsf{ct}'_{\mathsf{in}} \leftarrow \mathsf{ct_{in}} + \left(0, \cdots, 0, \frac{\Delta_{\mathsf{out}}}{2}\right)$
2  for $j \in [\![0; \delta - 1]\!]$ do
       /* Extract the LSB of the message with generalized PBS (Algorithm 17)      */
3      $\epsilon = \frac{q}{4}$
4      $\alpha_j = \frac{\Delta_{\mathsf{in}} \cdot 2^j}{2}$
5      $L_j = [-\alpha_j, \cdots, -\alpha_j]$
6      $c_j \leftarrow \mathsf{KS\text{-}PBS}\left(\left(\mathsf{ct}'_{\mathsf{in}} \cdot 2^{\delta-1-j}\right) + (0, \cdots, 0, \epsilon), \mathsf{PUB}, L_j, \left(\varkappa = \log_2(\Delta_{\mathsf{in}}) + j, \vartheta = 0\right)\right)$
7      $c'_j \leftarrow c_i + (0, \cdots, 0, \alpha_{i,j})$
       /* Subtract the extracted bit from the original ciphertext                 */
8      $\mathsf{ct}'_{\mathsf{in}} \leftarrow \mathsf{Sub}(\mathsf{ct}'_{\mathsf{in}}, c'_j)$
   /* Apply the LUT on the rounded message (Algorithm 17)                         */
9  $\mathsf{ct_{out}} \leftarrow \mathsf{KS\text{-}PBS}(\mathsf{ct}'_{\mathsf{in}}, \mathsf{PUB}, L, \varkappa = 0, \vartheta = 0)$
10 return $\mathsf{ct_{out}}$
```

and $\vec{s'} = (s'_0, \cdots, s'_{kN-1})$ be the corresponding binary LWE secret key. Let L_f be an r-redundant LUT (Definition 12) for a function $f : \mathbb{Z} \to \mathbb{Z}$, Δ_{in} be the input scaling factor and Δ_{out} be the output scaling factor. Let $(\varkappa, \vartheta)$ be the two integer variables defining (along with N) the window size to be modulus switched, such that $\frac{q2^{\vartheta}}{\Delta_{\mathsf{in}}2^{\varkappa}} < 2N$, and let $(\beta, m') = \mathsf{PTModSwitch}_q(m, \Delta_{\mathsf{in}}, \varkappa, \vartheta) \in \{0,1\} \times \mathbb{N}$. Let δ, the number of bits to remove in the input message prior to the LUT evaluation, so that we have $\Delta_{\mathsf{out}} = 2^{\delta} \cdot \Delta_{\mathsf{in}}$.

Then Algorithm 19 takes as input an LWE ciphertext $\mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s'}}(m \cdot \Delta_{\mathsf{in}}) \in \mathbb{Z}_q^{kN+1}$ with noise distribution $\chi_{\sigma_{\mathsf{in}}}$, a bootstrapping key $\mathsf{BSK}{=}\left\{\overline{\overline{\mathsf{CT}}}_i{\in}\mathsf{GGSW}^{\mathfrak{B},\ell}_{\vec{S'}}(s_i)\right\}_{i=0}^{n-1}$ from $\vec{s}$ to $\vec{S'}$, a keyswitching key $\mathsf{KSK}{=}\left\{\overline{\mathsf{CT}}_i{\in}\mathsf{GLev}^{\mathfrak{B},\ell}_{\vec{s}}\left(s'_i\right){\in}\mathfrak{R}_q^{\ell\times(n+1)}\right\}_{0\le i\le kN-1}$ from $\vec{s'}$ to $\vec{s}$ with noise sampled from $\chi_{\sigma_{\mathsf{KSK}}}$ and a (possibly trivial) GLWE encryption of $P_f \cdot \Delta_{\mathsf{out}}$, and returns an LWE ciphertext $\mathsf{ct}_{\mathsf{out}}$ under the secret key $\vec{s'}$, encrypting the message $f\left(\left\lfloor \frac{m}{2^{\delta}} \right\rceil\right) \cdot \Delta_{\mathsf{out}}$ with a probability $1 - p_{\mathsf{fail}}$ if and only if the input noise has a variance verifying

$$\begin{cases} \sigma_{\mathsf{in}}^2 & < \left(\frac{\Delta_{\mathsf{in}}^2}{4\Gamma^2} - \frac{q'^2}{12w^2} + \frac{1}{12} - \frac{nq'^2}{24w^2} - \frac{n}{48}\right) \cdot 2^{2-2\delta} \\ \sigma_{\mathsf{in}}^2 & < \frac{\Delta_{\mathsf{out}}^2}{4\Gamma^2} - \frac{q^2}{12w^2} + \frac{1}{12} - \frac{nq^2}{24w^2} - \frac{n}{48} - \delta \cdot \sigma_{\mathsf{PBS}}^2 \end{cases} \tag{5.7}$$

where Γ is the standard score (Definition 22) associated with the failure probability p_{fail}, $w = 2N \cdot 2^{-\vartheta}$, $q' = q \cdot 2^{-\varkappa}$ and $\sigma_{\mathsf{PBS}}^2 = \mathsf{Var}\left(\mathsf{PBS}\right)$.

The output noise after the rounded PBS is

$$\mathsf{Var}\left(\mathsf{Rounded\text{-}PBS}\right) = \mathsf{Var}\left(\mathsf{PBS}\right) \ . \tag{5.8}$$

The cost of Algorithm 19 is

$$\mathsf{Cost}\left(\mathsf{Rounded\text{-}PBS}\right) = (\delta + 1) \cdot \left(\mathsf{Cost}\left(GenPBS\right) + \mathsf{Cost}\left(KS\right)\right). \tag{5.9}$$

Proof 25 (Theorem 25) *In line 1, we use a trick to replace a round by a floor.*

Let us prove that $\left\lfloor \frac{m_{\mathsf{in}}}{2^{\delta}} \right\rceil = \left\lfloor \frac{m_{\mathsf{in}} + 2^{\delta-1}}{2^{\delta}} \right\rfloor$.

Define $\bar{m}$ and $\underline{m}$ such that $m_{\mathsf{in}} = \bar{m} \cdot 2^{\delta} + \underline{m}$ with $0 \le \underline{m} < 2^{\delta}$.

We have

$$\left\lfloor \frac{m_{\mathsf{in}}}{2^{\delta}} \right\rceil = \begin{cases} \bar{m} & \text{if } \frac{\underline{m}}{2^{\delta}} < 0.5 \\ \bar{m} + 1 & \text{otherwise} \end{cases}$$

We also have

$$\left\lfloor \frac{m_{\mathsf{in}} + 2^{\delta-1}}{2^\delta} \right\rceil = \bar{m} + \left\lfloor \frac{m}{2^\delta} + \frac{1}{2} \right\rfloor$$
$$= \begin{cases} \bar{m} \text{ if } \frac{m}{2^\delta} + \frac{1}{2} < 1 \text{ i.e., } \frac{m}{2^\delta} < 0.5 \\ \bar{m} + 1 \text{ otherwise} \end{cases}$$

The rest of the proof is trivial using Proof 30 and Appendix A.2.

□

Remark 30 (Non power-of-two message modulus) *In Algorithm 19, we give the algorithm for a power-of-two message modulus. It could easily be tweaked to work with a non power-of-two message modulus similarly to Algorithm 30.*

Remark 31 (Faster Rounded-PBS) *Algorithm 19 introduces an efficient way to compute a LUT on a rounded message. If we look carefully at the constraints in Equation* (5.7)*, we notice that the second constraint is almost always dominated by the first constraint. If we want to have a faster algorithm, we could use different parameters for the PBS (Algorithm 17) in the bit extraction part of the algorithm and for the final PBS to evaluate the lookup table. The input ciphertext of this new variant of Algorithm 19 is assumed to be an output of the final PBS. We first keyswitch it to encrypt it with the same secret key as the one at the output of the PBSs in the bit extraction part. We then apply the same algorithm. The keyswitch could be replaced by the FFT keyswitch introduced in [Che+21].*

5.4 LWE Multiplication

We first recall the multiplication algorithm for GLWE ciphertexts in Algorithm 20. It is composed of a *tensor product* followed by a *relinearization* and is widely used in the literature [FV12; Che+17] (we recall the GLWE [BGV12] algorithm, instead of the more limited RLWE version). We thoroughly study its noise growth and provide a formal noise analysis where $\mathsf{Var}(S)$ is the variance of a GLWE secret key polynomial $S \in \mathfrak{R}_q$, $\mathsf{Var}(S'_{\mathsf{even}})$ (resp. $\mathsf{Var}(S'_{\mathsf{odd}})$) is the variance of even (resp. odd) coefficients in S^2 and $\mathsf{Var}(S'')$ is the variance of coefficients in $S_i \cdot S_j$ which is the product between two independent secret key polynomials $S_i, S_j \in \mathfrak{R}_q$. We provide concrete cryptographic parameters depending on the precision (number of bits of the message) and the multiplicative depth in Table 5.1. Those parameters are obtained using the optimization framework described in Chapter 4.

Algorithm 20: $\mathsf{CT} \leftarrow \mathsf{GLWEMult}(\mathsf{CT}_1, \mathsf{CT}_2, \mathsf{RLK})$

```
Context: { S⃗ = (S_0, ..., S_{k-1}) ∈ R_q^k : a GLWE secret key
         { Δ = min(Δ_1, Δ_2) ∈ Z_q
         { PT_1 = M_1 Δ_1 ∈ R_q
         { PT_2 = M_2 Δ_2 ∈ R_q
Input:   { $\mathsf{CT}_1 \in \mathsf{GLWE}_{\vec{S}}(\mathsf{PT}_1) = \left(A_{1,0}, \cdots, A_{1,k-1}, B_1\right) \subseteq \mathfrak{R}_q^{k+1}$
         { $\mathsf{CT}_2 \in \mathsf{GLWE}_{\vec{S}}(\mathsf{PT}_2) = \left(A_{2,0}, \cdots, A_{2,k-1}, B_2\right) \subseteq \mathfrak{R}_q^{k+1}$
         { $\mathsf{RLK} = \left\{\overline{\mathsf{CT}}_{i,j} \in \mathsf{GLev}_{\vec{S}}^{(\mathfrak{B},\ell)}(S_i \cdot S_j)\right\}_{0 \le i \le k-1}^{0 \le j \le i}$ : a relinearization key for $\vec{S}$
Output: $\mathsf{CT} \in \mathsf{GLWE}_{\vec{S}}\left(\frac{\mathsf{PT}_1 \cdot \mathsf{PT}_2}{\Delta}\right) \subseteq \mathfrak{R}_q^{k+1}$
1  begin
       /* Tensor product                                                  */
2      for $0 \le i \le k-1$ do
3          $T'_i \leftarrow \left[\left\lfloor \frac{[A_{1,i} \cdot A_{2,i}]_Q}{\Delta} \right\rceil\right]_q$
4      end
5      for $0 \le i \le k-1,\ 0 \le j < i$ do
6          $R'_{i,j} \leftarrow \left[\left\lfloor \frac{[A_{1,i} \cdot A_{2,j} + A_{1,j} \cdot A_{2,i}]_Q}{\Delta} \right\rceil\right]_q$
7      end
8      for $0 \le i \le k-1$ do
9          $A'_i \leftarrow \left[\left\lfloor \frac{[A_{1,i} \cdot B_2 + B_1 \cdot A_{2,i}]_Q}{\Delta} \right\rceil\right]_q$
10     end
11     $B' \leftarrow \left[\left\lfloor \frac{[B_1 \cdot B_2]_Q}{\Delta} \right\rceil\right]_q$
       /* Relinearization                                                 */
12     $\mathsf{CT} \leftarrow \left(A'_0, \cdots, A'_{k-1}, B'\right) + \sum_{i=0}^{k-1} \left\langle \overline{\mathsf{CT}}_{i,i}, \mathsf{dec}^{(\mathfrak{B},\ell)}\left(T'_i\right)\right\rangle + \sum_{0 \le i \le k-1}^{0 \le j < i} \left\langle \overline{\mathsf{CT}}_{i,j} \cdot \mathsf{dec}^{(\mathfrak{B},\ell)}\left(R'_{i,j}\right)\right\rangle$
13 end
```

Precision	1	2	3	4	5	6	7	8	9	10	11	12
Max. depth	32	16	16	8	8	8	8	4	4	4	4	4
$\log_2(N)$	12	11	12	11	11	12	12	11	11	11	12	12
$\log_2(\mathfrak{B})$	8	5	8	12	10	8	8	20	17	15	17	17
ℓ	8	10	8	4	5	8	8	2	3	3	3	3

Precision	13	14	15	16	17	18	19	20	21	22	23	24
Max. depth	4	2	2	2	2	2	2	2	2	2	2	2
$\log_2(N)$	12	11	11	11	11	11	11	11	12	12	12	12
$\log_2(\mathfrak{B})$	8	30	30	20	20	20	20	20	20	20	20	20
ℓ	8	1	1	2	2	2	2	2	2	2	2	2

Table 5.1: Parameters depending on the GLWE multiplicative depth and the precision. This table was generated in May 2021 with $\alpha = 0.05287332817861731$ and $\beta = 4.551576767993042$ (as defined in Section 3.1).

Theorem 26 (GLWE multiplication) *Let* $\mathsf{CT}_1 \in \mathsf{GLWE}_{\vec{S}}(\mathsf{PT}_1) \in \mathfrak{R}_q^{k+1}$ *and* $\mathsf{CT}_2 \in \mathsf{GLWE}_{\vec{S}}(\mathsf{PT}_2) \in \mathfrak{R}_q^{k+1}$ *be two GLWE ciphertexts, encrypting respectively* $\mathsf{PT}_1 = M_1\Delta_1 \in \mathfrak{R}_q$ *and* $\mathsf{PT}_2 = M_2\Delta_2 \in \mathfrak{R}_q$*, under the same secret key* $\vec{S} = (S_0, \ldots, S_{k-1}) \in \mathfrak{R}_q^k$*, with noises sampled respectively from* χ_{σ_1} *and* χ_{σ_2}*. Let* $\mathsf{RLK}{=}\left\{\overline{\mathsf{CT}}_{i,j}{\in}\mathsf{GLev}_{\vec{S}}^{(\mathfrak{B},\ell)}(S_i{\cdot}S_j){\in}\mathfrak{R}_q^{\ell\times(k+1)}\right\}_{0\leq i\leq k-1}^{0\leq j\leq i}$ *be a relinearization key for the GLWE secret key* $\vec{S}$*, with noise sampled from* $\chi_{\sigma_{\mathsf{RLK}}}$*.*

Algorithm 20 computes a new GLWE ciphertext CT *encrypting the product* $\mathsf{PT}_1 \cdot \mathsf{PT}_2/\Delta \in \mathfrak{R}_q$ *where* $\Delta = \min(\Delta_1, \Delta_2)$ *(a scaling factor), under the secret key* $\vec{S}$*, with a noise variance* $\mathsf{Var}_{\mathsf{GLWEMult}}$ *estimated by the following noise formula (Definition 19):*

$$\begin{aligned}
\mathsf{Var}_{\mathsf{GLWEMult}} &= \frac{N}{\Delta^2}\left(\Delta_1^2||M_1||_\infty^2\sigma_2^2 + \Delta_2^2||M_2||_\infty^2\sigma_1^2 + \sigma_1^2\sigma_2^2\right) \\
&+\frac{N}{\Delta^2}\left(\frac{q^2-1}{12}\left(1 + kN\mathsf{Var}(S) + kN\mathbb{E}^2(S)\right) + \frac{kN}{4}\mathsf{Var}(S) + \frac{1}{4}\left(1 + kN\mathbb{E}(S)\right)^2\right)(\sigma_1^2 + \sigma_2^2) \\
&+\frac{1}{12} + \frac{kN}{12\Delta^2}\cdot\left((\Delta^2-1)\cdot\left(\mathsf{Var}(S) + \mathbb{E}^2(S)\right) + 3\cdot\mathsf{Var}(S)\right) \\
&+\frac{k(k-1)N}{24\Delta^2}\cdot\left((\Delta^2-1)\cdot\left(\mathsf{Var}(S'') + \mathbb{E}^2(S'')\right) + 3\cdot\mathsf{Var}(S'')\right) \\
&+\frac{kN}{24\Delta^2}\cdot\left((\Delta^2-1)\cdot\left(\mathsf{Var}(S'_{\mathsf{odd}}) + \mathsf{Var}(S'_{\mathsf{even}}) + 2\cdot\mathbb{E}^2(S'_{\mathsf{mean}})\right) + 3\cdot(\mathsf{Var}(S'_{\mathsf{odd}}) + \mathsf{Var}(S'_{\mathsf{even}}))\right) \\
&+k\ell N\sigma_{\mathsf{RLK}}^2\cdot\frac{(k+1)}{2}\cdot\frac{\mathfrak{B}^2+2}{12} + \frac{kN}{8}\cdot((k-1)\cdot\mathsf{Var}(S'') + \mathsf{Var}(S'_{\mathsf{odd}}) + \mathsf{Var}(S'_{\mathsf{even}})) \\
&+\frac{kN}{2}\left(\frac{q^2}{12\mathfrak{B}^{2\ell}} - \frac{1}{12}\right)\left((k-1)\cdot(\mathsf{Var}(S'') + \mathbb{E}^2(S''_{\mathsf{mean}})) + \mathsf{Var}(S'_{\mathsf{odd}}) + \mathsf{Var}(S'_{\mathsf{even}}) + 2\mathbb{E}^2(S'_{\mathsf{mean}})\right)
\end{aligned} \tag{5.10}$$

Let $k^* = \dfrac{k(k+1)}{2}$ *and* $k^+ = \dfrac{(k+1)(k+2)}{2}$.

The cost (Definition 23) of the algorithm is

$$\mathsf{Cost}\,(GLWEMult)^{(k,\ell,n,N)} = \mathsf{Cost}\,(TensorProduct)^{(k,N)} + \mathsf{Cost}\,(Relin)^{(k,\ell,N)} \tag{5.11}$$

where

$$\begin{aligned}
\mathsf{Cost}\,(TensorProduct)^{(k,N)} &= 2(k+1)\mathsf{Cost}\,(FFT) + k^+\mathsf{Cost}\,(iFFT) \\
&\quad + (k+1)^2N\mathsf{Cost}\,(multFFT) + k^*N\mathsf{Cost}\,(addFFT)\,,
\end{aligned} \tag{5.12}$$

$$\begin{aligned}\mathsf{Cost}\left(Relin\right)^{(k,\ell,N)} &= N\ell k^{*}\mathsf{Cost}\left(dec\right) + k^{*}\ell\mathsf{Cost}\left(FFT\right) + k^{*}\ell(k+1)N\mathsf{Cost}\left(multFFT\right)\\ &\quad + (k^{*}\ell-1)(k+1)N\mathsf{Cost}\left(addFFT\right) + (k+1)\mathsf{Cost}\left(iFFT\right).\end{aligned} \tag{5.13}$$

Proof 26 (Theorem 26) *The detailed computation leading us to the aforementioned noise formula is provided in Appendix A.1. In the proof, we compute the decryption of the resulting ciphertext, obtaining the message plus the noise in order to estimate its variance.*

□

Algorithm 20 can be adapted in order to perform a GLWE square: the square is more efficient since $R'_{i,j}$ and A'_i are computed with a single multiplication instead of two. For more details, we refer to Algorithm 21.

Algorithm 21: $\mathsf{CT} \leftarrow \mathsf{GLWESquare}(\mathsf{CT}_{\mathsf{in}}, \mathsf{RLK})$

```
Context: $\begin{cases}\vec{S} = (S_0,\dots,S_{k-1}) : \text{ a GLWE secret key}\\ \Delta = \min\left(\{\Delta_i\}\right)\\ \mathsf{PT} = \sum_{i=0}^{N-1} m_i\Delta_i X^i\end{cases}$
Input: $\begin{cases}\mathsf{CT}_{\mathsf{in}} \in \mathsf{GLWE}_{\vec{S}}\left(\mathsf{PT}\right) = (A_0,\cdots,A_{k-1},B)\\ \mathsf{RLK} = \left\{\overline{\mathsf{CT}}_{i,j} \in \mathsf{GLev}_{\vec{S}}^{(\mathfrak{B},\ell)}\left(S^{(i)}\cdot S^{(j)}\right)\right\}_{0\le i\le k-1}^{0\le j\le i} : \text{ a relinearization key for } \vec{S}\end{cases}$
Output: $\mathsf{CT} \in \mathsf{GLWE}_{\vec{S}}\left(\frac{\mathsf{PT}^2}{\Delta}\right)$
1  begin
       /* Tensor product                                                */
2      for $0 \le i \le k-1$ do
3          $T'_i \leftarrow \left[\left\lfloor \frac{[A_i^2]_Q}{\Delta} \right\rceil\right]_q$
4      end
5      for $0 \le i \le k-1,\ 0 \le j < i$ do
6          $R'_{i,j} \leftarrow \left[\left\lfloor \frac{[2\cdot A_i\cdot A_j]_Q}{\Delta} \right\rceil\right]_q$
7      end
8      for $0 \le i \le k-1$ do
9          $A'_i \leftarrow \left[\left\lfloor \frac{[2\cdot A_i\cdot B]_Q}{\Delta} \right\rceil\right]_q$
10     end
11     $B' \leftarrow \left[\left\lfloor \frac{[B^2]_Q}{\Delta} \right\rceil\right]_q$
       /* Relinearization                                               */
12     $\mathsf{CT} \leftarrow \left(A'_0,\cdots,A'_{k-1},B'\right) + \sum_{i=0}^{k-1}\left\langle \overline{\mathsf{CT}}_{i,i}, \mathsf{dec}^{(\mathfrak{B},\ell)}\left(T'_i\right)\right\rangle + \sum_{0\le i\le k-1}^{0\le j<i}\left\langle \overline{\mathsf{CT}}_{i,j}\cdot \mathsf{dec}^{(\mathfrak{B},\ell)}\left(R'_{i,j}\right)\right\rangle$
13 end
```

Using Algorithm 20 or Algorithm 21, we build an LWE multiplication and several variants where we perform efficiently several products or even a sum of several products.

5.4.1 Single LWE Multiplication

We now define Algorithm 22 to homomorphically *multiply two LWE ciphertexts*, which removes Limitation 4.

It requires the *sample extract* (Algorithm 5 and Theorem 9). This algorithm does not impact the noise in a ciphertext and consists in simply rearranging some of the coefficients of the GLWE input ciphertext to build the output LWE ciphertext encrypting one of the coefficients of the input polynomial plaintext. The sample extract algorithm is described in [Chi+20a, Section 4.2] for RLWE inputs, and can be easily extended to GLWE ones as we did in Algorithm 5.

It also requires a packing key switch described in Section 2.3.2. It takes as input several LWE ciphertexts, key switches them using the LWE Keyswitch (Algorithm 1), rotates them (Theorem 3) and finally adds the keyswitched ciphertexts together (Theorem 1).

Theorem 8 provides some details on this procedure and gives the noise (Definition 19) and cost formulae (Definition 23).

```
Algorithm 22: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{LWEMult}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{RLK}, \mathsf{KSK})$

Context: $\vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n$ : the input LWE secret key
         $\vec{s'} = (s'_0, \cdots, s'_{kN-1}) \in \mathbb{Z}_q^{kN}$ : the output LWE secret key
         $\vec{S'} = \left(S'_0, \dots, S'_{k-1}\right) \in \mathfrak{R}_q^k$ : a GLWE secret key
         $\forall 0 \le i \le k-1,\ S'_i = \sum_{j=0}^{N-1} s'_{i\cdot N+j} X^j \in \mathfrak{R}_q$
         $\Delta_{\mathsf{out}} = \max(\Delta_1, \Delta_2) \in \mathbb{Z}_q$
Input: $\mathsf{ct}_1 \in \mathsf{LWE}_{\vec{s}}(m_1 \cdot \Delta_1) \in \mathbb{Z}_q^{n+1}$
       $\mathsf{ct}_2 \in \mathsf{LWE}_{\vec{s}}(m_2 \cdot \Delta_2) \in \mathbb{Z}_q^{n+1}$
       $\mathsf{RLK}$ : a relinearization key for $\vec{S'}$ as defined in algorithm 20
       $\mathsf{KSK} = \left\{\overline{\mathsf{CT}}_i \in \mathsf{GLev}_{\vec{S'}}^{\mathfrak{B},\ell}(s_i)\right\}_{0\le i\le n-1}$ : a key switching key from $\vec{s}$ to $\vec{S'}$
Output: $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s'}}(m_1 \cdot m_2 \cdot \Delta_{\mathsf{out}}) \in \mathbb{Z}_q^{kN+1}$
1 begin
      /* KS from LWE to GLWE (Section 2.3.2) */
2     $\mathsf{CT}_1 \in \mathsf{GLWE}_{\vec{S'}}(m_1 \cdot \Delta_1) \leftarrow \mathsf{PackingKS}(\{\mathsf{ct}_1\}, \{0\}, \mathsf{KSK})$ ;
3     $\mathsf{CT}_2 \in \mathsf{GLWE}_{\vec{S'}}(m_2 \cdot \Delta_2) \leftarrow \mathsf{PackingKS}(\{\mathsf{ct}_2\}, \{0\}, \mathsf{KSK})$ ;
      /* GLWE multiplication (Algorithm 20) */
4     $\mathsf{CT} \in \mathsf{GLWE}_{\vec{S'}}(m_1 \cdot m_2 \cdot \Delta_{\mathsf{out}}) \leftarrow \mathsf{GLWEMult}(\mathsf{CT}_1, \mathsf{CT}_2, \mathsf{RLK})$
      /* Sample extract the constant term (Algorithm 5) */
5     $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s'}}(m_1 \cdot m_2 \cdot \Delta_{\mathsf{out}}) \leftarrow \mathsf{SampleExtract}(\mathsf{CT}, 0)$
6 end
```

Now, we have everything to build the LWE multiplication described in Algorithm 22 and in Theorem 27.

Theorem 27 (LWE Multiplication) *Let* $\mathsf{ct}^{(1)} \in \mathsf{LWE}_{\vec{s}}(m_1 \cdot \Delta_1)$ *and* $\mathsf{ct}^{(2)} \in \mathsf{LWE}_{\vec{s}}(m_2 \cdot \Delta_2)$ *be two LWE ciphertexts, encrypting respectively* $m_1 \cdot \Delta_1$ *and* $m_2 \cdot \Delta_2$*, both encrypted under the LWE secret key* $\vec{s} = (s_0, \dots, s_{n-1})$*, with noise sampled respectively from* χ_{σ_1} *and* χ_{σ_2}*. Let* $\mathsf{KSK} = \left\{\overline{\mathsf{CT}}_i \in \mathsf{GLev}^{\mathfrak{B},\ell}_{\vec{S'}}(s_i)\right\}_{0\le i\le n-1}$ *a key switching key from* $\vec{s}$ *to* $\vec{S'}$ *where* $\vec{S'} = \left(S'_0, \dots, S'_{k-1}\right)$*, with noise sampled from* $\chi_{\sigma_{\mathsf{KSK}}}$*. Let* RLK *be a relinearization key for* $\vec{S'}$*, defined as in Theorem 26.*

Algorithm 22 computes a new LWE ciphertext $\mathsf{ct}_{\mathsf{out}}$*, encrypting the product* $m_1 \cdot m_2 \cdot \Delta_{\mathsf{out}}$*, where* $\Delta_{\mathsf{out}} = \max(\Delta_1, \Delta_2)$*, under the secret key* $\vec{s'}$*. The variance of the noise in* $\mathsf{ct}_{\mathsf{out}}$ *can be estimated by replacing the variances* σ_1 *and* σ_2 *in the GLWE multiplication (Equation* (5.10)*, Theorem 26) with the variance estimated after a packing key switch (Equation* (2.12)*, Theorem 8). The cost is*

$$\begin{aligned}\mathsf{Cost}\left(LWEMult\right)^{(\ell_{\mathsf{KS}},\ell_{\mathsf{RL}},n,k,N)} &= 2\cdot\mathsf{Cost}\left(PackingKS\right)^{(1,\ell_{\mathsf{KS}},n,k,N)}\\ &+ \mathsf{Cost}\left(GLWEMult\right)^{(k,\ell_{\mathsf{RL}},n,N)}\\ &+ \mathsf{Cost}\left(SampleExtract\right)^{(N)}\ .\end{aligned} \tag{5.14}$$

Proof 27 (Theorem 27) *The cost formula is easy to obtain using Algorithm 22. For the noise formulae, we refer to Appendices A.1, A.3 and A.4.*

□

5.4.2 Variants of the LWE Multiplication

It is possible to use the LWE multiplication introduced in Algorithm 22 to compute with a single multiplication several products, or several squares, or a sum of several products, or even a sum of several squares. These four functionalities can be easily achieved by slightly modifying Algorithm 22.

Packed Products

The $\mathsf{PackedMult}$ algorithm takes as input two sets of LWE ciphertexts $\left\{\mathsf{ct}^{(1)}_i\right\}_{0\le i<\alpha}$ and $\left\{\mathsf{ct}^{(2)}_i\right\}_{0\le i<\alpha}$ such that for $j \in \{1,2\}$ and for $0 \le i < \alpha, \mathsf{ct}^{(j)}_i \in \mathsf{LWE}_{\vec{s}}(m^{(j)}_i \cdot \Delta_j)$.

Its goal is to compute LWE encryptions of the products $m^{(1)}_i \cdot m^{(2)}_i \cdot \Delta_{\mathsf{out}}$, where $\Delta_{\mathsf{out}} = \max(\Delta_1, \Delta_2)$. The algorithm is described in Algorithm 23.

Algorithm 23: $\left\{\mathsf{ct}_i^{(\mathsf{out})}\right\}_{i=0}^{\alpha-1} \leftarrow \mathsf{PackedMult}\left(\left\{\mathsf{ct}_i^{(1)}\right\}_{i=0}^{\alpha-1}, \left\{\mathsf{ct}_i^{(2)}\right\}_{i=0}^{\alpha-1}, \mathsf{RLK}, \mathsf{KSK}\right)$

```
Context: { $\vec{s} = (s_0, \cdots, s_{n-1})$ : the input LWE secret key
         { $\vec{s'} = (s'_0, \cdots, s'_{kN-1})$ : the output LWE secret key
         { $\vec{S'} = \left(S'_0, \ldots, S'_{k-1}\right)$ : a GLWE secret key
         { $\forall 0 \leq i \leq k-1,\ S'_i = \sum_{j=0}^{N-1} s'_{i \cdot N+j} X^j$
         { $\alpha$ : such that $\alpha^2 \leq N$
         { $\Delta_{\mathsf{out}} = \max(\Delta_1, \Delta_2)$
Input: { $\forall 0 \leq i < \alpha,\ \mathsf{ct}_i^{(1)} \in \mathsf{LWE}_{\vec{s}}(m_i^{(1)} \cdot \Delta_1)$
       { $\forall 0 \leq i < \alpha,\ \mathsf{ct}_i^{(2)} \in \mathsf{LWE}_{\vec{s}}(m_i^{(2)} \cdot \Delta_2)$
       { $\mathsf{RLK}$ : a relinearization key for $\vec{S'}$ as defined in Algorithm 20
       { $\mathsf{KSK}$ : a key switching key from $\vec{s}$ to $\vec{S'}$ as defined in Algorithm 22
Output: $\left\{\mathsf{ct}_i^{(\mathsf{out})} \in \mathsf{LWE}_{\vec{s'}}\left(m_i^{(1)} \cdot m_i^{(2)} \cdot \Delta_{\mathsf{out}}\right)\right\}_{i=0}^{\alpha-1}$
1  begin
       /* KS from LWE to GLWE (Section 2.3.2) */
2      $\mathcal{L}_1 = \{0, 1, 2, \cdots, \alpha-1\}$;
3      $\mathcal{L}_2 = \{0, \alpha, 2\alpha, \cdots, (\alpha-1)\alpha\}$;
4      $\mathsf{CT}_1 \leftarrow \mathsf{PackingKS}(\{\mathsf{ct}_i^{(1)}\}_{i=0}^{\alpha-1}, \mathcal{L}_1, \mathsf{KSK})$ ;
5      $\mathsf{CT}_2 \leftarrow \mathsf{PackingKS}(\{\mathsf{ct}_i^{(2)}\}_{i=0}^{\alpha-1}, \mathcal{L}_2, \mathsf{KSK})$ ;
       /* GLWE multiplication (Algorithm 20) */
6      $\mathsf{CT} \leftarrow \mathsf{GLWEMult}(\mathsf{CT}_1, \mathsf{CT}_2, \mathsf{RLK})$
       /* Sample extractions (Algorithm 5) */
7      for $0 \leq i < \alpha$ do
8          $\mathsf{ct}_i^{(\mathsf{out})} \in \mathsf{LWE}_{\vec{s'}}\left(m_i^{(1)} \cdot m_i^{(2)} \cdot \Delta_{\mathsf{out}}\right) \leftarrow \mathsf{SampleExtract}\left(\mathsf{CT}, i \cdot (\alpha+1)\right)$
9      end
10 end
```

First, we use a packing key switch (Section 2.3.2) to pack the two input sets into two GLWE ciphertexts. To do so, we use two sets of indexes $\mathcal{L}_1$ and $\mathcal{L}_1$ such that $\mathcal{L}_1 = \{0, 1, 2, \cdots, \alpha - 1\}$ and $\mathcal{L}_2 = \{0, \alpha, 2\alpha, \cdots, (\alpha - 1)\alpha\}$. Then, the resulting GLWE ciphertexts are multiplied with the GLWE multiplication recalled in Algorithm 20. Finally all the coefficients at indexes $i \cdot (\alpha + 1)$ (for $0 \leq i < \alpha$) are extracted with Algorithm 5.

Sum of Products

Algorithm 24: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{PackedSumProducts}(\{\mathsf{ct}_i^{(1)}\}_{i=0}^{\alpha-1}, \{\mathsf{ct}_i^{(2)}\}_{i=0}^{\alpha-1}, \mathsf{RLK}, \mathsf{KSK})$

```
Context: { $\vec{s} = (s_0, \cdots, s_{n-1})$ :  the input LWE secret key
         { $\vec{s}' = (s'_0, \cdots, s'_{kN-1})$ :  the output LWE secret key
         { $\vec{S}' = \left(S'_0, \ldots, S'_{k-1}\right)$ :  a GLWE secret key
         { $\forall 0 \leq i \leq k-1,\ S'_i = \sum_{j=0}^{N-1} s'_{i \cdot N + j} X^j$
         { $\alpha$ : such that $\alpha \leq N$
         { $\Delta_{\mathsf{out}} = \max(\Delta_1, \Delta_2)$
Input: { $\forall 0 \leq i < \alpha,\ \mathsf{ct}_i^{(1)} \in \mathsf{LWE}_{\vec{s}}(m_i^{(1)} \cdot \Delta_1)$
       { $\forall 0 \leq i < \alpha,\ \mathsf{ct}_i^{(2)} \in \mathsf{LWE}_{\vec{s}}(m_i^{(2)} \cdot \Delta_2)$
       { $\mathsf{RLK}$ :  a relinearization key for $\vec{S}'$ as defined in algorithm 20
       { $\mathsf{KSK}$ :  a key switching key from $\vec{s}$ to $\vec{S}'$ as defined in algorithm 22
Output: $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s}'}\left(\sum_{i=0}^{\alpha-1} m_i^{(1)} \cdot m_i^{(2)} \cdot \Delta_{\mathsf{out}}\right)$
1 begin
      /* KS from LWE to GLWE (Section 2.3.2)                                    */
2     $\mathcal{L}_1 = \{0, 1, 2, \cdots, \alpha - 1\}$;
3     $\mathcal{L}_2 = \{\alpha - 1, \alpha - 2, \alpha - 3, \cdots, 0\}$;
4     $\mathsf{CT}_1 \leftarrow \mathsf{PackingKS}(\{\mathsf{ct}_i^{(1)}\}_{i=0}^{\alpha-1}, \mathcal{L}_1, \mathsf{KSK})$ ;
5     $\mathsf{CT}_2 \leftarrow \mathsf{PackingKS}(\{\mathsf{ct}_i^{(2)}\}_{i=0}^{\alpha-1}, \mathcal{L}_2, \mathsf{KSK})$ ;
      /* GLWE multiplication (Algorithm 20)                                     */
6     $\mathsf{CT} \leftarrow \mathsf{GLWEMult}(\mathsf{CT}_1, \mathsf{CT}_2, \mathsf{RLK})$
      /* Sample extraction (Algorithm 5)                                        */
7     $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s}'}\left(\sum_{i=0}^{\alpha-1} m_i^{(1)} \cdot m_i^{(2)} \cdot \Delta_{\mathsf{out}}\right) \leftarrow \mathsf{SampleExtract}\,(\mathsf{CT}, \alpha - 1)$
8 end
```

The $\mathsf{PackedSumProducts}$ algorithm also takes as input two sets of LWE ciphertexts $\left\{\mathsf{ct}_i^{(1)}\right\}_{0 \leq i < \alpha}$ and $\left\{\mathsf{ct}_i^{(2)}\right\}_{0 \leq i < \alpha}$ such that for $j \in \{1, 2\}$ and for $0 \leq i < \alpha, \mathsf{ct}_i^{(j)} \in \mathsf{LWE}_{\vec{s}}(m_i^{(j)} \cdot \Delta_j)$. Its goal is to compute an LWE encryption of the sum of products $\sum_{i=0}^{\alpha-1} m_i^{(1)} \cdot m_i^{(2)} \cdot \Delta_{\mathsf{out}}$, where $\Delta_{\mathsf{out}} = \max(\Delta_1, \Delta_2)$. The $\mathsf{PackedSumProducts}$ is described in Algorithm 24.

First, the two input sets are packed with a packing key switch into two GLWE ciphertexts with two sets of indexes $\mathcal{L}_1 = \{0, 1, 2, \cdots, \alpha - 1\}$ and $\mathcal{L}_2 = \{\alpha - 1, \alpha - 2, \alpha - 3, \cdots, 0\}$ respectively. Then, the resulting GLWE ciphertexts are multiplied with the GLWE multiplication recalled in Algorithm 20. Finally, the coefficient at index $\alpha - 1$ is extracted using Algorithm 5.

It is also possible to compute squares and a sum of squares by computing a GLWE multiplication between a GLWE ciphertext and itself.

Packed Squares

Algorithm 25: $\left\{\mathsf{ct}_i^{(\mathsf{out})}\right\}_{i=0}^{\alpha-1} \leftarrow \mathsf{PackedSquares}\left(\{\mathsf{ct}_i\}_{i=0}^{\alpha-1}, \mathsf{RLK}, \mathsf{KSK}\right)$

```
Context: $\begin{cases} \vec{s} = (s_0, \cdots, s_{n-1}) : \text{ the input LWE secret key} \\ \vec{s}' = (s'_0, \cdots, s'_{kN-1}) : \text{ the output LWE secret key} \\ \vec{S}' = \left(S'_0, \ldots, S'_{k-1}\right) : \text{ a GLWE secret key} \\ \forall 0 \le i \le k-1,\ S'_i = \sum_{j=0}^{N-1} s'_{i \cdot N + j} X^j \\ \alpha : \text{such that } 2^\alpha \le N \end{cases}$
Input: $\begin{cases} \forall 0 \le i < \alpha,\ \mathsf{ct}_i \in \mathsf{LWE}_{\vec{s}}(m_i \cdot \Delta) \\ \mathsf{RLK} : \text{ a relinearization key for } \vec{S}' \text{ as defined in Algorithm 20} \\ \mathsf{KSK} : \text{ a key switching key from } \vec{s} \text{ to } \vec{S}' \text{ as defined in Algorithm 22} \end{cases}$
Output: $\{\mathsf{ct}_i^{(\mathsf{out})} \in \mathsf{LWE}_{\vec{s}'}(m_i^2 \cdot \Delta)\}_{i=0}^{\alpha-1}$
1 begin
      /* KS from LWE to GLWE (Section 2.3.2)                              */
2     $\mathcal{L} = \{2^0 - 1, 2^1 - 1, 2^2 - 1, \cdots, 2^{\alpha-1} - 1\}$;
3     $\mathsf{CT} \leftarrow \mathsf{PackingKS}(\{\mathsf{ct}_i\}_{i=0}^{\alpha-1}, \mathcal{L}, \mathsf{KSK})$ ;
      /* GLWE Square (Algorithm 21)                                       */
4     $\mathsf{CT} \leftarrow \mathsf{GLWESquare}(\mathsf{CT}, \mathsf{RLK})$
      /* Sample extractions (Algorithm 5)                                 */
5     for $0 \le i < \alpha$ do
6         $\mathsf{ct}_i^{(\mathsf{out})} \in \mathsf{LWE}_{\vec{s}'}\left(m_i^2 \cdot \Delta\right) \leftarrow \mathsf{SampleExtract}\left(\mathsf{CT}, 2^{i+1} - 2\right)$
7     end
8 end
```

The $\mathsf{PackedSquares}$ algorithm takes as input a single set of LWE ciphertexts $\{\mathsf{ct}_i\}_{0 \le i < \alpha}$ such that for $0 \le i < \alpha, \mathsf{ct}_i \in \mathsf{LWE}_{\vec{s}}(m_i \cdot \Delta)$. Its goal is to compute LWE encryptions of the squares $m_i^2 \cdot \Delta$. The $\mathsf{PackedSquares}$ is described in Algorithm 25.

First, the input set is packed with a packing key switch into a GLWE ciphertext with the set of indexes $\mathcal{L} = \{2^0 - 1, 2^1 - 1, 2^2 - 1, \cdots, 2^{\alpha-1} - 1\}$. Then, the resulting GLWE ciphertext is squared by using the GLWE square algorithm described in Algorithm 21. Finally all the coefficients at indexes $2^{i+1} - 2$ (for $0 \le i < \alpha$) are extracted with Algorithm 5.

Sum of Squares

```
Algorithm 26: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{PackedSumSquares}\left(\{\mathsf{ct}_i\}_{i=0}^{\alpha-1}, \mathsf{RLK}, \mathsf{KSK}\right)$

Context: $\begin{cases} \vec{s} = (s_0, \cdots, s_{n-1}) : \text{ the input LWE secret key} \\ \vec{s}' = (s'_0, \cdots, s'_{kN-1}) : \text{ the output LWE secret key} \\ \vec{S}' = \left(S'_0, \ldots, S'_{k-1}\right) : \text{ a GLWE secret key} \\ \forall 0 \le i \le k-1,\ S'_i = \sum_{j=0}^{N-1} s'_{i \cdot N + j} X^j \\ \alpha : \text{such that } 2^\alpha \le N \end{cases}$
Input: $\begin{cases} \forall 0 \le i < \alpha,\ \mathsf{ct}_i \in \mathsf{LWE}_{\vec{s}}(m_i \cdot \Delta) \\ \mathsf{RLK} : \text{ a relinearization key for } \vec{S}' \text{ as defined in algorithm 20} \\ \mathsf{KSK} : \text{ a key switching key from } \vec{s} \text{ to } \vec{S}' \text{ as defined in algorithm 22} \end{cases}$
Output: $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s}'}\left(\sum_{i=0}^{\alpha-1} m_i^2 \cdot 2\Delta\right)$
1 begin
      /* KS from LWE to GLWE (Section 2.3.2)                          */
2     $\mathcal{L}_1 = \{0, 1, 2, \cdots, \alpha - 1\}$;
3     $\mathcal{L}_2 = \{2\alpha - 1, 2\alpha - 2, 2\alpha - 3, \cdots, \alpha\}$;
4     $\mathsf{CT} \leftarrow \mathsf{PackingKS}(\{\mathsf{ct}_i\}_{i=0}^{\alpha-1} || \{\mathsf{ct}_i\}_{i=0}^{\alpha-1}, \mathcal{L}_1 || \mathcal{L}_2, \mathsf{KSK})$ ;
      /* GLWE square (Algorithm 21)                                   */
5     $\mathsf{CT} \leftarrow \mathsf{GLWESquare}(\mathsf{CT}, \mathsf{RLK})$
      /* Sample extraction (Algorithm 5)                              */
6     $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s}'}\left(\sum_{i=0}^{\alpha-1} m_i^2 \cdot 2\Delta\right) \leftarrow \mathsf{SampleExtract}(\mathsf{CT}, 2\alpha - 1)$
7 end
```

The PackedSumSquares algorithm takes as input a single set of LWE ciphertexts $\{\mathsf{ct}_i\}_{0 \le i < \alpha}$ such that for $0 \le i < \alpha, \mathsf{ct}_i \in \mathsf{LWE}_{\vec{s}}(m_i \cdot \Delta)$. Its goal is to compute a LWE encryption of the sum of squares $\sum_{i=0}^{\alpha-1} m_i^2 \cdot 2\Delta$. The PackedSumSquares is described in Algorithm 26.

To achieve this goal, the input set is packed with a packing key switch into a GLWE ciphertext with redundancy, using two indexes sets $\mathcal{L}_1 = \{0, 1, 2, \cdots, \alpha - 1\}$ and $\mathcal{L}_2 = \{2\alpha - 1, 2\alpha - 2, 2\alpha - 3, \cdots, \alpha\}$. Then, the resulting GLWE ciphertext is squared using the GLWE square algorithm described in Algorithm 21. Finally the coefficient at index $2\alpha - 1$ is extracted with Algorithm 5.

Notice that we could also compute packed products and a packed sum of products with a GLWE squaring algorithm by changing $\mathcal{L}$, $\mathcal{L}_1$ and $\mathcal{L}_2$ and also extracting different coefficients. Also note that for these four algorithms, there are restrictions regarding the maximum value that α can take each time. The noise analysis of PackedMult, PackedSumProducts, PackedSquares and PackedSumSquares can be deduced from the results of Theorem 27.

Chapter 6

Without Padding Programmable Bootstrap

In Limitation 1, we explained that we need to have a bit of padding to perform a correct PBS (Algorithm 10 and Theorem 14) with an arbitrary lookup table evaluation. The only exception is when the function is negacyclic (Remark 8), in this case, we do not need a bit of padding. When working with Boolean messages, this is not an issue as we can use negacyclic functions only. If we want to support integer messages, we need to guarantee that we have a bit of padding before applying a PBS.

This constraint makes it difficult to use TFHE in practice. To guarantee a bit of padding, we need the degree of fullness introduced in Section 2.4 to track the maximum value of the message and make sure that it does not overwrite the most significant bit. If we are able to remove this constraint, we could create lots of new FHE algorithms, for instance, we could apply a function on the least significant bits of a message without having to pay the price of a function on the whole message.

In this chapter, we present 3 ways to build a *Without Padding Programmable Bootstrapping* (WoP-PBS).

The first two constructions leverage the LWE multiplication we introduced in Section 5.4 (Algorithm 22 and Theorem 27) and the generalized PBS introduced in Section 5.1 (Algorithm 17 and Theorem 22). Intuitively, we use the multiplication to correct the sign after the lookup table evaluation. Before those constructions, there was no way to compute a PBS without a bit of padding. On top of those new algorithms, we overcome Limitation 5 by introducing a way to homomorphically change the encoding of a message to have a radix encoding as described in Section 2.4.

Then, we introduce another WoP-PBS by leveraging this time the circuit bootstrap recalled in Algorithm 11, the PBS (Algorithm 17) and the vertical packing (Algorithm 13). This algorithm can be easily tweaked to support several LUT evaluations which, with the PBSmanyLUT introduced in Algorithm 18 and Theorem 23, overcome Limitation 6.

The new WoP-PBS scales particularly well with the precision of the message, and we will use it in the description of our homomorphic integers (see Chapter 7). With it, applying arbitrary functions over radix or CRT encoded messages becomes possible, thus overcoming Limitation 13.

All those new algorithms can be parallelized to leverage a multi-thread execution environment which overcomes Limitation 3.

6.1 WoP-PBS: First Attempt

In this section, we describe the first known WoP-PBS which means a PBS without a bit of padding. Having a bit of padding is troublesome when using TFHE in practice because we cannot leverage the native reduction modulo q of the scheme to build an FHE modular arithmetic as explained in Limitation 1.

In this section, we give two WoP-PBS algorithms (Algorithms 27 and 28) that are built on top of the LWEMult (Algorithm 22).

6.1.1 WoP-PBS from sign correction

A big constraint with the TFHE PBS is the *negacyclicity of the rotation of the LUT*. It implies the need for a *padding bit* (as mentioned in Limitation 1). We propose a solution to remove that requirement, by using the aforementioned LWE multiplication (Algorithm 20) and the generalized PBS (Algorithm 17). This new bootstrapping is called the *programmable bootstrapping without padding* (WoP-PBS) and a first version is described in Algorithm 27 and in Theorem 28.

Theorem 28 (PBS Without Padding (V1)) *Let $\vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n$ be a binary LWE secret key. Let $\vec{S'} = \left(S'_0, \ldots, S'_{k-1}\right) \in \mathfrak{R}_q^k$ be a GLWE secret key such that $S'_i = \sum_{j=0}^{N-1} s'_{i \cdot N+j} X^j \in \mathfrak{R}_q$, and $\vec{s'} = (s'_0, \cdots, s'_{kN-1}) \in \mathbb{Z}_q^{kN}$ be the corresponding binary LWE key. Let $P_f \in \mathfrak{R}_q$ (resp. $P_{\mathbb{1}} \in \mathfrak{R}_q$) be an r-redundant LUT (Definition 12) for the function $f : \mathbb{Z} \mapsto \mathbb{Z}$, (resp. the constant function $x \mapsto 1$) and $\Delta_{\mathsf{out}} \in \mathbb{Z}_q$ be the output scaling factor. Let CT_f be a (possibly trivial) GLWE encryption of $P_f \cdot \Delta_{\mathsf{out}}$ and $\mathsf{CT}_{\mathbb{1}}$ be a trivial GLWE encryption of $P_{\mathbb{1}} \cdot \Delta_{\mathsf{out}}$. Let $(\varkappa, \vartheta) \in \mathbb{Z} \times \mathbb{N}$ be the two integer parameters defining (along with N) the chunk of the plaintext that is going to be bootstrapped, such that $\frac{q 2^{\vartheta}}{\Delta_{\mathsf{in}} 2^{\varkappa}} < 2N$, and let $(\beta, m') = \mathsf{PTModSwitch}_q(m, \Delta_{\mathsf{in}}, \varkappa, \vartheta) \in \{0, 1\} \times \mathbb{N}$.*

Algorithm 27: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{WoP\text{-}PBS}_1(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \mathsf{RLK}, \mathsf{KSK}, P_f, \Delta_{\mathsf{out}}, \varkappa, \vartheta)$

Context:
$$\begin{cases} \vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n \\ \vec{s'} = (s'_0, \cdots, s'_{kN-1}) \in \mathbb{Z}_q^{kN} \\ \vec{S'} = \left(S'^{(0)}, \ldots, S'^{(k-1)}\right) \in \mathfrak{R}_q^k \\ \forall 0 \le i \le k-1,\ S'^{(i)} = \sum_{j=0}^{N-1} s'_{i\cdot N+j} X^j \in \mathfrak{R}_q \\ f : \mathbb{Z} \to \mathbb{Z} : \text{a function} \\ P_{\mathbb{1}} \in \mathfrak{R}_q :\ \text{a redundant } \mathsf{LUT} \text{ for } x \mapsto 1 \\ (\beta, m') = \mathsf{PTModSwitch}_q(m, \Delta, \varkappa, \vartheta) \in \{0,1\} \times \mathbb{N} \\ \mathsf{CT}_f \in \mathsf{GLWE}_{\vec{S'}}\left(P_f \cdot \Delta_{\mathsf{out}}\right) \in \mathfrak{R}_q^{k+1} \text{ (might be a trivial encryption)} \\ \mathsf{CT}_{\mathbb{1}} \in \mathfrak{R}_q^{k+1} :\ \text{a trivial encryption of } P_{\mathbb{1}} \cdot \Delta_{\mathsf{out}} \end{cases}$$

Input:
$$\begin{cases} \mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}}(m \cdot \Delta_{\mathsf{in}}) = (a_0, \cdots, a_{n-1}, a_n = b) \in \mathbb{Z}_q^{n+1} \\ \mathsf{BSK} = \left\{\mathsf{BSK}_i \in \mathsf{GGSW}_{\vec{S'}}^{(\beta,\ell)}\left(s_i\right)\right\}_{0 \le i \le n-1} :\ \text{a bootstrapping key from } \vec{s} \text{ to } \vec{S'} \\ \mathsf{RLK} = \left\{\overline{\mathsf{CT}}_{i,j} \in \mathsf{GLev}_{\vec{S'}}^{(\beta,\ell)}\left(S'_i \cdot S'_j\right)\right\}_{0 \le i \le k-1}^{0 \le j \le i} :\ \text{a relinearization key for } \vec{S'} \\ \mathsf{KSK} = \left\{\overline{\mathsf{CT}}_i \in \mathsf{GLev}_{\vec{S'}}^{(\beta,\ell)}\left(s'_i\right)\right\}_{0 \le i \le kN-1} :\ \text{a key switching key from } \vec{s'} \text{ to } \vec{S'} \\ P_f \in \mathfrak{R} : \text{a redundant LUT for } x \mapsto f(x) \\ \Delta_{\mathsf{out}} \in \mathbb{Z}_q : \text{the output scaling factor} \\ (\varkappa, \vartheta) \in \mathbb{Z} \times \mathbb{N} : \text{define along with } N \text{ the window size} \end{cases}$$

Output: $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s'}}(f(m') \cdot \Delta_{\mathsf{out}})$ if we respect the requirements of Theorem 28

```
1 begin
      /* Compute two PBSs in parallel:                                  */
2     $\mathsf{ct}_{\mathrm{f}} \in \mathsf{LWE}_{\vec{s'}}((-1)^\beta \cdot f(m') \cdot \Delta_{\mathsf{out}}) \leftarrow \mathsf{GenPBS}\left(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \mathsf{CT}_f, \varkappa - 1, \vartheta\right)$ ;
3     $\mathsf{ct}_{\mathrm{Sign}} \in \mathsf{LWE}_{\vec{s'}}((-1)^\beta \cdot \Delta_{\mathsf{out}}) \leftarrow \mathsf{GenPBS}\left(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \mathsf{CT}_{\mathbb{1}}, \varkappa - 1, \vartheta\right)$ ;
      /* Compute the multiplication                                     */
4     $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{LWEMult}(\mathsf{ct}_{\mathrm{f}}, \mathsf{ct}_{\mathrm{Sign}}, \mathsf{RLK}, \mathsf{KSK})$;
5 end
```

Let $\mathsf{KSK}{=}\left\{\overline{\mathsf{CT}}_i{\in}\mathsf{GLev}_{\vec{S'}}^{(\mathfrak{B},\ell)}\left(s'_i\right)\right\}_{0\leq i\leq n-1}$ *be a key switching key from* $\vec{s'}$ *to* $\vec{S'}$*, with noise sampled respectively from* $\chi_{\sigma^{(1)}}$ *and* $\chi_{\sigma^{(2)}}$*. Let* $\mathsf{RLK}{=}\left\{\overline{\mathsf{CT}}_{i,j}{\in}\mathsf{GLev}_{\vec{S'}}^{(\mathfrak{B},\ell)}\left(S'_i{\cdot}S'_j\right)\right\}_{0\leq i\leq k-1}^{0\leq j\leq i}$ *be a relinearization key for* $\vec{S'}$*, defined as in Theorem 26. Let* $\mathsf{BSK}{=}\left\{\overline{\overline{\mathsf{CT}}}_i{\in}\mathsf{GGSW}_{\vec{S'}}^{\mathfrak{B},\ell}(s_i)\right\}_{i=0}^{n-1}$ *be a bootstrapping key from* $\vec{s}$ *to* $\vec{S'}$*.*

Then Algorithm 27 takes as input an LWE ciphertext $\mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}}(m \cdot \Delta_{\mathsf{in}}) \in \mathbb{Z}_q^{n+1}$ *where* $\mathsf{ct}_{\mathsf{in}} = (a_0, \cdots, a_{n-1}, a_n = b)$*, with noise sampled from* $\chi_{\sigma_{\mathsf{in}}}$*, and returns an LWE ciphertext* $\mathsf{ct}_{\mathsf{out}} \in \mathbb{Z}_q^{kN+1}$ *under the secret key* $\vec{s'}$ *encrypting the messages* $f(m') \cdot \Delta_{\mathsf{out}}$ *if and only if the input noise variance satisfies the constraint of Theorem 22.*

The output ciphertext noise variance verifies $\mathsf{Var}(\mathsf{WoP\text{-}PBS}_1) = \mathsf{Var}(\mathsf{LWEMult})$ *with input variances for the LWE multiplication (Algorithm 22) defined as* $\sigma_i^2 = \mathsf{Var}(\mathsf{GenPBS})$*, for* $i \in \{1, 2\}$*.*

The cost of Algorithm 27 is

$$\begin{aligned}\mathsf{Cost}\left(\mathsf{WoP\text{-}PBS}_1\right)^{(n,\ell_{\mathsf{PBS}},k_1,N_1,\ell_{\mathsf{KS}},\ell_{\mathsf{RL}},k_2,N_2)} &= 2 \cdot \mathsf{Cost}\left(GenPBS\right)^{(n,\ell_{\mathsf{PBS}},k_1,N_1)} \\ &+ \mathsf{Cost}\left(LWEMult\right)^{(\ell_{\mathsf{KS}},\ell_{\mathsf{RL}},N_1,k_2,N_2)} \quad .\end{aligned} \tag{6.1}$$

Proof 28 (Theorem 28) *We only provide a proof of correctness of the algorithm, considering that the noise and the cost are directly deduced from the* GenPBS *and* LWEMult *algorithms (see Theorem 22 and Theorem 27).*

Both of the GenPBS *are applied with the same parameters except for the evaluated function (*P_f *or* $P_{\mathbb{1}}$*). Thus, in both ciphertexts* ct_{f} *and* $\mathsf{ct}_{\mathrm{Sign}}$ *the value of* β *is the same. Then,* $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s}}((-1)^{2\beta} \cdot f(m') \cdot \Delta_{\mathsf{out}}) = \mathsf{LWE}_{\vec{s}}(f(m') \cdot \Delta_{\mathsf{out}})$*.* ☐

Remark 32 (Several Secret Keys) *Observe that, in Algorithm 27 we set* KSK *as a key switching key from* $\vec{s'}$ *to* $\vec{S'}$ *where* $\vec{s'}$ *is the LWE secret key composed of the coefficients in* $\vec{S'}$*. In practice, the key switching can be done to a key* $\vec{S''}$*, that has nothing to do with* $\vec{s'}$*. In this case, the* RLK *should be adapted as well to the key* $\vec{S''}$*.*

Below, we describe several variants of Algorithm 27. These improvements could increase the noise but lower the cost of the algorithm.

The two GenPBS have the same input ciphertext. To make the evaluation more efficient (evaluating a single bootstrap instead of two), it is possible to use either the multi-value bootstrap (Algorithm 14) described in [CIM19], which will be faster but at the cost of a higher output noise. Another option would be to take advantage of the PBSmanyLUT, which we describe in detail in Section 5.2 (Algorithm 18) if the input message is small enough (*cf.* Remark 33).

There could only be one key switch done in LWEMult instead of two if one of the two inputs is provided as a GLWE ciphertext and if we remove the final sample extract in one of the two GenPBS.

The LWEMult on line 4 can be replaced by a **LWESquare** which corresponds to a PackedSquares with $\alpha = 2$ (Algorithm 25) which is faster.

6.1.2 WoP-PBS from LUT Splitting

Another big constraint with the PBS of TFHE is that *the polynomial size N is directly linked to the size of the message we want to bootstrap* (as mentioned in Limitation 2). Since N is a power of two, the smallest growth of the polynomial size slows down the computation by more than a factor 2 as the cost of TFHE's PBS is proportional to the cost of FFT conversions, $N \log_2(N)$. Keeping that in mind, we offer a different way to perform a bootstrap without padding in Algorithm 28 which can be more efficient in a multi-threaded machine. The main idea behind this algorithm is to write a message m as $\beta || m'$ where β is the most significant bit and m' the rest of the message. The function f to be computed is broken into two functions: f_0 and f_1. We want f_0 if β is equal to 0 and f_1 if $\beta = 1$. We use β as an encrypted decision bit, so we can choose between $f_0(m')$ or $f_1(m')$ thanks to the LWEMult (Algorithm 22).

Precision	n	PBS/KS $\log_2(N)$	BR $\log_2(\mathfrak{B})$	BR ℓ	KS $\log_2(\mathfrak{B})$	KS ℓ	Relin $\log_2(\mathfrak{B})$	Relin ℓ
1	550	11	17	2	21	2	30	1
2	550	11	13	3	17	2	30	1
3	550	11	10	4	17	2	20	2
4	550	11	9	5	13	3	20	2
5	550	11	5	9	10	4	24	2
6	550	12	10	5	12	4	24	2
7	550	12	4	13	10	5	24	2

Table 6.1: Parameter sets of a $\mathsf{WoP\text{-}PBS}_2$ followed by a GLWE multiplication for different precisions and for 128 bits of security. In the table, *KS* means Key Switching, *BR* means Blind Rotation and *Relin* means Relinearization. This table was generated in May 2021 with $\alpha = 0.05287332817861731$ and $\beta = 4.551576767993042$ (as defined in Section 3.1).

We give the complete set of cryptographic parameters for different precisions in Table 6.1. In a nutshell, for precisions from 1 to 5 bits, we use $\log_2(N) = 11$ and for 6 and 7 bits of precisions, we use $\log_2(N) = 12$.

Algorithm 28: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{WoP\text{-}PBS}_2(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \mathsf{RLK}, \mathsf{KSK}, P_f, \Delta_{\mathsf{out}}, \varkappa, \vartheta)$

```
Context: {
  $\vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n$
  $\vec{s}' = (s'_0, \cdots, s'_{kN-1}) \in \mathbb{Z}_q^{kN}$
  $\vec{S}' = \left(S'^{(0)}, \ldots, S'^{(k-1)}\right) \in \mathfrak{R}_q^k$
  $\forall 0 \le i \le k-1,\ S'^{(i)} = \sum_{j=0}^{N-1} s'_{i \cdot N + j} X^j \in \mathfrak{R}_q$
  $f_0(x) = f(x) = f_1(x-p)$ for a certain $p$
  $(\beta, m') = \mathsf{PTModSwitch}_q(m, \Delta, \varkappa, \vartheta) \in \{0,1\} \times \mathbb{N}$
  $P_{\mathbb{1}} \in \mathfrak{R}_q$ : as defined in Algorithm 27
  $\mathsf{CT}_{f_i} \in \mathsf{GLWE}_{\vec{S}'}\left(P_{f_i} \cdot \Delta_{\mathsf{out}}\right) \in \mathfrak{R}_q^{k+1}$ (might be a trivial encryption)
  $\mathsf{CT}_{\mathbb{1}} \in \mathfrak{R}_q^{k+1}$ : a trivial encryption of $P_{\mathbb{1}} \cdot \frac{\Delta_{\mathsf{out}}}{2}$
  $P_{f_0}, P_{f_1} \in \mathfrak{R}_q$ : redundant LUTs of the two halves of $P_f$
Input: {
  $\mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}}(m \cdot \Delta_{\mathsf{in}}) = (a_0, \cdots, a_{n-1}, a_n = b) \in \mathbb{Z}_q^{n+1}$
  $\mathsf{BSK}, \mathsf{KSK}, \mathsf{RLK}$ : as defined in Algorithm 27
  $P_f \in \mathfrak{R}_q$ : a redundant LUT for $x \mapsto f(x)$
  $\Delta_{\mathsf{out}} \in \mathbb{Z}_q$ : the output scaling factor
  $(\varkappa, \vartheta) \in \mathbb{Z} \times \mathbb{N}$ : define the window size along with $N$
Output: $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s}'}(f(m) \cdot \Delta_{\mathsf{out}})$ if we respect the requirements of Theorem 29
1  begin
       /* Compute in parallel 3 PBS:                                        */
2      $\mathsf{ct}_{f_0} \in \mathsf{LWE}_{\vec{s}'}((-1)^\beta \cdot \Delta_{\mathsf{out}} \cdot f_0(m')) \leftarrow \mathsf{GenPBS}(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \mathsf{CT}_{f_0}, \varkappa, \vartheta)$ ;
3      $\mathsf{ct}_{f_1} \in \mathsf{LWE}_{\vec{s}'}((-1)^\beta \cdot \Delta_{\mathsf{out}} \cdot f_1(m')) \leftarrow \mathsf{GenPBS}(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \mathsf{CT}_{f_1}, \varkappa, \vartheta)$ ;
4      $\mathsf{ct}_{\mathrm{Sign}} \in \mathsf{LWE}_{\vec{s}'}((-1)^\beta \cdot \frac{\Delta_{\mathsf{out}}}{2}) \leftarrow \mathsf{GenPBS}(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \mathsf{CT}_{\mathbb{1}}, \varkappa, \vartheta)$ ;
       /* Compute two sums in parallel:                                     */
5      $\mathsf{ct}_{\beta_0} \in \mathsf{LWE}_{\vec{s}'}((1-\beta) \cdot \Delta_{\mathsf{out}}) \leftarrow \mathsf{ct}_{\mathrm{Sign}} + (\vec{0}, \frac{\Delta_{\mathsf{out}}}{2})$ ;
6      $\mathsf{ct}_{\beta_1} \in \mathsf{LWE}_{\vec{s}'}(-\beta \cdot \Delta_{\mathsf{out}}) \leftarrow \mathsf{ct}_{\mathrm{Sign}} - (\vec{0}, \frac{\Delta_{\mathsf{out}}}{2})$ ;
       /* Compute two multiplications in parallel:                          */
7      $\mathsf{ct}_{\beta \cdot f_0} \leftarrow \mathsf{LWEMult}(\mathsf{ct}_{f_0}, \mathsf{ct}_{\beta_0}, \mathsf{RLK}, \mathsf{KSK})$ ;
8      $\mathsf{ct}_{\beta \cdot f_1} \leftarrow \mathsf{LWEMult}(\mathsf{ct}_{f_1}, \mathsf{ct}_{\beta_1}, \mathsf{RLK}, \mathsf{KSK})$ ;
       /* Add the previous results:                                         */
9      $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{ct}_{\beta \cdot f_0} + \mathsf{ct}_{\beta \cdot f_1}$ ;
10 end
```

Theorem 29 (PBS Without Padding (V2)) *Let f_0 and f_1 be the two functions representing f such that $f_0(x) = f(x) = f_1(x-p)$ for a certain $p \in \mathbb{N}$. Then, under the same hypotheses than the ones of Theorem 28, Algorithm 28 takes as input an LWE ciphertext $\mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}}(m \cdot \Delta_{\mathsf{in}}) = (a_0, \cdots, a_{n-1}, a_n = b)$, with noise drawn from $\chi_{\sigma_{\mathsf{in}}}$, and returns an LWE ciphertext $\mathsf{ct}_{\mathsf{out}}$ under the secret key $\vec{s}'$ encrypting the messages $f(m') \cdot \Delta_{\mathsf{out}}$ if and only if the input noise variance verifies the constraint of Theorem 22.*

The noise variance of the output ciphertext is

$$\mathsf{Var}(\mathsf{WoP\text{-}PBS}_2) = 2 \cdot \mathsf{Var}(\mathsf{LWEMult}), \tag{6.2}$$

with input variances of the $\mathsf{LWEMult}$ *defined as*

$$\sigma_i^2 = \mathsf{Var}(\mathsf{GenPBS}), \forall i \in \{1,2\} \ . \tag{6.3}$$

The cost of Algorithm 28 is

$$\begin{aligned}\mathsf{Cost}\left(\mathsf{WoP\text{-}PBS}_2\right)^{(n,\ell_{\mathsf{PBS}},k_1,N_1,\ell_{\mathsf{KS}},\ell_{\mathsf{RL}},k_2,N_2)} &= 3\cdot \mathsf{Cost}\left(GenPBS\right)^{(n,\ell_{\mathsf{PBS}},k_1,N_1)} \\ &+ 2\mathsf{Cost}\left(LWEMult\right)^{(\ell_{\mathsf{KS}},\ell_{\mathsf{RL}},N_1,k_2,N_2)} \\ &+ (N_2+3)\mathsf{Cost}\left(add\right) \ .\end{aligned} \tag{6.4}$$

Proof 29 (Theorem 29) *We have* $\mathsf{ct}_{\beta_0} \in \mathsf{LWE}_{\vec{s'}}\left(\frac{\Delta_{\mathsf{out}}}{2}((-1)^\beta+1)\right)$. *If* $\beta = 0$, *then* $\mathsf{ct}_{\beta_0} \in \mathsf{LWE}_{\vec{s'}}(\Delta_{\mathsf{out}})$ *else* $\mathsf{ct}_{\beta_0} \in \mathsf{LWE}_{\vec{s'}}(0)$. *So,* $\mathsf{ct}_{\beta_0} \in \mathsf{LWE}_{\vec{s'}}((1-\beta)\Delta_{\mathsf{out}})$. *Similarly, we obtain* $\mathsf{ct}_{\beta_1} \in \mathsf{LWE}_{\vec{s'}}((-\beta)\Delta_{\mathsf{out}})$.

Consequently,

$$\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s'}}\left((-1)^\beta\cdot(1-\beta)\cdot\Delta_{\mathsf{out}}\cdot f_0(m') + (-1)^\beta\cdot(-\beta)\cdot\Delta_{\mathsf{out}}\cdot f_1(m')\right) .$$

Thus, if $\beta = 0$, $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s'}}\left(f_0(m')\cdot\Delta_{\mathsf{out}}\right)$ *else* $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s'}}\left(f_1(m')\cdot\Delta_{\mathsf{out}}\right)$, *as expected.*

$\square$

Below, we describe several variants of Algorithm 28. These improvements could increase the noise but they also decrease the cost of the algorithm.

- The three GenPBS have the same input ciphertext. As we observed for Algorithm 27, to make the evaluation more efficient by evaluating a single bootstrapping instead of three, it is possible to use either the multi-value bootstrap (Algorithm 14) described in [CIM19] or to take advantage of the PBSmanyLUT (Algorithm 18 and *cf.* Remark 33).
- We could remove two key switches (among four) as explained for the $\mathsf{WoP\text{-}PBS}_1$.
- To improve both performance and noise, in practice, we can do a lazy relinearization as described in [Lee+20b], i.e., the relinearization step of the two LWEMult is done after the final addition.

- The two LWEMult followed by the final addition can be replaced by a PackedSumProducts (Algorithm 24).

Remark 33 (Combining PBSmanyLUT and WoP-PBS) *Observe that the* PBSmanyLUT *(Algorithm 18) and the* WoP-PBS *(Algorithm 27 and Algorithm 28) can be combined in two different ways:*

1. *Using* PBSmanyLUT *to improve* WoP-PBS*: In* WoP-PBS$_1$*,* ct_{Sign} *and* ct_f *resulting from distinct* GenPBS*s can be evaluated at once by using a single* PBSmanyLUT*. Similarly, in* WoP-PBS$_2$*,* ct_{Sign}*,* ct_{f_0} *and* ct_{f_1} *can be evaluated all at once. In both cases, this variant can be applied only if the polynomial size chosen for the* WoP-PBS *is large enough to allow multiple LUT evaluations (i.e, if precision is not yet a bottleneck condition): this variant of the* WoP-PBS *will improve the cost of the algorithm, without impacting the noise growth.*

2. *Using* WoP-PBS *to improve* PBSmanyLUT*: The* PBSmanyLUT *algorithm implicitly performs a* GenPBS *with a special modulus switching. This* GenPBS *can actually be replaced by a* WoP-PBS *(with the same special modulus switching) as a* WoP-PBS *performs the same operation as* GenPBS*, without the constraint of the bit of padding. This technique is what we call* WoPBSmanyLUT*.*

6.1.3 Large Precision Without Padding (Programmable) Bootstrapping

Using one of the WoP-PBS techniques introduced in Algorithms 27 and 28, we build two new algorithms that can work on messages encoded with a radix or a CRT encoding as described in Section 2.4.

We first describe a way to efficiently reduce the noise of an LWE ciphertext with larger precision (bootstrap) and then show how to also evaluate a function on such ciphertexts (programmable). These algorithms do not require the input LWE ciphertext to have a bit of padding.

Larger Precision Without Padding Bootstrapping

We introduce a new procedure in Algorithm 29 to homomorphically decompose a message encrypted inside a ciphertext in α ciphertexts each encrypting a small chunk of the original message. It means that this algorithm will be able to convert a message encoded with the traditional method (Definition 13) to a message encoded with the radix encoding described in Section 2.4 and Definition 15.

This algorithm is especially efficient because we begin by extracting the least significant bits instead of the most significant bits. To do so, we use the previously introduced parameter $\varkappa$ to remove some of the most significant bits of the input message m and apply the bootstrapping algorithm on the remaining bits as described in Section 5.1. The bootstrapping algorithm must be a $\mathsf{WoP\text{-}PBS}$ (Algorithm 27 or Algorithm 28) as the value of the most significant bit is not guaranteed to be set to zero. This procedure allows us to obtain an encryption of the least significant bits of the message. Next, by subtracting this result from the input ciphertext, we remove the least significant bits of the input message. This gives a new ciphertext encrypting only the most significant bits of the input message. From now on, this procedure is repeated on the resulting ciphertext until we obtain α ciphertexts, each encrypting $m_i\Delta_i$ such that $m_{\mathsf{in}}\Delta_{\mathsf{in}} = \sum_{i=0}^{\alpha-1} m_i\Delta_i$. This process is somehow similar to the approach called *Digit Extraction* applied on the BGV/BFV schemes, presented in [HS15; CH18].

This entails a significantly better cost than the naive method which consists in bootstrapping the whole message several times to extract each chunk. In this new method, each bootstrap only needs a ring dimension big enough to bootstrap correctly the number of bits of each chunk instead of having to be big enough to bootstrap correctly the total number of bits of the input ciphertext.

Efficiency might be improved within the multiplication inside each $\mathsf{WoP\text{-}PBS}$ by adding a keyswitching during the relinearization step to reduce the size of the LWE dimension. As the cost of the $\mathsf{WoP\text{-}PBS}$ depends on this LWE dimension, this will result in a faster version of Algorithm 29.

Lemma 1 *Let* $\mathsf{ct_{in}} \in \mathsf{LWE}_{\vec{s}}(m_{\mathsf{in}} \cdot \Delta_{\mathsf{in}}) \in \mathbb{Z}_q^{n+1}$ *be an LWE ciphertext, encrypting* $m_{\mathsf{in}} \cdot \Delta_{\mathsf{in}} \in \mathbb{Z}_q$. *under the LWE secret key* $\vec{s} = (s_0, \ldots, s_{n-1}) \in \mathbb{Z}_q^n$, *with noise sampled from* χ_σ. *Let* $\mathsf{BSK}, \mathsf{KSK}$ *and* RLK *as defined in Theorem 28. Let* $\mathcal{L} = \{d_i\}_{i\in[0,\alpha-1]}$ *with* $d_i \in \mathbb{N}^*$ *s.t.* $\Delta_{\mathsf{in}} 2^{\sum_{i=0}^{\alpha-1} d_i} \leq q$ *be the list defining the bit size of each output chunk. Algorithm 29 computes* $\alpha \in \mathbb{N}^*$ *new LWE ciphertexts* $\{\mathsf{ct}_{\mathsf{out},i}\}_{i\in[0,\alpha-1]}$, *where each one of them encrypts* $m_i \cdot \Delta_i$, *where* $\Delta_i = \Delta_{\mathsf{in}} \cdot 2^{\sum_{j=1}^{i-1} d_j}$, *under the secret key* $\vec{s'}$. *The variances of the noise is*

$$\mathsf{Var}(\mathsf{ct}_{\mathsf{out},i}) = \mathsf{Var}(\mathsf{WoP\text{-}PBS}) \tag{6.5}$$

Algorithm 29: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{Decomp}(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \mathsf{RLK}, \mathsf{KSK}, \mathcal{L})$

```
Context: $\begin{cases} \vec{s} = (s_0, \cdots, s_{n-1}) \in \mathbb{Z}_q^n \\ \vec{s}' = (s'_0, \cdots, s'_{kN-1}) \in \mathbb{Z}_q^{kN} \\ \vec{S}' = \left(S'^{(0)}, \ldots, S'^{(k-1)}\right) \in \mathfrak{R}_q^k \\ \forall 0 \le i \le k-1,\ S'^{(i)} = \sum_{j=0}^{N-1} s'_{i\cdot N+j} X^j \in \mathfrak{R}_q \\ \{P_{f_i}\}_{i\in[0,\alpha-1]} : \text{LUTs for the functions } f_i \\ \forall i \in [1, \alpha-1], \Delta_i = \Delta_{\mathsf{in}} \cdot 2^{\sum_{j=1}^{i-1} d_j} \le q \\ \Delta_0 = \Delta_{\mathsf{in}}, m_{\mathsf{in}}\Delta_{\mathsf{in}} = \sum_{i=0}^{\alpha-1} m_i \Delta_i \end{cases}$
Input: $\begin{cases} \mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}}(m_{\mathsf{in}} \cdot \Delta_{\mathsf{in}}) \in \mathbb{Z}_q^{n+1} \\ \mathsf{BSK}, \mathsf{KSK}, \mathsf{RLK} : \text{as defined in Algorithm 27} \\ \mathcal{L} = \{d_i\}_{i\in[0,\alpha-1]} \text{ with } d_i \in \mathbb{N}^* \end{cases}$
Output: $\{\mathsf{ct}_{\mathsf{out},i} \in \mathsf{LWE}_{\vec{s}'}(m_i \cdot \Delta_i)\}_{i\in[0,\alpha-1]}$
begin
    $\mathsf{ct} \leftarrow \mathsf{ct}_{\mathsf{in}}$
    for $i \in [0, \alpha-1]$ do
        $\varkappa_i \leftarrow \sum_{j=i+1}^{\alpha-1} d_j$
        $\mathsf{ct}_{\mathsf{out},i} \leftarrow \mathsf{WoP\text{-}PBS}(\mathsf{ct}, \mathsf{BSK}, \mathsf{RLK}, \mathsf{KSK}, P_{f_i}, \Delta_i, \varkappa_i, 0)$
        $\mathsf{ct} \leftarrow \mathsf{ct} - \mathsf{ct}_{\mathsf{out},i}$
    end
end
```

The cost is

$$\begin{aligned} \mathsf{Cost}\,(\mathit{Decomp})^{(n,\ell_{\mathsf{PBS}},k_1,N_1,\ell_{\mathsf{KS}},\ell_{\mathsf{RL}},\alpha)} &= \alpha \mathsf{Cost}\,(\mathsf{WoP\text{-}PBS}_1)^{(n,\ell_{\mathsf{PBS}},k_1,N_1,\ell_{\mathsf{KS}},\ell_{\mathsf{RL}},1,n)} \\ &+ \alpha(n+1)\mathsf{Cost}\,(\mathit{add}) + \left(\frac{\alpha(\alpha+1)}{2}\right)\mathsf{Cost}\,(\mathit{add}) \end{aligned} \tag{6.6}$$

An immediate application of Algorithm 29 is a high precision bootstrap algorithm. By using the decomposition and then adding each $\mathsf{ct}_{\mathsf{out},i}$, one can get, with the right parameters, a noise smaller than the one of the input ciphertext.

Larger Precision WoP-PBS

With the Tree-PBS and the ChainPBS algorithms introduced in [GBA21] and recalled in Section 2.3 and Algorithm 15, we can compute large precision programmable bootstraps assuming that the input ciphertexts are already decomposed in chunks. In a nutshell, the idea behind the Tree-PBS is to encode a high-precision function in several LUTs. The first input ciphertext is used to select a subset among all the LUTs. This subset is then rearranged thanks to a key switching to build new encrypted LUTs. The previous steps can be repeated on the second input ciphertext, and so on. The Tree-PBS relies on the multi-output bootstrap from [CIM19] recalled in Section 2.3 and Algorithm 14.

Thanks to Algorithm 29, we are able to efficiently decompose a ciphertext. This allows to quickly switch from one representation (one ciphertext for one message) to another (e.g., several ciphertexts for one message) before calling the Tree-PBS or the ChainPBS algorithms. As Algorithm 27 and Algorithm 28 can perform the same operation as the PBS, one can replace the PBS in [GBA21] algorithms by a WoP-PBS. This relaxes the need to call Tree-PBS or ChainPBS with ciphertexts having a bit of padding. We call these two algorithms respectively the Tree-WoP-PBS and the Chained-WoP-PBS. Note that these algorithms can also be used to implement the $\mathsf{Add}_{2\mathrm{P}}^{\mathrm{MSB}}$ and $\mathsf{Mul}_{2\mathrm{P}}^{\mathrm{MSB}}$ operators (Algorithm 31).

6.2 WoP-PBS: Second Attempt

In Section 6.1, we described two new WoP-PBS (Algorithm 27 and Algorithm 28) based on the GLWE multiplication (Algorithm 20) which can be quite costly to evaluate. In fact, we need to use a large ciphertext modulus to compute it (e.g., $q = 2^{128}$) which is not a native type supported by machines, making the implementation slower than expected. This constraint comes from the very large noise introduced by the multiplication.

Here, we present a new WoP-PBS that does not rely on the GLWE multiplication. We implemented and compared this new technique to prior art using the optimization framework introduced in Chapter 4 and Section 4.3. Later in Section 7.2.4, we provide benchmarks using this new algorithm.

The PBS [Chi+20a; CJP21] takes as input a single LWE ciphertext and outputs the LUT evaluation on the encrypted message. However, when the message is encoded in multiple LWE ciphertexts, a single PBS is not enough. The Tree-PBS method proposed in 2021 by Guimarães, Borin and Aranha [GBA21] and recalled in Algorithm 15 enables to evaluate a large look-up table over many input ciphertexts. For completeness, we will provide details about how to use this technique for our large homomorphic integers in Section 7.2.3. The Tree-PBS is a valid solution for the evaluation of generic LUTs over large integers, but its cost increases exponentially with the number of blocks (i.e., the number of LWE ciphertexts composing a large integer ciphertext). Additionally, the Tree-PBS technique uses the classical PBS [Chi+20a; CJP21], which puts constraints on the bit of padding (Limitation 1) and on the small precision of the messages (Limitation 2).

In this section, we propose an alternative technique to evaluate generic LUTs on large integers, that scales better than the Tree-PBS and does not suffer from the constraint on the bit of padding.

6.2.1 LUT Evaluation over Large Integers

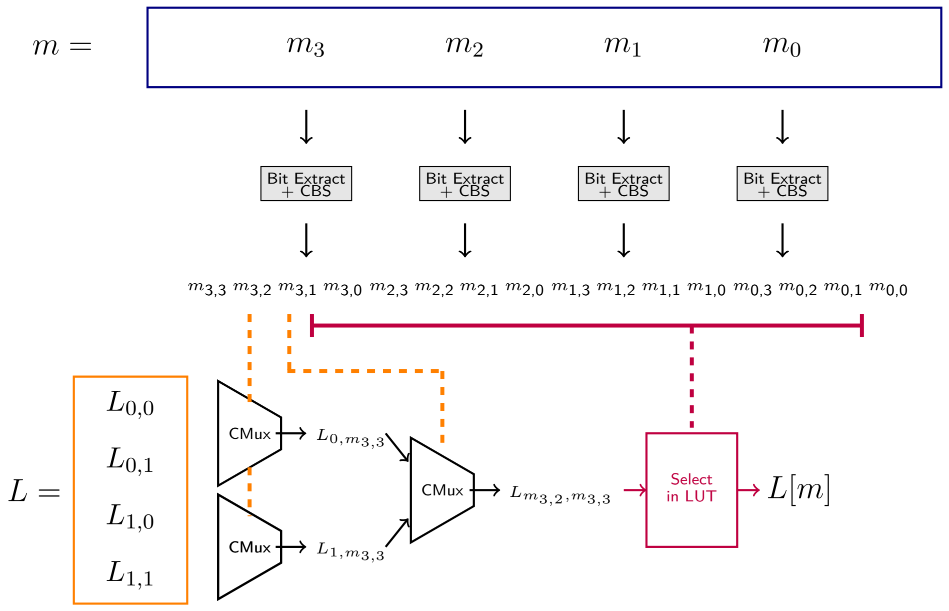


Figure 6.1: Cleartext evaluation of the new WoP-PBS (toy example). The values $m_{i,j}$ (for $i, j \in \{0, 1, 2, 3\}$) are bits. We split the LUT L into 4 smaller LUTs ($L_{0,0}, L_{0,1}, L_{1,0}, L_{1,1}$) to be evaluated in the CMux tree. The output LUT of this tree is given as input to the operation selecting the right output of a LUT (corresponding to the blind rotation). The output $L[m]$ is the element of the LUT L corresponding to the input message m. The Bit Extract blocks correspond to the lines 2 to 11 in Algorithm 30 and the CMux tree followed by a blind rotation corresponds to the vertical packing on line 12 (Algorithm 13).

A WoP-PBS, i.e., a PBS which does not require a bit of padding takes as input an LWE ciphertext with or without bits of padding in the MSB, a public key called bootstrapping key, and a LUT L as the classical PBS. It outputs the homomorphic evaluation of the LUT on the input message, i.e., an LWE encryption of $L[m]$.

Here, we propose a new WoP-PBS that is able to take as input not only one LWE ciphertext but several. It is able to round (or truncate or more) all input messages to a given precision. It can be used to compute several LUT on the same set of inputs at the cost of (about) a single LUT.

Our method is based on two building blocks: the circuit bootstrapping (Algorithm 11) and the mixed (or vertical or horizontal) packing (Algorithm 13). The cleartext representation of the new WoP-PBS is presented in Figure 6.1. In practice, the algorithm executes the following steps:

- It starts by using the generalized PBS (Algorithm 17) to evaluate a (negacyclic) scaled sign function (see Remark 8). We then perform a homomorphic subtraction to extract all the bits of the encrypted large message as we did in Algorithm 19. Each bit is outputted as an LWE ciphertext.
- It converts each of the LWE ciphertexts extracted with the previous step into GGSW ciphertexts, by using circuit bootstraps [Chi+20a].
- It uses the GGSW ciphertexts from the previous step to evaluate the LUT with the mixed (or vertical or horizontal) packing introduced in [Chi+20a] and recalled in Algorithm 13: it consists in practice in a cmux tree (Algorithm 12), followed by a blind rotation (Algorithm 9) and one (or several) sample extract (Algorithm 5).

In general, the circuit bootstrap is the most expensive part of the algorithm (each circuit bootstrap requires several PBSs, each followed by several private key switches, Algorithm 3). Since the number of circuit bootstraps corresponds to the number of bits composing the input message, the technique generally scales linearly in the size of the input message. However, as the input size increases, the mixed packing stops being negligible and becomes as costly as (or even more costly than) the circuit bootstrap part: roughly speaking, this happens when the number of cmuxes in the vertical packing part becomes as big as the number of cmuxes in the PBSs computed inside the circuit bootstraps (e.g., for the parameter sets that we use in our experiments, this happens when the input size is about 28 bits).

We provide the details of the technique (using vertical packing in this case) in Algorithm 30. To evaluate several LUTs, we simply need to repeat the vertical packing for each LUT evaluation.

Theorem 30 (WoP-PBS) *In Algorithm 30, we assume that all the keys are drawn from an uniformly binary distribution. The noise of the output of Algorithm 30 corresponds to the noise of a circuit bootstrap – a PBS, followed by a private functional KS (Theorem 7 and Algorithm 3) – followed by $\sum_{i=0}^{\kappa-1} \delta_i$ cmuxes (Theorem 12 and Algorithm 8).*

The formula can be obtained from the noise formulae of the GenPBS *(Theorem 22) and holds as*

Algorithm 30: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{WoP\text{-}PBS}((\mathsf{ct}_0, \ldots, \mathsf{ct}_{\kappa-1}), \mathsf{PUB}, L)$

```
Context: {
    $x_i$ : as defined in Proof 30 i.e. $\beta_i = 2^{x_i} \cdot \beta'_i$ and $\beta'_i \mod 2 \neq 0$
    $\Delta_i$ : scaling factor for the ciphertext $\mathsf{ct}_i$
    $\delta_i$ : bits occupied by the message in ciphertext $\mathsf{ct}_i$ starting from $\Delta_i$
    $\Omega = 2^{\sum_{i=0}^{\kappa-1} \delta_i}$
    $(\mathfrak{B}_{\mathsf{CB}}, \ell_{\mathsf{CB}})$ : the base and level of the output GGSW
    ciphertexts to the circuit bootstrapping
    $(\varkappa, \vartheta) \in \mathbb{N} \times \mathbb{N}$ defining the modulus switching in the
      generalized PBS (Algorithm 17)
Input: {
    $(\mathsf{ct}_0, \ldots, \mathsf{ct}_{\kappa-1})$ encrypting $\mathsf{msg} = (m_0, \ldots, m_{\kappa-1})$
     with for all $0 \leq i < \kappa$, $\mathsf{Decode}(\mathsf{Decrypt}(\mathsf{ct}_i)) = m_i$
    $\mathsf{PUB}$ : public keys required for the whole algorithm
    $L = [l_0, l_1, \cdots, l_{\Omega-1}]$ : a LUT, s.t. $l_h \in \mathbb{Z}_\omega$
Output: $\mathsf{ct}_{\mathsf{out}}$ encrypting $l_{\mathsf{msg}}$
1  for $i \in [\![0; \kappa - 1]\!]$ do
2      for $j \in [\![0; \delta_i - 2]\!]$ do
           /* Extract from the LSB of the message with generalized PBS
              (Algorithm 17)                                                    */
3          if $j == 0$ then
               /* see Proof 30                                                  */
4              $\epsilon_i \leftarrow \left\lfloor \frac{q \cdot 2^{x_i - 2}}{\beta_i} \right\rceil$
5          $\alpha_{i,j} = \frac{\Delta_i \cdot 2^j}{2}$
6          $L_{i,j} = [-\alpha_{i,j}, \cdots, -\alpha_{i,j}]$
7          $c_i \leftarrow \mathsf{KS\text{-}PBS}\left(\left(\mathsf{ct}_i \cdot 2^{\delta_i - 1 - j}\right) + (0, \cdots, 0, \epsilon_i), \mathsf{PUB}, L_{i,j}, (\varkappa = \log_2(\Delta_i) + j, \vartheta = 0)\right)$
8          $c'_i \leftarrow c_i + (0, \cdots, 0, \alpha_{i,j})$
           /* Subtract the extracted bit from the original ciphertext          */
9          $\mathsf{ct}_i \leftarrow \mathsf{Sub}(\mathsf{ct}_i, c'_i)$
           /* Circuit bootstrap (Algorithm 11) the extracted bit into a GGSW    */
10         $\overline{\overline{C}}_{i,j} \leftarrow \mathsf{CircuitBootstrap}(c'_i, \mathsf{PUB}, (\mathfrak{B}_{\mathsf{CB}}, \ell_{\mathsf{CB}}), (\varkappa = \log_2(\Delta_i) + j, \vartheta = 0))$
       /* Circuit bootstrap (Algorithm 11) the last bit into a GGSW             */
11     $\overline{\overline{C}}_{i,\delta_i - 1} \leftarrow \mathsf{CircuitBootstrap}(\mathsf{ct}_i, \mathsf{PUB}, (\mathfrak{B}_{\mathsf{CB}}, \ell_{\mathsf{CB}}), (\varkappa = \log_2(\Delta_i) + \delta_i - 1, \vartheta = 0))$
   /* Vertical Packing LUT evaluation (Algorithm 13)                            */
12 $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{VPLut}\left(\left\{\overline{\overline{C}}_{i,j}\right\}_{i \in [\![0; \kappa-1]\!]}^{j \in \lceil 0; \delta_i - 1 \rceil}, L\right)$
13 return $\mathsf{ct}_{\mathsf{out}}$
```

$$\mathsf{Var}(E_{\mathsf{WoP\text{-}PBS}}) = \mathsf{Var}(E_{\mathsf{CB}}) + \underbrace{\left(\sum_{i=0}^{\kappa-1}\delta_i\right)\ell_{\mathsf{CB}}(k+1)N\frac{\mathfrak{B}_{\mathsf{CB}}^2+2}{12}\mathsf{Var}(E_{\mathsf{CB}}) + \left(\sum_{i=0}^{\kappa-1}\delta_i\right)\frac{kN}{32}+}_{\textit{mixed packing}}$$

$$\underbrace{+\left(\sum_{i=0}^{\kappa-1}\delta_i\right)\frac{q^2-\mathfrak{B}_{\mathsf{CB}}^{2\ell_{\mathsf{CB}}}}{24\mathfrak{B}_{\mathsf{CB}}^{2\ell_{\mathsf{CB}}}}\left(1+\frac{kN}{2}\right)+\frac{\left(\sum_{i=0}^{\kappa-1}\delta_i\right)}{16}\left(1-\frac{kN}{2}\right)^2}_{\textit{mixed packing}}$$

with

$$\mathsf{Var}(E_{\mathsf{CB}}) = \underbrace{n\ell_{\mathsf{BR}}(k+1)N\frac{\mathfrak{B}_{\mathsf{BR}}^2+2}{12}\mathsf{Var}(\mathsf{BSK}) + n\frac{q^2-\mathfrak{B}_{\mathsf{BR}}^{2\ell_{\mathsf{BR}}}}{24\mathfrak{B}_{\mathsf{BR}}^{2\ell_{\mathsf{BR}}}}\left(1+\frac{kN}{2}\right)+}_{\mathsf{PBS}}$$

$$\underbrace{+\frac{nkN}{32}+\frac{n}{16}\left(1-\frac{kN}{2}\right)^2}_{\mathsf{PBS}} + \underbrace{\ell_{\mathsf{BR}}(n+1)\frac{\mathfrak{B}_{\mathsf{BR}}^2+2}{12}\mathsf{Var}(\mathsf{KSK})+}_{\textit{private functional}\ \mathsf{KS}}$$

$$\underbrace{+\frac{q^2-\mathfrak{B}_{\mathsf{BR}}^{2\ell_{\mathsf{BR}}}}{24\mathfrak{B}_{\mathsf{BR}}^{2\ell_{\mathsf{BR}}}}\left(1+\frac{n}{2}\right)+\frac{n}{32}+\frac{1}{16}\left(1-\frac{n}{2}\right)^2}_{\textit{private functional}\ \mathsf{KS}}.$$

The cost of Algorithm 30 corresponds to the cost of $\sum_{i=0}^{\kappa-1}(\delta_i-1)$ KSs and PBSs (with parameters n, k, N, ℓ_{BR}, $\mathfrak{B}_{\mathsf{BR}}$, ℓ_{KS}, $\mathfrak{B}_{\mathsf{KS}}$, σ_{BSK}, σ_{KSK}), plus the cost of $\sum_{i=0}^{\kappa-1}\delta_i$ circuit bootstrappings (with parameters n, k, N, ℓ_{BR}, $\mathfrak{B}_{\mathsf{BR}}$, ℓ_{CB}, $\mathfrak{B}_{\mathsf{CB}}$, σ_{BSK}, σ_{FPKS}), plus the cost of $\log_2(N)+2^{\sum_{i=0}^{\kappa-1}(\delta_i)-\log_2(N)}-1$ cmuxes (with parameters k, N, ℓ_{CB}, $\mathfrak{B}_{\mathsf{CB}}$).

Proof 30 (Theorem 30) *In what follows, we prove that the output of Algorithm 30 is: $\mathsf{ct}_{\mathsf{out}} = (\mathsf{LWE}_s(l_0(m)), \cdots, \mathsf{LWE}_s(l_{\kappa-1}(m)))$, with $l_i \in \mathbb{Z}_\omega$ a LUT.*

The first step is called the Bit Extract (corresponding to the lines 2 to 8 in Algorithm 30): all the bits of information from all the κ^{th} bits are going to be extracted to be stored in a new block. The first extracted bit is the least significant bit of the message. To do so, it is first shifted to the MSB of the ciphertext. More formally, the first step is to shift the α_i-th MSB to the 1-st MSB by multiplying ct_i by 2^{α_i-1} with $\alpha_i = \lceil\log_2(\beta_i)\rceil$. At this point, the next step would be to compute a PBS with a LUT defined by the polynomial

$P(X) = -\frac{q}{2^{\alpha_i+1}} \cdot \sum_{i=0}^{N-1} X^i$ s.t.:

$$\begin{cases} \frac{q}{2^{\alpha+1}} \textit{ if } \mathsf{Decrypt}(\mathsf{ct}_i \cdot 2^{\alpha_i-1}) \in [\frac{q}{2}, q[\\ \frac{-q}{2^{\alpha+1}} \textit{ otherwise} \end{cases}$$

Then, by homomorphically adding $\frac{q}{2^{\alpha+1}}$, this gives either an encryption of $\frac{q}{2^\alpha}$ or 0. However, observe that for a cleartext equal to 0, any negative noise will lead to a wrong result in the extracted bit (i.e., after decryption, we get $\frac{q}{2^{\alpha+1}}$ instead of $-\frac{q}{2^{\alpha+1}}$). To avoid this type of errors we need to add a small correction integer denoted ϵ_i (for $i \in [0, \kappa - 1]$) to the shifted ciphertext $\mathsf{ct}_i \cdot 2^{\alpha-1}$. In what follows, we compute the value of the corrective term.

We need to be cautious about encoded values that are closed to the two bounds 0 or $\frac{q}{2}$: if the noise is negative, then the PBS will return an incorrect result. In order to choose the right correcting term, we need to determine the smaller distance (denoted $\mathsf{d}(\cdot,\cdot)$) between $\frac{q}{2}$ and its preceding encoded value v_1, and q and its preceding encoded value v_2 i.e.,

$$\min\left(\mathsf{d}\left(v_1 \in \left[0; \frac{q}{2}\right[, \frac{q}{2}\right), \mathsf{d}\left(v_2 \in \left[\frac{q}{2}; q\right[, q\right)\right).$$

We now compute the two distances. At this point, we need to distinct between two cases:

1. *β_i **is a power of two**, i.e., $\beta_i = 2^{x_i}$. The values are encoded by $\left\lfloor \frac{q}{2^{x_i}} \cdot i \right\rceil \mod q$ with $i \in [\![0, 2^{x_i}[\![$. For the shift we compute $\left\lfloor \frac{q}{2^{x_i}} \cdot i \right\rceil \cdot 2^{x_i-1} \mod q$, so the only remaining encoded values are 0 and $\frac{q}{2}$, so the distance between these two values is $d = \frac{q}{2} = \left\lfloor \frac{q \cdot 2^{x_i-1}}{\beta_i} \right\rfloor = 2\epsilon_i$.*

2. *β_i **is not a power of two**, i.e., $\beta_i = \beta_i' \cdot 2^{x_i}$, with some odd $\beta_i' \neq 1$. As $\beta_i \ll q$, after the first shift, for all $j \in \mathbb{N}$ we obtain the following bound over the encoded values:*

$$\begin{aligned} \left\lfloor \frac{q}{\beta_i} \cdot (j \bmod \beta_i) \right\rceil \cdot 2^{\alpha_i-1} \mod q &\leq \left(\frac{q}{\beta_i} \cdot (j \bmod \beta_i) + \frac{1}{2}\right) \cdot 2^{\alpha_i-1} \mod q \\ &\leq \frac{q}{\beta_i} \cdot (j \bmod \beta_i) \cdot 2^{\alpha_i-1} + 2^{\alpha_i-2} \mod q \end{aligned}$$

We now want to work with β'_i instead of β_i. Then, for all $j \in \mathbb{N}, \exists j', j'' \in \mathbb{N}$, such that:

$$\begin{aligned}\frac{q}{\beta_i} \cdot (j \bmod \beta_i) \cdot 2^{\alpha_i - 1} \mod q &= \frac{q}{\beta'_i} \cdot (j' \bmod \beta'_i) \cdot 2^{\alpha'_i - 1} \mod q. \\ &= \frac{q}{\beta'_i} \cdot (j'' \bmod \beta'_i) \mod q.\end{aligned}$$

The next is step is to compute the minimum of the distances:

$$\min \left(\mathsf{d} \left(v_1 \in \left[0; \frac{q}{2}\right[, \frac{q}{2} \right), \mathsf{d} \left(v_2 \in \left[\frac{q}{2}; q\right[, q \right) \right) - 2^{\alpha_i - 2}, v_i \in \left\{ \frac{q}{\beta'_i} \cdot j' \mod q \right\}_{j' \in \{0, \beta'_i - 1\}}.$$

v_1 is bounded by

$$v_1 \leq \left\lceil \frac{q}{\beta'_i} \cdot \left\lfloor \frac{\beta'_i}{2} \right\rfloor \right\rceil = \frac{q}{2} - \left\lfloor \frac{q}{2\beta'_i} \right\rfloor$$

So we have $d_1 = \mathsf{d}\left(v_1 \in [0; \frac{q}{2}[, \frac{q}{2}\right) \geq \left\lfloor \frac{q}{2\beta'_i} \right\rfloor$. Next, v_2 is bounded by

$$v_2 \leq \left\lceil \frac{q}{\beta'_i} \cdot (\beta'_i - 1) \right\rceil = q - \left\lfloor \frac{q}{\beta'_i} \right\rfloor$$

So we have $d_2 = \mathsf{d}\left(v_2 \in [\frac{q}{2}; q[, q\right) > \left\lfloor \frac{q}{\beta'_i} \right\rfloor \geq \left\lfloor \frac{q}{2\beta'_i} \right\rfloor$. The distance is then bounded by $\left\lfloor \frac{q}{2\beta'_i} \right\rfloor - 2^{\alpha_i - 2}$. The correcting term is finally defined as half of this bound, i.e., $\epsilon_i = \left\lfloor \frac{q}{4\beta'_i} \right\rfloor - 2^{\alpha_i - 3} = \left\lfloor \frac{q \cdot 2^{x_i - 2}}{\beta_i} \right\rfloor - 2^{\alpha_i - 3}$.

Since the term $2^{\alpha_i - 3}$ in Proof 30 is very small regarding q, it can be neglected to have the same ϵ in the both cases. About the noise bound, this term is also negligible, since it is smaller than 1 before the shift.

By taking $\epsilon_i = \left\lfloor \frac{q \cdot 2^{x_i - 2}}{\beta_i} \right\rfloor$ and adding ϵ_i to $\mathsf{ct}_i \cdot 2^{\alpha - 1}$ we ensure that for any message, an error e of size $|e| < \epsilon_i$ will lead to a correct PBS evaluation. This means that before the shift, the noise in ct_i should be smaller than $\left\lfloor \frac{q \cdot 2^{x_i - 2}}{\beta_i} \right\rfloor \cdot 2^{-\lfloor \log_2(\beta_i) \rfloor}$.

At this point, the less significant bit (the α-th bit) has been extracted and stored into a new $\mathsf{LWE}_{i,\alpha}$. To extract the next bit, we first subtract $\mathsf{LWE}_{i,\alpha}$ to ct_i. With this operation we ensure that the α-th bit is now equal to 0. As we want to extract the $(\alpha - 1)$-th bit, we now shift by $2^{\alpha - 2}$. Finding the corrective term ϵ_i is much easier in this case, as the second bit is equal to 0 after the shift. Hence, we can take $\epsilon = \frac{q}{4}$ and extract the bit with a PBS.

To extract the remaining bits, we just need to repeat the previous steps (subtraction, shift, add $\epsilon = \frac{q}{4}$ and PBS).

Concerning the correctness of circuit bootstrap and vertical packing, we refer to [Chi+20a].

□

Several optimizations are possible in Algorithm 30. We did not include them directly in Algorithm 30 to simplify the explanation:

- Observe that the base and level used in the PBS for bit extraction and in the PBS for circuit bootstrapping might be chosen differently.
- The PBSs in the first step of the algorithm can either be computed independently, or sequentially, from LSB to MSB, by removing an extracted bit from the input ciphertext before extracting the next one.
- The second step of the circuit bootstrapping, which is a series of several packing private key switchings, can be improved by following a similar footstep as a technique proposed in [CCR19]. We perform an initial LWE-to-GLWE KS (not functional) to each of the outputs of the PBS, and then, as already done in [CCR19], we perform an external product with the GGSW encryption of the GLWE secret key to obtain the remaining GLWE ciphertexts. This allows us to reduce the size of public evaluation keys at the cost of a slightly larger noise in the output.
- The KS-PBS performed in Line 7 is a Generalized PBS (Algorithm 17), so the modulus switch directly reads the next bit to be extracted. The sign function is evaluated in order to re-scale the bit at the right scaling factor. The circuit bootstraps used in Lines 10 and 11 are also instantiated with a Generalized PBS. If we chose a value of $\vartheta > 0$ we could improve the circuit bootstraps with a PBSmanyLUT (Algorithm 18), i.e., perform all the PBSs in a circuit bootstrap at the cost of a single PBS. Using this technique imposes an additional constraint on the noise in the input of the circuit bootstrap.
- We can observe that one of the PBSs of the circuit bootstraps used in Line 10 could be avoided thanks to the KS-PBS in Line 7, that might already provide the bit extracted at the right re-scaling factor.

Remark 34 (Carry Buffer and Bit Extract) *In general, the number of circuit bootstraps performed in Algorithm 30 corresponds to the number of bits of the input message. However, this number might be slightly larger in some special cases, such as the case where the carry buffers have not been emptied beforehand, or the case of native CRT (Definition 16). In these cases, we might need to extract more bits of information, and so perform more PBSs during bit extraction and more circuit bootstraps. Furthermore, different possible inputs might encode the same value, hence the LUT L needs to contain some kind of redundancy. If the goal is to compute the discrete function f, one needs to compute the LUT L as $L[(m_0, \cdots, m_{\kappa-1})] = \mathsf{Encode}\left(f\left(\mathsf{Decode}\left(m_0, \cdots, m_{\kappa-1}\right)\right)\right)$.*

Remark 35 (Faster Algorithm 30 for Special LUTs) *Observe that the new WoP-PBS approach can be also adapted, and be very convenient, for particular LUTs such as the ReLU and the sign functions in the radix mode. Indeed, for these functions we are only interested in the MSB part of the message. We can leave out some of the least significant bits and only do the Vertical Packing with a subset of the extracted bits which would be faster.*

6.2.2 Comparison Between $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$, $\mathcal{A}^{(\mathbf{CJP21})}$ and $\mathcal{A}^{(\mathbf{GBA21})}$

In Algorithm 30, we introduced a new WoP-PBS. We can now resume our comparison, started in Section 4.3.1, to find out which algorithm is the best (depending on some parameters) to compute over ciphertexts with large precision. To do so, we consider a new atomic pattern type (see Definition 28) $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$.

Definition 34 ($\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ Atomic Pattern Type) *We define a new atomic pattern type $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ as a subgraph composed of a dot product (Theorem 5) and the WoP-PBS defined in Algorithm 30.*

As before, we assume that the inputs of the dot product are outputs of a bootstrap. An atomic pattern of type $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ can be fully described by the pair (ν, t) with ν, the 2-norm and t the noise bound (Definition 21).

As this algorithm can work on a single ciphertext or on several ciphertexts containing chunks of the message, we present three variants: 1, 2 and 4 blocks. We display a comparison between $\mathcal{A}^{(\mathrm{CJP21})}$, $\mathcal{A}^{(\mathrm{GBA21})}$ and $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ on Figure 6.2. We used the exact same context as in Figure 4.3 for this experiment, so the failure probability is for the three of them $p_{\mathsf{fail}} \approx 2^{-35}$.

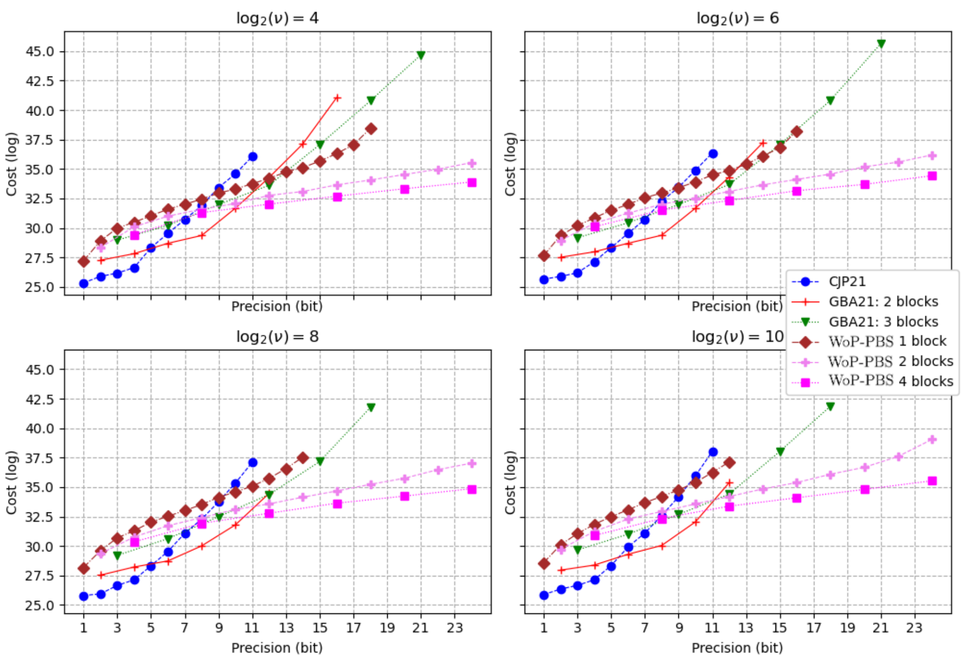


Figure 6.2: In this figure, we evaluate a LUT over a few encrypted inputs. We compare the AP type $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$, corresponding to the WoP-PBS introduced in Algorithm 30 (1, 2 and 4 blocks), with the AP type $\mathcal{A}^{(\mathrm{GBA21})}$, corresponding to the Tree-PBS [GBA21] (2 and 3 blocks). As a baseline, the AP of type $\mathcal{A}^{(\mathrm{CJP21})}$ is also plotted.

Remark 36 (Noise Bound) *For $\mathcal{A}^{(CJP21)}$ and $\mathcal{A}^{(GBA21)}$, please refer to Remark 27. For $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$, we have a certain number of sequential bit extractions per input LWE ciphertext and per block. In theory, we want to take into account all those potential PBSs (one per bit extraction), but we noticed that the first one dominates all the others regarding the noise. In fact, their impact on the total failure probability is negligible compared with the first bit extraction. Our experiments showed that for 2-norms $\nu \geq 4$, and for failure probability below 2^{-25} this assumption holds. We leave to future works the exploration of this topic. With this assumption, we start by computing the failure probability needed for one PBS defined as $p_i' = 1 - (1 - p_{\mathsf{fail}})^{\frac{1}{\kappa}}$ with κ the number of input LWE ciphertexts. From it, we can finally compute the noise bound for each PBS $t\,(p, 0) = \frac{q}{2^{1 \cdot p \cdot z^*}\left(p_i'\right)}$.*

The brown/♦ curve represents the cost of the best parameter set for an atomic pattern $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ working over one block. We can immediately notice that, between 1 and 9 bits of precision, $\mathcal{A}^{(\mathrm{CJP21})}$ is more interesting than the new bootstrap (Algorithm 30). However, with precisions from 10 bits and above, $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ has solutions that are more efficient than the ones existing for $\mathcal{A}^{(\mathrm{CJP21})}$, and finds solutions when there is none for $\mathcal{A}^{(\mathrm{CJP21})}$.

For a small ν, it offers solutions that are slightly better than the ones from $\mathcal{A}^{(\mathrm{GBA21})}$.

The pink/✚ curve (respectively the pink/■ curve) represents the atomic pattern $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ for two blocks (respectively four blocks) of message. On those curves, we see that it scales much better with the precision than the other atomic pattern types. With Algorithm 30, we manage to find solutions up to 24 bits of precision. Those solutions are costly but far less than the ones for $\mathcal{A}^{(\mathrm{GBA21})}$, and for comparison, it is only 2^{10} times more costly to compute a LUT over a message with 24-bits of precision with $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ than to compute a LUT with 1 bit of precision with $\mathcal{A}^{(\mathrm{CJP21})}$. For comparison, in Section 4.3 and Figure 4.3, we saw that computing $\mathcal{A}^{(\mathrm{GBA21})}$ over a message of 21 bits costs more than 2^{20} times the cost of a PBS over Boolean messages. Finally, for 18 bits of precision, the new WoP-PBS with two blocks is approximately 2^7 times faster than the tree-PBS in $\mathcal{A}^{(\mathrm{GBA21})}$ with three blocks.

To sum up, for small precisions (up to 5 bits), TFHE's PBS is the best option among the three considered. Above 10/11 bits of precision, the algorithm we introduced in this thesis (Algorithm 30) becomes the best alternative and improves the state of the art by a non-negligible factor.

Remark 37 (LUT Evaluation for Even More Precision) *It is important to observe that evaluating a LUT on integers larger than e.g., 30 bits, even in cleartext, becomes too expensive in terms of memory. For instance, a LUT for 30-bit input and output integers contains* $2^{30} \cdot 64$ *bits* $= 8$ *GB of information. So both techniques – Tree-PBS and our new WoP-PBS – are in any case not practical anymore.*

Remark 38 (Small Public Key Material for Algorithm 30) *In Algorithm 30, the size of the needed public material scales way better than a tree-PBS as in [GBA21]. As an example, for a total of 18 bits of precision we have a key of* 1.65 *GB for* $\mathcal{A}^{(GBA21)}$ *and a size of* 0.926 *GB for* $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$.

6.2.3 Comparison Between $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ and $\mathcal{A}^{(\mathbf{LMP}21)}$

Since the publication of the first WoP-PBS (presented in Section 6.1, in Algorithm 27 and in Algorithm 28), a few WoP-PBS constructions have been proposed in the literature. Some works [KS21; LMP21] already compare them somehow, but our optimization framework enables to truly do so by comparing them at the best of their efficiency. This can be done by putting each of them in a different atomic pattern type and finding optimal parameters for different 2-norms and precisions. To do so, we create one additional atomic pattern type called $\mathcal{A}^{(\mathrm{LMP21})}$ composed of a DP, a KS and the WoP-PBS from [LMP21]. We use the exact same context as in Figure 4.3 for this experiment, so the failure probability is for the both of them $p_{\mathsf{fail}} \approx 2^{-35}$. We display in Figure 6.3 the comparison between our new WoP-PBS (Algorithm 30, blue/● curve) in $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ and the WoP-PBS from [LMP21] in $\mathcal{A}^{(\mathrm{LMP21})}$ (red/+ curve).

Remark 39 (Noise Bound of $\mathcal{A}^{(\mathbf{LMP21})}$) *For $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$, please refer to Remark 36. For $\mathcal{A}^{(LMP21)}$, we consider the two sequential PBSs involved in the algorithms. They almost have the same amount of input noise and thus we assume that they both contribute equally to the overall failure probability. We experimented with the two possible scenarios, (i) taking the input noise of the first PBS for the computation or (ii) taking the second one. We did not observe any difference between the two approaches for the considered failure probabilities and 2-norms.*

We start by computing the failure probability needed for one PBS defined as $p_i' = 1-(1-p_{\mathsf{fail}})^{\frac{1}{2}}$ and from it we can finally compute the noise bound for each PBS $t\,(p,0) = \frac{q}{2^{1+1}\cdot p\cdot z^*\left(p_i'\right)}$.

The first thing that we learn about the WoP-PBS of [LMP21] is that it does not scale well with big precisions, which is not surprising as the algorithm uses as subroutine two PBSs from [Chi+20a] to compute the WoP-PBS over the entire input message. Thus, as for $\mathcal{A}^{(\mathrm{CJP21})}$, for precisions above 10, we do not find feasible solutions. We can also identify as before two parts on the curve, the first one for small precisions (1-5 bits) and a second one for higher precisions: the reason behind this sudden growth in cost is due to the increase of the polynomial size to manage bigger messages.

Thanks to the new WoP-PBS (Algorithm 30), we are able to compute a WoP-PBS over large messages. To conclude, this new algorithm scales better than existing algorithms to compute LUTs over large messages and we do not need a padding bit which is a known constraint of TFHE's bootstrapping (Limitation 1).

6.2.4 Failure Probability Analysis

In this section we analyze the impact of decreasing the failure probability on the cost, for the atomic patterns $\mathcal{A}^{(\mathrm{CJP21})}$ and $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$. We consider four different failure probabilities: $p_{\mathsf{fail}} \in \{2^{-14}, 2^{-20}, 2^{-35}, 2^{-50}\}$ and to simplify the analysis, we fix the 2-norm $\nu = 2^4$ since the behaviour is pretty similar for other 2-norms.

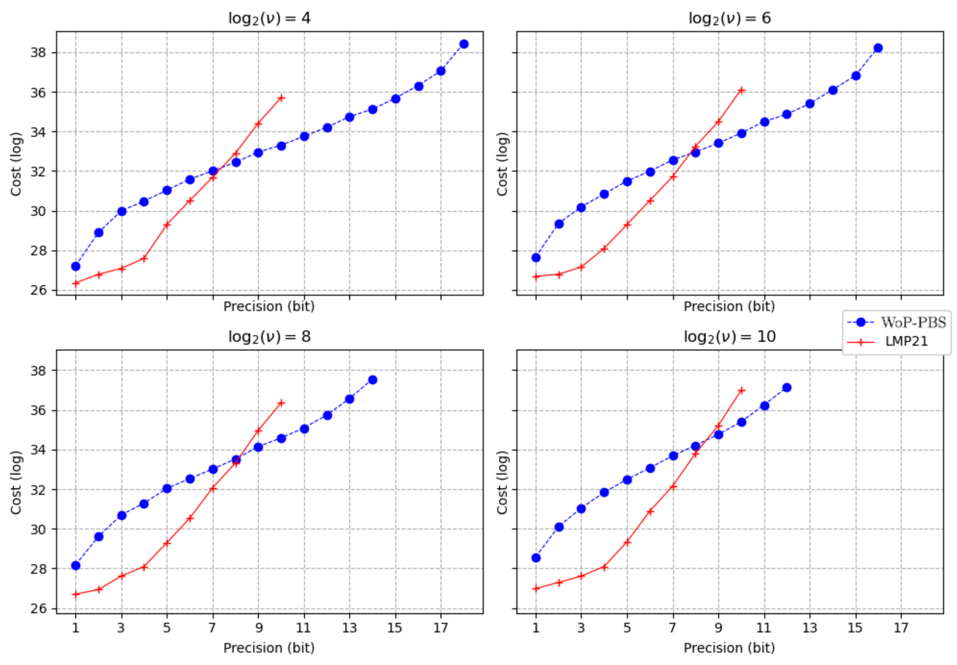


Figure 6.3: In this figure, we compare the cost of the AP types $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ and $\mathcal{A}^{(\mathrm{LMP21})}$. The first one corresponds to DP followed by our new WoP-PBS (Algorithm 30), and the second one to DP-KS followed by the WoP-PBS from [LMP21].

Figure 6.4 is dedicated to the AP of type $\mathcal{A}^{(\mathrm{CJP21})}$. We plot the cost for precisions between 1 and 12 bits. As expected, we can observe that if we decrease the failure probability, the cost increases. Roughly speaking, starting from 4-5 bits of precision, for every additional bit, N has to be twice as big, which more than doubles the cost of the atomic pattern. Observe that the cost is very close for certain curves: for instance, the brown/◆ curve, corresponding to $p_{\mathsf{fail}} = 2^{-50}$, and the green/▼ curve, corresponding to $p_{\mathsf{fail}} = 2^{-35}$, have almost the same cost. The red/+ curve, corresponding to $p_{\mathsf{fail}} = 2^{-20}$, has a cost that is close to the one of the blue/● curve, corresponding to $p_{\mathsf{fail}} = 2^{-14}$, up to 7 bits of precision, and starting from 8 bits of precision it gets closer to the green/▼ curve. This change is due to the fact that there are no parameter sets fulfilling the requirements with a bigger N only twice as big, it has to be 4 times bigger.

Figure 6.5 is dedicated to the AP of type $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$. The brown/◆ curve, corresponds to $p_{\mathsf{fail}} = 2^{-50}$, the green/▼ curve, corresponds to $p_{\mathsf{fail}} = 2^{-35}$, the red/+ curve corresponds to $p_{\mathsf{fail}} = 2^{-20}$, and finally the blue/● curve corresponds to $p_{\mathsf{fail}} = 2^{-14}$. All the curves follow the same behaviour and are simply shifted up when the failure probability is decreased.

To sum up, with the AP of type $\mathcal{A}^{(\mathrm{CJP21})}$, for each additional bit of precision, the

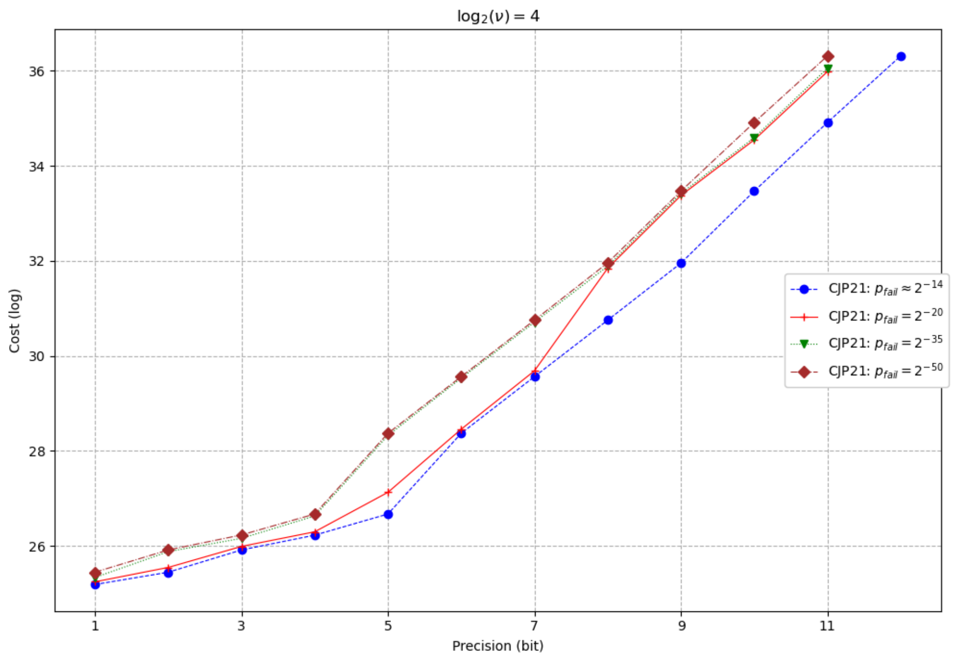


Figure 6.4: Cost comparison for the same AP of type $\mathcal{A}^{(\text{CJP21})}$, with respect to the following failure probabilities: $p_{\mathsf{fail}} \in \{2^{-14}, 2^{-20}, 2^{-35}, 2^{-50}\}$.

overall cost of the AP is doubled. However, this is not the case with the AP of type $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$. The behaviour of the curve in this region (precision below 24 bits) looks more like a linear one. For probabilities $p_{\mathsf{fail}} = 2^{-35}$ and $p_{\mathsf{fail}} = 2^{-50}$, the curves are almost overlapping. If we look for instance at 7 bits of precision, the polynomial size N are the same, but the noise added by the key switch is slightly lower thanks to a bigger n (output of the key switch) and to other small changes in the key switch decomposition parameters. Indeed, at this precision, N is already quite big so it can handle the message precision. Thus the noise of the modulus switch is not the most constraining one, the one from the key switch actually is. This explains the small overhead in this context.

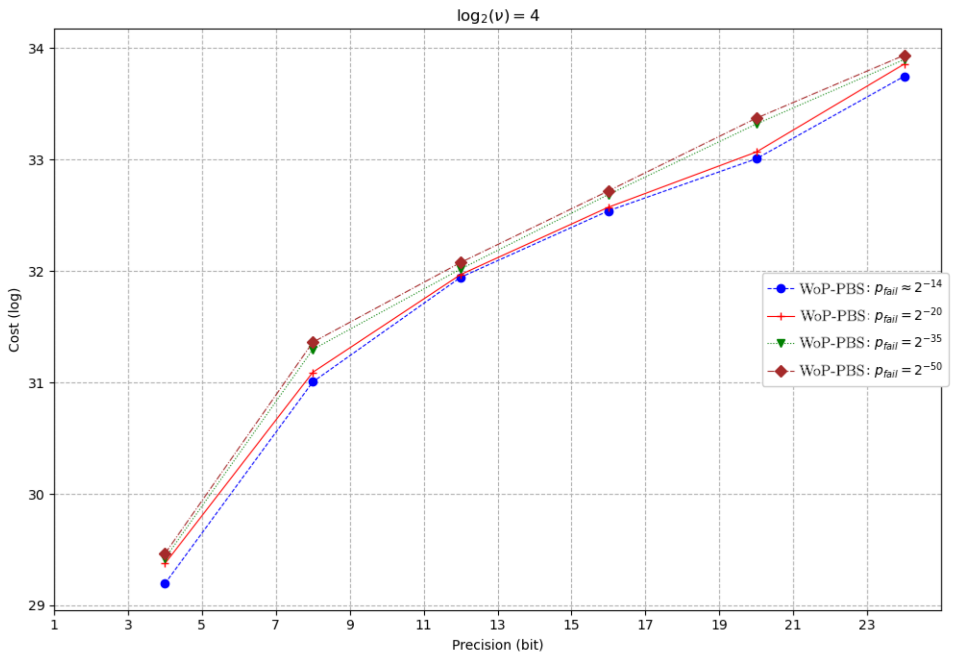


Figure 6.5: Cost comparison for the same AP of type $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$, with respect to the failure probabilities $p_{\mathsf{fail}} \in \{2^{-14}, 2^{-20}, 2^{-35}, 2^{-50}\}$.

Chapter 7

Homomorphic Integers

In this chapter, we combine the new algorithms introduced in Chapters 5 and 6 with those from the literature to explain how to build efficient integer arithmetics with TFHE.

First, we explain how to leverage the LWE multiplication (Algorithm 22), the generalized PBS (Algorithm 17) and the first two WoP-PBSs (Algorithms 27 and 28) to build a fast Boolean arithmetic. TFHE's Boolean approach consists in computing a PBS for every Boolean gate in a Boolean circuit to correct the encoding and to reduce the noise. Thanks to our new algorithms, we are able to execute Boolean gates without having to perform a PBS every time. We perform a PBS only when it is necessary to reduce the noise. We then extend this new Boolean arithmetic to build a modular arithmetic working with power-of-two message moduli. Finally, we describe how to tweak it to build an efficient integer arithmetic that supports the opposite, the addition, the multiplication and arbitrary lookup tables. This approach is particularly efficient for small messages (less than 8 bits of precision).

Finally, we explain how to implement an integer arithmetic for high precision integers. To do so, we leverage the WoP-PBS described in Algorithm 30. In Limitation 11, we explained that there was no existing solution to support arbitrary message moduli with the radix encoding. We introduce a *generalization of this encoding* that overcomes this limitation. In Limitation 12, we explained that there is a limited number of co-prime bases that are smaller than 2^8, the maximal message modulus to have efficient lookup table evaluations with the PBS. We introduce a new *hybrid* type of encoding that mixes the radix and the CRT encodings to overcome this limitation. We also show how to apply arbitrary functions to a message encoded with our new encodings, thus overcoming Limitation 13. We explain how to do it with the tree-PBS introduced in [GBA21] (recalled in Algorithm 15) and with our WoP-PBS (Algorithm 30). We conclude by giving extensive benchmarks of the main operations (addition, multiplication, LUT evaluation) on high-precision messages with radix, CRT and hybrid encodings using parameters given by our optimizer framework from Chapter 4.

7.1 Small Integers

As studied in Section 4.2 and Definition 30, we can easily work over small integers using a dot product, a key switch and a PBS. With this method, we can compute additions, multiplications by plaintext integers and LUT evaluations. A big restriction comes with the PBS (Limitation 1): we need the padding bit to be known. This restriction is particularly annoying if we only care about modular arithmetic. In fact, as every operation is done modulo q, we may want to leverage this modular reduction to only keep the least significant bit(s) after each computation. If we were to do that using a regular PBS (Algorithm 17), we would have issues because to use this modular reduction we need to fill the padding bit. A solution is to use the WoP-PBS introduced in this work (Algorithm 27, Algorithm 28 or Algorithm 30).

We start by describing an improvement of FHE Boolean circuit evaluation. Then, we extend it to arithmetic circuits working with integers encoded in more than a single bit. Finally, we describe how to use the later to build exact computations on larger encrypted integers.

7.1.1 Boolean Arithmetic

In TFHE [Chi+16a], authors improve techniques proposed in FHEW [DM15] to perform fast homomorphic evaluations of Boolean circuits and called this feature *gate bootstrapping*. It is very easy to use and implement, because it performs one bootstrap for each bivariate Boolean gate evaluated: there is no need to be careful with the noise management anymore because each gate resets the noise systematically. Furthermore, the same parameter set can be used for all the bootstraps in the circuit. These also make the conversion between the cleartext Boolean circuits and the encrypted circuits quite straightforward in practice.

However, performing a bootstrap at each bivariate Boolean gate is very expensive when we want to evaluate large circuits and seems unnecessary. One idea to make the evaluation more efficient is to mix the bootstrap with some leveled operations, at the cost of now caring about noise growth. But this idea cannot be immediately applied when it comes to gate bootstraps: in fact, the bootstrap also takes care of ensuring a fixed encoding in the ciphertexts, that may not be ensured if we introduce leveled operations (additions, subtractions or multiplications by an integer). Furthermore, TFHE can only evaluate linear combinations between LWE ciphertexts; non linear operations would require the use of a PBS or of a circuit bootstrap (Algorithm 11) followed by an external product. This is especially problematic when we want to evaluate an AND gate, for instance.

To be more clear, when gate bootstrapping, messages are encoded with what we call one *bit of carry*: meaning that we know that the MSB of the plaintext (without noise) is set to zero. This bit is used to perform a linear combination while preserving the (plaintext) MSB of this combination so we can bootstrap it (the function is negacyclic, so does not need a bit of padding) and get a correct result. Roughly speaking, the initial linear combination evaluates the linear part of the gate and potentially fills the bit of carry, while the bootstrap takes care of the evaluation of the non-linear part of the gate, reduces the noise and empties the bit of carry to be able to perform a future operation.

We propose a novel approach based on the GenPBS (Algorithm 17) and $\mathsf{LWEMult}$ (Algorithm 22), which removes both the constraint of padding bits and the difficulties with the non-linear leveled evaluations. Thus, this offers the possibility of computing series of Boolean gates without the need for computing a bootstrap at every gate. A GenPBS should only be computed to reduce the noise when needed. In Lemma 2, we only describe some of the most common Boolean gates (i.e., $\mathsf{XOR}, \mathsf{NOT}$ and AND), whose combination offers functional completeness. The other gates can be obtained by combining these operations.

Lemma 2 (New Boolean Gates) *Let* $b_i \in \{0,1\}$ *such that* $\mathsf{ct}_i = \mathsf{LWE}_{\vec{s}}\left(b_i \cdot \frac{q}{2}\right) \in \mathbb{Z}_q^{n+1}$*, for* $i \in \{1,2\}$*. Let* $\left(\vec{0}, \frac{q}{2}\right) \in \mathbb{Z}_q^{n+1}$ *be a trivial LWE encryption of* 1*. Then, the following equalities between Boolean gates and homomorphic operators hold:*

$$\begin{aligned}\mathsf{ct}_1\,\mathsf{XOR}\,\mathsf{ct}_2 &:= \mathsf{ct}_1 + \mathsf{ct}_2\\ \mathsf{ct}_1\,\mathsf{AND}\,\mathsf{ct}_2 &:= \mathsf{LWEMult}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{RLK}, \mathsf{KSK})\\ \mathsf{NOT}\,\mathsf{ct}_1 &:= \mathsf{ct}_1 + \left(\vec{0}, \frac{q}{2}\right)\end{aligned}$$

Proof (Lemma 2) *A bit is naturally encoded as a* 0 *(resp.* $\frac{q}{2}$*) if its value is* 0 *(resp.* 1*). Then the Boolean gates* XOR *and* NOT *stem from that encoding. The* AND *is a direct* *application of the* $\mathsf{LWEMult}$ *operator.*

□

The noise increases after each computed gate since no bootstrap is performed. Therefore, after chaining a number of them, a noise reduction may be required. We propose two simple processes exploiting the GenPBS with the (negacyclic) sign function.

Lemma 3 (Boolean Bootstrap) *Let* $\mathsf{ct_{in}}$ *be an LWE ciphertext resulting from a Boolean circuit with gates defined as in Lemma 2. We can bootstrap* $\mathsf{ct_{in}}$ *during the Boolean circuit evaluation with one of the following operators:*

$$\mathsf{ct_{out}} \leftarrow \mathsf{GenPBS}\left(\mathsf{ct_{in}}, \mathsf{BSK}, P_{\mathbb{1}} \cdot X^{N/2}, \Delta_{\mathsf{out}} = \frac{q}{4}, \varkappa = 0, \vartheta = 0\right) + \left(\vec{0}, \frac{q}{4}\right) \tag{7.1}$$

$$\mathsf{ct_{out}} \leftarrow \mathsf{GenPBS}\left(\mathsf{ct_{in}}, \mathsf{BSK}, P_f = \sum_{i=\frac{N}{4}}^{\frac{3N}{4}-1} X^i, \Delta_{\mathsf{out}} = \frac{q}{2}, \varkappa = -1, \vartheta = 0\right) \tag{7.2}$$

Proof (Lemma 3) *The first method 7.1 uses* GenPBS *with the parameters* $\Delta_{\mathsf{out}} = \frac{q}{4}, \varkappa = 0, \vartheta = 0$ *and* $P_f = P_{\mathbb{1}} \cdot X^{N/2}$*. The output of the* GenPBS *gives* $\mathsf{ct}_{\mathrm{tmp}} = \mathsf{LWE}_{\vec{s}}(\pm\frac{q}{4})$*. Therefore, depending on the sign, the term* $\mathsf{ct}_{\mathrm{tmp}} + (\vec{0}, \frac{q}{4})$ *is equal to* $\mathsf{LWE}_{\vec{s}}(0)$ *or* $\mathsf{ct}_{\mathrm{tmp}} = \mathsf{LWE}_{\vec{s}}(\frac{q}{2})$*.*

The second approach 7.2 uses other parameters for the modulus switch which can be seen as shifted by one bit, i.e., $\varkappa = -1$*,* $\vartheta = 0$ *and* $\Delta_{\mathsf{out}} = \frac{q}{2}$*. In this case, the sign does not impact the value of the encoded bit, since* $\pm 0 = 0$ *and* $\pm\frac{q}{2} = \frac{q}{2}$*. Then, evaluating* GenPBS *with the function* $P_f = \sum_{i=\frac{N}{4}}^{\frac{3N}{4}-1} X^i$ *and* $\Delta_{\mathsf{out}} = \frac{q}{2}$*, we obtain* $\mathsf{ct_{out}} = \mathsf{LWE}_{\vec{s}}(\pm 0)$ *or* $\mathsf{LWE}_{\vec{s}}\left(\pm\frac{q}{2}\right)$*.*

□

7.1.2 Arithmetic Modulo a Power of 2

Here, we generalize the Boolean circuit approach described in Lemma 2 to arithmetic circuits modulo any power of two. This enables a more efficient exact arithmetic modulo 2^p for some integer p. For $i \in \{1, 2\}$, let $\mathsf{ct}_i = \mathsf{LWE}_{\vec{s}}(m_i \cdot \frac{q}{2^p})$ be an LWE ciphertext encrypting the message $m_i \in [\![0, 2^p[\![$ (i.e., m_i has a precision of p bits). As we did for the Boolean arithmetic, we define three natural homomorphic operators to mimic modulo 2^p arithmetic: the addition (Add_{2^p}) which is evaluated as an homomorphic LWE addition, the multiplication (Mul_{2^p}) which is evaluated as an $\mathsf{LWEMult}$, and the unary opposite (Opp_{2^p}) which is obtained by simply negating the LWE input.

When we deal with integers encoded with more than one bit, the functions we have to apply during a PBS are no longer negacyclic. It means that without a WoP-PBS we would have to have at least 2 bits of padding (one for a linear combination and another one for the PBS with a non-negacyclic function). To ensure correctness during a PBS when we work with larger powers of two, we need a larger polynomial size N which will result in a slower execution. With a WoP-PBS, we do not need to have bits of padding. We can simply compute leveled additions and multiplications, and only use a WoP-PBS when we have to reset the noise to a lower level or to do a LUT evaluation. In the following, to perform a WoP-PBS, we use Algorithm 27 or Algorithm 28 but it can easily be adapted to use Algorithm 30.

7.1.3 From Modular Arithmetic to Exact Integer Arithmetic

Here, we use the radix encoding i.e., several LWE ciphertexts to represent a single large integer as described in Section 2.4 and Definition 15. We now present some operators that extend homomorphic computations modulo a power of two to a larger integer arithmetic. The basic operations introduced in Section 7.1.2 offer the possibility to compute an exact integer multiplication between two LWE ciphertexts and to keep the LSB of the computation. However, we also need to be able to recover the MSB of additions and multiplications for carry propagation when we deal with large integers encrypted as several ciphertexts. The operators keeping the MSB of the computation between two messages $m_1, m_2 \in [\![0, 2^p[\![$ are defined as

$$\begin{cases} \mathsf{Add}_{2^p}^{\mathrm{MSB}} : (m_1, m_2) \mapsto \left\lfloor \frac{m_1+m_2}{2^p} \right\rfloor \mod 2^p \\ \mathsf{Mul}_{2^p}^{\mathrm{MSB}} : (m_1, m_2) \mapsto \left\lfloor \frac{m_1 \cdot m_2}{2^p} \right\rfloor \mod 2^p \end{cases}$$

Their implementation is described in Algorithm 31.

In Algorithm 31, to improve efficiency, we remove both key switches and include them in the relinearization steps of the previous WoP-PBS. If the parameters allow it, one might also replace Lines 6 and 7 of Algorithm 31 by a single WoP-PBS to extract the MSB directly.

Lemma 4 (MSB Operations) *For $i \in \{1, 2\}$, let $\mathsf{ct}_i \in \mathsf{LWE}_{\vec{s}}(m_i \cdot \Delta)$ be two LWE ciphertexts, encrypting $m_i \cdot \Delta$ with $0 \leq m_i < 2^p$ and $\Delta = \frac{q}{2^p}$, both encrypted under the*

```
Algorithm 31: $\mathsf{ct}_{\mathsf{out}} \leftarrow \boxed{\mathsf{Add}^{\mathrm{MSB}}_{2\mathrm{p}}}\ \boxed{\mathsf{Mul}^{\mathrm{MSB}}_{2\mathrm{p}}}\ (\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{BSK}, \mathsf{KSK}_1, \mathsf{KSK}_2, \mathsf{RLK})$

Context: $\begin{cases} \vec{s} = (s_1, \cdots, s_n) \in \mathbb{Z}_q^n \\ \vec{s'} = (s'_1, \cdots, s'_{kN}) \in \mathbb{Z}_q^{kN} \\ \vec{S'} = \left(S'^{(1)}, \ldots, S'^{(k)}\right) \in \mathfrak{R}_q^k \\ \forall 1 \le i \le k,\ S'^{(i)} = \sum_{j=0}^{N-1} s'_{(i-1)\cdot N+j+1} X^j \in \mathfrak{R}_q \\ \Delta = \frac{q}{2^p} \in \mathbb{Z}_q \\ 0 \le m_1, m_2 < 2^p \\ P_{\mathsf{Id}} : \text{a redundant LUT for } x \mapsto x \text{ (identity function)} \end{cases}$
Input:
    $\begin{cases} \mathsf{ct}_1 = \mathsf{LWE}_{\vec{s}}(m_1 \cdot \Delta) \in \mathbb{Z}_q^{n+1} \\ \mathsf{ct}_2 = \mathsf{LWE}_{\vec{s}}(m_2 \cdot \Delta) \in \mathbb{Z}_q^{n+1} \\ \mathsf{BSK} = \left\{\mathsf{BSK}_i = \mathsf{GGSW}_{\vec{S'}}^{(\beta,\ell)}(s_i)\right\}_{1 \le i \le n} : \text{ a bootstrapping key from } \vec{s} \text{ to } \vec{S'} \\ \mathsf{KSK}_1 = \left\{\overline{\mathsf{CT}}_i = \mathsf{GLev}_{\vec{S'}}^{(\beta,\ell)}(s'_i)\right\}_{1 \le i \le kN} : \text{ a key switching key from } \vec{s'} \text{ to } \vec{S'} \\ \mathsf{KSK}_2 = \left\{\overline{\mathsf{ct}}_i = \mathsf{Lev}_{\vec{s}}^{(\beta,\ell)}(s'_i)\right\}_{1 \le i \le kN} : \text{ a key switching key from } \vec{s'} \text{ to } \vec{s} \\ \mathsf{RLK} = \left\{\overline{\mathsf{CT}}_{i,j} = \mathsf{GLev}_{\vec{S'}}^{(\beta,\ell)}\left(S'_i \cdot S'_j\right)\right\}_{1 \le i \le k}^{1 \le j \le i} : \text{ a relinearization key for } \vec{S'} \end{cases}$
Output: $\boxed{\mathsf{ct}_{\mathsf{out}} = \mathsf{LWE}_{\vec{s}}\left(\left[\left\lfloor \frac{m_1+m_2}{2^p} \right\rceil\right]_{2^p} \cdot \Delta\right)}$ $\boxed{\mathsf{ct}_{\mathsf{out}} = \mathsf{LWE}_{\vec{s}}\left(\left[\left\lfloor \frac{m_1 \cdot m_2}{2^p} \right\rceil\right]_{2^p} \cdot \Delta\right)}$
1 begin
      /* add p bits of padding                                          */
2     $\mathsf{ct}'_1 \leftarrow \mathsf{WoP\text{-}PBS}(\mathsf{ct}_1, \mathsf{BSK}, \mathsf{RLK}, \mathsf{KSK}_1, P_{\mathsf{Id}}, \Delta/2^p, 0, 0)$;
3     $\mathsf{ct}'_2 \leftarrow \mathsf{WoP\text{-}PBS}(\mathsf{ct}_2, \mathsf{BSK}, \mathsf{RLK}, \mathsf{KSK}_1, P_{\mathsf{Id}}, \Delta/2^p, 0, 0)$;
      /* compute the operation                                          */
4     $\boxed{\mathsf{ct}' \leftarrow \mathsf{ct}'_1 + \mathsf{ct}'_2}$ $\boxed{\mathsf{ct}' \leftarrow \mathsf{LWEMult}(\mathsf{ct}'_1, \mathsf{ct}'_2, \mathsf{RLK}, \mathsf{KSK}_1)}$ ;
      /* key switch                                                     */
5     $\mathsf{ct}'' \leftarrow \mathsf{PublicKS}(\mathsf{ct}', \mathsf{KSK}_2, \mathsf{Id})$ ;
      /* extract the LSB                                                */
6     $\mathsf{ct}'_{\mathrm{LSB}} \leftarrow \mathsf{WoP\text{-}PBS}(\mathsf{ct}'', \mathsf{BSK}, \mathsf{RLK}, \mathsf{KSK}_1, P_{\mathsf{Id}}, \Delta/2^p, p, 0)$;
      /* subtract the LSB to only keep the MSB                          */
7     $\mathsf{ct} \leftarrow \mathsf{ct}' - \mathsf{ct}'_{\mathrm{LSB}}$ ;
      /* key switch                                                     */
8     $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{PublicKS}(\mathsf{ct}, \mathsf{KSK}_2, \mathsf{Id})$ ;
9 end
```

same secret key $\vec{s} = (s_0, \ldots, s_{n-1}) \in \mathbb{Z}_q^n$*, with noise sampled from* χ_{σ_i}*. Let* $\mathsf{BSK}, \mathsf{KSK}, \mathsf{RLK}$ *be defined as in Theorem 28.*

Then, Algorithm 31 is able to compute a new LWE ciphertext $\mathsf{ct}_{\mathsf{out}}$*, encrypting the MSB of the sum, i.e., the carry,* $\left\lfloor\left\lfloor \frac{m_1 + m_2}{2^p} \right\rceil\right\rceil_{2^p} \cdot \Delta$ *(resp. a new LWE ciphertext* $\mathsf{ct}_{\mathsf{out}}$*, encrypting the MSB of the product* $\left\lfloor\left\lfloor \frac{m_1 \cdot m_2}{2^p} \right\rceil\right\rceil_{2^p} \cdot \Delta$*), under the secret key* $\vec{s'}$*.*

The variance of the noise of $\mathsf{ct}_{\mathsf{out}}$ *can be estimated by composing the noise formulae of the different operations composing the algorithm (Theorems 1, 6 and 27 to 29).*

The cost of Algorithm 31 is:

$$\begin{aligned}
\mathsf{Cost}\left(\mathsf{Add}_{2^p}^{\mathrm{MSB}}\right)^{(n,\ell_{\mathsf{PBS}},k_1,N_1,\ell_{\mathsf{KS}},\ell_{\mathsf{RL}},k_2,N_2)} &= 3\ \mathsf{Cost}\left(\mathsf{WoP\text{-}PBS}\right)^{(n,\ell_{\mathsf{PBS}},k_1,N_1,\ell_{\mathsf{KS}},\ell_{\mathsf{RL}},k_2,N_2)} \\
&\quad + 2\ \mathsf{Cost}\left(\mathsf{PublicKS}\right)^{(1,\ell_{\mathsf{KS}},k_2N_2,1,n)} \\
&\quad + 2\ (N_2+1)\mathsf{Cost}\left(\mathsf{add}\right) \\
\mathsf{Cost}\left(\mathsf{Mul}_{2^p}^{\mathrm{MSB}}\right)^{(n,\ell_{\mathsf{PBS}},k_1,N_1,\ell_{\mathsf{KS}},\ell_{\mathsf{RL}},k_2,N_2)} &= 3\ \mathsf{Cost}\left(\mathsf{WoP\text{-}PBS}\right)^{(n,\ell_{\mathsf{PBS}},k_1,N_1,\ell_{\mathsf{KS}},\ell_{\mathsf{RL}},k_2,N_2)} \qquad (7.3) \\
&\quad + 2\mathsf{Cost}\left(\mathsf{PublicKS}\right)^{(1,\ell_{\mathsf{KS}},k_2N_2,1,n)} \\
&\quad + (N_2+1)\mathsf{Cost}\left(\mathsf{add}\right) \\
&\quad + \mathsf{Cost}\left(\mathsf{LWEMult}\right)^{(\ell_{\mathsf{KS}},\ell_{\mathsf{RL}},k_2N_2,1,k_2N_2)}
\end{aligned}$$

Proof (Lemma 4) *The first two* WoP-PBS *of the algorithm send the two messages* m_1 *and* m_2 *to a lower scaling factor* $\frac{q}{2^{2p}}$*. This way, when the leveled addition (resp. the* LWEMult*) operation is performed, the new precision* $2p$ *will be able to store the entire (both MSB and LSB) exact result. The third* WoP-PBS *is used to extract only the LSB of the result, which will be subtracted from the result of the previous computation to obtain an encryption of the MSB with scaling factor* $\frac{q}{2^p}$*, i.e, ready to be used in the following computation. Observe that the key switches are used in order to switch the secret key in order to be compatible with the following operation.*

□

To conclude, we offer different ways to generalize TFHE's gate bootstrapping using the WoP-PBS introduced in Algorithm 27, Algorithm 28 or Algorithm 30, a key switch described in Algorithm 1 and the LWEMult introduced in Algorithm 22. Instead of doing a PBS after each Boolean gate, we only need to bootstrap when strictly needed, namely when we need to reduce the noise, change the encoding or apply a function.

	Gate Bootstrap TFHE	**Binary arithmetic ($p = 1$) as in Sec. 7.1.1**	**Integer arithmetic ($p > 1$) generalization in Sec. 7.1.3**
$\mathsf{Opp}_{2^\mathrm{P}}$	Negation	Addition with a constant	Negation
$\mathsf{Add}_{2^\mathrm{P}}$	Bootstrapped XOR	Homomorphic Add	Homomorphic Add
$\mathsf{Add}_{2^\mathrm{P}}^{\mathrm{MSB}}$	Bootstrapped AND	MultLWE	3 WoPBS + 2 Homomorphic Add + 2 public key switches
$\mathsf{Mul}_{2^\mathrm{P}}$	Bootstrapped AND	MultLWE	MultLWE
$\mathsf{Mul}_{2^\mathrm{P}}^{\mathrm{MSB}}$	$x \mapsto 0$	$x \mapsto 0$	3 WoPBS + MultLWE + Homomorphic Add + 2 public key switches
Noise reduction frequency	PBS at each gate	PBS when necessary	WoPBS when necessary

Table 7.1: Generalization of TFHE gate bootstrapping.

7.2 Big Integers

In order to overcome the limitations in the radix and CRT approaches for large integers in TFHE presented in Section 2.4, we propose two improvements. We start with a generalization of the radix approach to any large modulus Ω. Then, we propose a hybrid approach that takes the best of both the radix and the CRT approaches and allows us to work efficiently with any choice of moduli. In practice without the first improvement, the number of possible CRT residues is bounded by the numbers of small prime integers, thus harshly restricting the available general modulo Ω offered by the hybrid approach. Then, we explain how to use the Tree-PBS introduced by [GBA21] with our new large integers. Finally, we present extensive benchmarks to illustrate the interest of the WoP-PBS (Algorithm 30) coupled with our new integer representations.

7.2.1 Generalization of the Radix Approach

By using the radix representation, homomorphic modular integers are defined modulus Ω, which is a product of bases $\beta_i \in \mathbb{N}, i \in [0, \kappa - 1]$, i.e., $\Omega = \prod_{i=0}^{\kappa-1} \beta_i$. Here, we propose to remove this restriction by generalizing the previous arithmetic to any modulus Ω s.t. $\prod_{j=0}^{\kappa-2} \beta_j < \Omega < \prod_{j=0}^{\kappa-1} \beta_j$. The only difference with the previous approaches lies in the computation of the modular reduction. In what follows, we propose two complementary methods to perform this modular reduction, whose efficiency depends on Ω and the product of the selected bases.

A First Method for Modular Reduction. The first method consists in performing multiple LUT evaluations in the most significant block to reduce it modulo Ω. Indeed, the modular reduction is applied on the κ^{th} block (i.e., $\mathsf{ct}_{\kappa-1}$) which represents $m_{\kappa-1} \cdot \prod_{i=0}^{\kappa-2} \beta_i$ with $m_{\kappa-1} < p_{\kappa-1}$ and which might be larger than Ω. Here $p_{\kappa-1}$ is the carry-message modulus as described in Section 2.4. The complete process is detailed in Algorithm 32. The modular reduction is performed as a series of κ PBS (with KS, Line 2) and the result is a radix-based integer with a base $(\beta_0, \ldots, \beta_{\kappa-1})$ decomposition. The final step is to add the first $\kappa - 1$ blocks of the result of the modular reduction to the first $\kappa - 1$ blocks of the input (Line 4) and to replace the last block in the result by the $(\kappa - 1)$-th block obtained in the modular reduction (Line 5).

Algorithm 32: $(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1}) \leftarrow \mathsf{ModReduction}_1((\mathsf{ct}_0, \ldots, \mathsf{ct}_{\kappa-1}), \mathsf{PUB})$

```
Context: { P_j : r-redundant LUT for { $\mathbb{Z}_{p_{\kappa-1}} \to \mathbb{Z}_{\beta_j}$
                                       { $x \mapsto x'_j = \mathsf{Decomp}_j\left(x \cdot \prod_{h=0}^{\kappa-2} \beta_h \mod \Omega\right)$
         { $x'_j$ is the j-th element in the decomposition in base $(\beta_0, \ldots, \beta_{\kappa-1})$
         {   s.t. $x \cdot \prod_{h=0}^{\kappa-2} \beta_h \mod \Omega = x'_0 + \sum_{i=1}^{\kappa-1} x'_i \cdot \left(\prod_{j=0}^{i-1} \beta_j\right)$
Input: { $(\mathsf{ct}_0, \ldots, \mathsf{ct}_{\kappa-1})$, encrypting $\mathsf{msg} = m_0 + \sum_{i=1}^{\kappa-1} m_i \cdot \left(\prod_{j=0}^{i-1} \beta_i\right)$
       { s.t. $\mathsf{ct}_i$ encrypts message $m_i$ with parameters $(\beta_i, p_i)$
       { PUB: public material for KS-PBS
Output: $(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1})$, encrypting $\mathsf{msg} = m_0 + \sum_{i=1}^{\kappa-1} m_i \cdot \left(\prod_{j=0}^{i-1} \beta_i\right) \mod \Omega$
/* Decompose message in block $\kappa - 1$ with respect to base $(\beta_0, \ldots, \beta_{\kappa-1})$ */
1 for $j \in [\![0; \kappa - 1]\!]$ do
2     $c_j \leftarrow \mathsf{KS\text{-}PBS}(\mathsf{ct}_{\kappa-1}, \mathsf{PUB}, P_j)$
/* Add (as in Section 2.4.1) decomposition to all the blocks up to $\kappa - 2$ */
3 for $j \in [\![0; \kappa - 2]\!]$ do
4     $\mathsf{ct}'_j \leftarrow \mathsf{Add}(\mathsf{ct}_j, c_j)$
/* Replace block $\kappa - 1$ with element $\kappa - 1$ in the decomposition */
5 $\mathsf{ct}'_{\kappa-1} \leftarrow c_{\kappa-1}$
6 return $(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1})$
```

Observe that the κ KS-PBS in Line 2 of Algorithm 32 could be replaced by optimized procedures evaluating several different LUTs on the same input ciphertext. To do so, we could use the multi-value bootstrapping introduced in [CIM19] or the PBSmanyLUT presented in Algorithm 18.

Proof (Algorithm 32) *By construction, we have that $\prod_{j=0}^{\kappa-2} \beta_j < \Omega < \prod_{j=0}^{\kappa-1} \beta_j$. Then, reducing the $(\kappa - 1)$-th block encrypting the message $m_{\kappa-1} < p_{\kappa-1}$, rescaled by the product*

$\prod_{i=0}^{\kappa-2} \beta_i$ *modulus* Ω *is enough to correctly clear its carry space without loosing information.*

This is homomorphically done by evaluating the κ *functions* $x \in \mathbb{Z}_{p_{\kappa-1}} \mapsto \mathsf{Decomp}_j\left(x \cdot \prod_{h=0}^{\kappa-2} \beta_h \mod \Omega\right)$ *with* $j \in [\![0, \kappa-1]\!]$.

Then, for all $i \in [\![0; \kappa-1]\!]$, $\mathsf{Decrypt}(c_i) = r_i$, *giving* $r = r_0 + \sum_{i=1}^{\kappa-1} r_i \cdot \left(\prod_{j=0}^{i-1} \beta_j\right)$ *with* $r_i < \beta_i$ *and* $0 \leq r < \Omega$.

The last step is to compute the addition between each (ct_i, c_i) *for* $i \in [\![0, \kappa-2]\!]$ *and to replace the* $(\kappa-1)$*-th ciphertext with* $c_{\kappa-1}$. *The final output is given by* $\mathsf{ct}' = (\mathsf{ct}'_0, \ldots \mathsf{ct}'_{\kappa-1})$.

Then, for all $i \in [\![0; \kappa-2]\!]$, $\mathsf{Decrypt}(\mathsf{ct}'_i) = m_i + r_i$, *such that* $\mathsf{Decrypt}(\mathsf{ct}') = m_0 + r_0 + \sum_{i=1}^{\kappa-2}(m_i + r_i) \cdot \left(\prod_{j=0}^{i-1} \beta_j\right) + r_{\kappa-1} \prod_{j=0}^{\kappa-1} \beta_j$.

□

Second Method for Modular Reduction. The idea of the second method is based on the shape of $-\prod_{h=0}^{\kappa-2} \beta_h$ (i.e., the negation of the scaling factor of the message in the $\kappa-1$ block) reduced modulo Ω. The radix decomposition is:

$$\prod_{h=0}^{\kappa-2} \beta_h \mod \Omega = \nu_0 + \nu_1 \cdot \beta_0 + \nu_2 \cdot \beta_0\beta_1 + \ldots + \nu_{\kappa-1} \cdot \prod_{j=0}^{\kappa-2} \beta_j.$$

If $\nu_{\kappa-1} = 0$ and the other elements of the decomposition, i.e., $\nu_0, \nu_1, \ldots, \nu_{\kappa-2}$, are small integers (ideally many of them being 0), then this method is more efficient. Indeed, when these conditions are respected, the idea is to replace the MSB block by multiplying it by the non-zero constants ν_j and subtracting the results from the j-th input block, for $j \in [\![0, \kappa-2]\!]$. Some multiplications with positive constants are needed and might require some carry propagations beforehand depending on the degrees of fullness.

Each block can have a different base and we need to add ciphertexts of different blocks, at some point in the algorithm. Before this addition, we need to perform an homomorphic decomposition of a message into the right base with a series of PBSs as described in Algorithm 33. This step can be skipped if the bases are all equal. In Algorithm 33, a few functions are defined. Given a message m and its encoding $(m_0, \cdots, m_{\kappa-1})$ such that $m = m_0 + \sum_{i=1}^{\kappa-1} m_i \cdot \left(\prod_{j=0}^{i-1} \beta_j\right)$. We have

$$r_{i,\vec{\beta}}(m) = m_i.$$

$\gamma_{\vec{\beta}}(m)$ is defined such that for i $\in [\![\gamma_{\vec{\beta}}(m)+1, \kappa-1]\!]$, we have $m_i = 0$. When $\gamma_{\vec{\beta}}(m) =$

$\kappa - 1$, it means that all the m_i are non-zeros.

```
Algorithm 33: $(\mathsf{ct}_j)_{j\in[\![0,\gamma]\!]} \leftarrow \mathsf{Decomp}\left(\mathsf{ct_{in}}, \vec{\beta}, \vec{p}, \mathsf{PUB}\right)$
Context: $\begin{cases} (q,p,\deg) : \text{ parameters of } \mathsf{ct_{in}} \\ \mu := \deg\cdot(p-1) \\ q_{i,\vec{\beta}}(x) = \begin{cases} x & \text{if } i=-1 \\ \left\lfloor \frac{q_{i-1,\vec{\beta}}(x)}{\beta_i} \right\rfloor, & \text{if } i \geq 0 \end{cases} \\ r_{i,\vec{\beta}}(x) = q_{i-1,\vec{\beta}}(x) - q_{i,\vec{\beta}}(x)\cdot\beta_i,\ \forall i\geq 0 \\ \Omega'(x) = \left\{ i < |\vec{\beta}|,\ q_{i,\vec{\beta}}(x) = 0 \right\} \\ \gamma_{\vec{\beta}}(x) = \begin{cases} |\vec{\beta}| & \text{if } \Omega'(x)=\varnothing \\ \min(i\in\Omega'(x)), & \text{otherwise} \end{cases} \\ \gamma := \gamma_{\vec{\beta}}(\mu) \\ \vec{s}\in\mathbb{Z}^n : \text{the secret key} \\ P_{i,\vec{\beta}} : \text{a LUT for } x\to r_{i,\vec{\beta}}(x)\cdot\frac{q}{2\cdot p_i}, i\in[\![0,\kappa-1]\!] \end{cases}$
Input: $\begin{cases} \mathsf{ct_{in}} : \text{ LWE encryption of a message } m \\ (\vec{p},\vec{\beta})\in\mathbb{N}^{\kappa^2} \\ \mathsf{PUB}\text{: public material for } \mathsf{KS\text{-}PBS} \end{cases}$
Output: $(\mathsf{ct}_j)_{j\in[\![0,\gamma]\!]}$ encrypting the message $m$
1 for $j\in[\![0,\gamma]\!]$ do
2     $\mathsf{ct}_j \leftarrow \mathsf{KS\text{-}PBS}(\mathsf{ct_{in}}, \mathsf{PUB}, P_i)$
3     with $\mathsf{ct}_j$ LWE encryption with parameters $\left(q, \beta_j, p_j, \deg = \min(\frac{\beta_j-1}{p_j-1}, \frac{q_{j-1,\vec{\beta}}(\mu)}{p_j-1})\right)$
4 end
5 return $(\mathsf{ct}_j)_{j\in[\![0,\gamma]\!]}$
```

Our second method for modular reduction is detailed in Algorithm 34. Observe that in Algorithm 34 the subtraction algorithm follows the regular schoolbook subtraction modulo an integer Ω.

Remark 40 (Example of Algorithm 34) *Let us develop Algorithm 34 for a* 3*-block integer:*

$$m = m_0 + m_1\beta_0 + m_2\beta_0\beta_1 \textit{ and } \beta_0\beta_1 = \nu_0 + \nu_1\beta_0 + \nu_2\beta_0\beta_1 \bmod \Omega$$

therefore,

$$\begin{aligned} m &= m_0 + m_1\beta_0 + m_2\nu_0 + m_2\nu_1\beta_0 + m_2\nu_2\beta_0\beta_1 \\ &= (m_0 + m_2\nu_0) + (m_1 + m_2\nu_1)\beta_0 + (m_2\nu_2)\beta_0\beta_1 \quad \bmod \Omega. \end{aligned}$$

What happens is that if $\nu_2 = 0$, then we will have emptied the last block. Let us try this method on an example: $\Omega = 1055$, $\kappa = 3$, $\vec{\beta} = (\beta, \beta, \beta)$ with $\beta = 2^5$ and $\vec{p} = (p, p, p)$ with $p = 2^7$. Observe that $(2^5)^2 \mod 1055 = -31 = -1 \cdot 2^5 + 1$ (so $\nu_0 = 1$, $\nu_1 = -1$ and $\nu_2 = 0$). Then, the new reduced ciphertext would be composed of:

- *in the block 0: the addition between the previous block 0 and the previous block 2 multiplied times* 1*;*
- *in the block 1: the addition between the previous block 1 and the previous block 2 multiplied times* -1*;*
- *in the block 2: an encryption of* 0*.*

Algorithm 34: $\left(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1}\right) \leftarrow \mathsf{ModReduction}_2((\mathsf{ct}_0, \ldots, \mathsf{ct}_{\kappa-1}), \mathsf{PUB})$

```
Context: { $\vec{\nu} = (\nu_0, \nu_1, \ldots, \nu_{\kappa-1})$ be a convenient decomposition s.t.
         { $\prod_{h=0}^{\kappa-2} \beta_h \mod \Omega = \nu_0 + \nu_1\beta_0 + \nu_2\beta_0\beta_1 + \ldots + \nu_{\kappa-2}\prod_{j=0}^{\kappa-3}\beta_j$
Input: { $(\mathsf{ct}_0, \ldots, \mathsf{ct}_{\kappa-1})$, encrypting $\mathsf{msg} = m_0 + \sum_{i=1}^{\kappa-1} m_i \left(\prod_{j=0}^{i-1}\beta_i\right)$
       { s.t. $\mathsf{ct}_i$ encrypts message $m_i$ with parameters $(\beta_i, p_i)$
Output: $\left(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1}\right)$ encrypting $\mathsf{msg} = m_0 + \sum_{i=1}^{\kappa-1} m_i \left(\prod_{j=0}^{i-1}\beta_i\right) \mod \Omega$
/* Copy input and set the $\kappa-1$ block to zero (trivial encryption) */
1  $\left(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1}\right) \leftarrow (\mathsf{ct}_0, \ldots, \mathsf{ct}_{\kappa-2}, 0)$
2  for $j \in [\![0; \kappa-2]\!]$ do
       /* Multiply block $\kappa-1$ times $\nu_j$, Multiplication with a Positive Constant as
          in Section 2.4.1 */
3      if $\nu_j < 0$ then
4          $c_j \leftarrow \mathsf{ScalarMul}(\mathsf{ct}_{\kappa-1}, -\nu_j)$
5      else
6          $c_j \leftarrow \mathsf{ScalarMul}(\mathsf{ct}_{\kappa-1}, \nu_j)$
       /* Decompose (as in Algorithm 33) $c_j$ block starting from the $\beta_j$ */
7      $(c_{j,0}, \ldots, c_{j,\kappa-j-1}) \leftarrow \mathsf{Decomp}\left(c_j, (\beta_i)_{i\in[\![j,\kappa-1]\!]}, (p_i)_{i\in[\![j,\kappa-1]\!]}, \mathsf{PUB}\right)$
       /* Update the output */
8      if $\nu_j < 0$ then
9          $\left(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1}\right) \leftarrow \mathsf{Add}\left(\left(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1}\right), (c_{j,0}, \ldots, c_{j,\kappa-j-1}, 0, \ldots, 0)\right)$
10     else
11         $\left(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1}\right) \leftarrow \mathsf{Sub}\left(\left(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1}\right), (c_{j,0}, \ldots, c_{j,\kappa-j-1}, 0, \ldots, 0)\right)$
12 return $\left(\mathsf{ct}'_0, \ldots, \mathsf{ct}'_{\kappa-1}\right)$
```

Proof (Correctness of Algorithm 34) *Let us assume that the degree of fullness (Definition 14) of the input $(\kappa-1)$-th block is small enough. For this algorithm to work, we need to be able to perform a constant multiplication between this ciphertext and the largest of the constants $\nu_0, \ldots, \nu_{\kappa-2}$, followed by a homomorphic addition.*

The algorithm consists in multiplying the non-zero constants ν_j times the block $\mathsf{ct}_{\kappa-1}$ and then to subtract the result to the input ct_j block, for $j \in [\![0, \kappa-2]\!]$.

The result of this operation, by definition of the constants $\nu_0, \ldots, \nu_{\kappa-1}$, is a new radix-based encryption of msg reduced modulo Ω.

We can only add ciphertexts with messages encoded using the same base, so in case the bases in the blocks are not the same, a homomorphic decomposition step (as described in Algorithm 33) needs to be performed beforehand.

As for Algorithm 32, this new ciphertext is not a fresh *ciphertext, in the sense that the carries in the blocks are not all empty (because of the previous homomorphic addition). A carry propagation step can be applied if necessary and it can be used to continue the computations.*

□

7.2.2 Supporting Larger Integers using a Hybrid Representation

As we explained in Section 2.4, the CRT-only approach has some limitations. To overcome them, we create a new homomorphic hybrid representation that mixes the CRT-based approach with the radix-based approach, in order to take advantage of the best of both worlds. The idea is to use the CRT approach as the top layer in the structure, and to represent the CRT residues by using radix-based modular integers when needed, with this approach we do not face restrictions on Ω any more.

Encode. Let $(\Omega_0, \cdots, \Omega_{\kappa-1})$ be pairwise coprime integers i.e., (Ω_i, Ω_j) coprime for all $i \neq j$, and let $\Omega = \prod_{i=0}^{\kappa-1} \Omega_i$. To encode a message $\mathsf{msg} \in \mathbb{Z}_\Omega$, as in the CRT-only approach, the message is split into a list of $\{\mathsf{msg}_i\}_{i=0}^{\kappa-1}$ such that $\mathsf{msg}_i = \mathsf{msg} \mod \Omega_i$ for all $0 \leq i < \kappa$. At this point, for each message msg_i for $i \in [\![0, \kappa-1]\!]$, the encoding used for radix-based modular integers is used (Encode from Definition 13). Then, every CRT residues Ω_i has its own list of radix bases $(\beta_{i,\kappa_i-1}, \cdots, \beta_{i,0})$ and more generally its parameters $\{(\beta_{i,j}, p_{i,j})\}_{0 \leq j < \kappa_i} \in \mathbb{N}^{2\kappa_i}$. The formal encoding is described in the Figure 7.1.

$$\mathsf{msg} \mod \Omega \mapsto \begin{cases} \mathsf{msg}_0 = \mathsf{msg} \mod \Omega_0 \mapsto \begin{cases} \{m_{0,j}\}_{j=0}^{\kappa_0 - 1} \text{ s.t.} \\ \mathsf{msg}_0 = m_{0,0} + \sum_{j=1}^{\kappa_0 - 1} m_{0,j} \cdot \left(\prod_{k=0}^{j-1} \beta_{0,k}\right) \\ \text{and } \tilde{m}_{0,j} = \mathsf{Encode}\left(m_{0,j}, p_{0,j}, q\right) \\ \forall 0 \leq j < \kappa_0 \end{cases} \\ \vdots \\ \mathsf{msg}_{\kappa-1} = \mathsf{msg} \mod \Omega_{\kappa-1} \mapsto \begin{cases} \{m_{\kappa-1,j}\}_{j=0}^{\kappa_{\kappa-1} - 1} \text{ s.t.} \\ \mathsf{msg}_{\kappa-1} = m_{\kappa-1,0} + \sum_{j=1}^{\kappa_{\kappa-1} - 1} m_{0,j} \cdot \left(\prod_{k=0}^{j-1} \beta_{\kappa-1,k}\right) \\ \text{and } \tilde{m}_{\kappa-1,j} = \mathsf{Encode}\left(m_{\kappa-1,j}, p_{\kappa-1,j}, q\right) \\ \forall 0 \leq j < \kappa_{\kappa-1} \end{cases} \end{cases}$$

Figure 7.1: Hybrid approach visualisation combining CRT representation on the top level and radix representation below.

Decode. The decoding is done in two steps: first, each independent radix-based modular integer is decoded to obtain the independent residues modulo $\Omega_0, \ldots, \Omega_{\kappa-1}$, and then the CRT recombination to retrieve the message modulo Ω.

Arithmetic Operations. To perform a homomorphic operation, it is enough to perform the computation on each radix component independently, as shown for the CRT-only approach. Then, depending on the Ω_i values, the modular reduction can be performed using Algorithm 32 or Algorithm 34.

The hybrid approach can be seen as a generalization of both the CRT-only approach (if $\kappa_i = 1$ for all $0 \leq i < \kappa$) and the pure radix-based modular integer approach (if $\kappa = 1$). It also covers the mixed cases where some of the κ_i are equal to 1 and the others are greater.

For generic LUT evaluations, the only known solution in the literature is the Tree-PBS [GBA21]. We explain how to use it in Section 7.2.3. We can also use the $\mathsf{WoP\text{-}PBS}$ introduced in Algorithm 30 and we provide several benchmarks to support the interest of this new algorithm in Section 7.2.4.

7.2.3 Tree PBS Approach on Radix-Based Modular Integers

In this section, we give more details on how to apply the TreePBS technique by [GBA21] to our new radix-based modular integers.

In [GBA21] the plaintext integers are all decomposed under the same basis β: we offer here the possibility to evaluate a large look-up table with integers set in different bases $(\beta_0, \ldots, \beta_{\kappa-1})$.

Let $\Omega = \prod_{i=0}^{\kappa-1} \beta_i$, and let $L = [l_0, l_1, \cdots, l_{\Omega-1}]$ be a LUT with Ω elements. We want to evaluate this LUT on a radix-based modular integer encrypting a message $\mathsf{msg} = m_0 + \sum_{i=1}^{\kappa-1} m_i \prod_{j=0}^{i-1} \beta_j$.

The generalized multi-radix tree-PBS takes as input a radix-based modular integer ciphertext, a large look-up table L and the public material required for the PBS and key switching and returns a LWE ciphertext. The signature is:

$$\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{Tree\text{-}PBS}((\mathsf{ct}_0, \ldots, \mathsf{ct}_{\kappa-1}), \mathsf{PUB}, L).$$

To evaluate the multi-radix tree-PBS, we perform the following steps:

1. We define $\vec{\beta} = \{\beta_i | i \in [\![0, \kappa-1]\!]\}$ and $\vartheta(\beta_i)$ the component m_i of msg associated to β_i.
2. We define $\beta_{\max} = \max(\beta \in \vec{\beta})$.
3. We split the LUT L into $\nu = \frac{\prod_{\beta_i \in \vec{\beta}} \beta_i}{\beta_{\max}}$ smaller LUTs $(L_0, \ldots, L_{\nu-1})$ that each contain $\beta_{\max}$ different elements of L.
4. We compute a PBS on each of the ν LUTs using the ciphertext encrypting $\vartheta(\beta_{\max})$ as a selector.
5. We build a new large look-up table L by packing, with a key switching, the results of the ν iterations of the PBS in previous step.
6. We remove $\beta_{\max}$ from $\vec{\beta}$: $\vec{\beta} \leftarrow \vec{\beta} \setminus \{\beta_{max}\}$.
7. We repeat the steps from 2 to 6 until $\vec{\beta}$ is empty.

For the CRT-only and hybrid approaches, the multi-radix tree-PBS works in the same way.

7.2.4 Benchmarks with the WoP-PBS

In this section, we provide a few practical benchmarks for integers of sizes 16 and 32 bits. All the cryptographic parameters are provided below. The specifications of the machine are: Intel(R) Xeon(R) Platinum 8375C@2.90GHz with 504GB of RAM. Note that such an amount of RAM is not needed: all benchmarks can be run on a basic laptop. All implementations are done using TFHE-rs [Zam22b] (the follow-up of the Concrete library [Zam22a]).

Cryptographic Parameters

In Tables 7.2 and 7.3, we report the cryptographic parameters that we use to compute our benchmarks. All of them have been obtained with the optimization framework described in Chapter 4. In those tables, the notation $\mathfrak{B}$ (resp. ℓ) refers to the basis (resp. the number of levels) parameter used for a given FHE algorithm such as a key switch or a [Chi+20a]'s PBS. By default, the cryptographic parameters ensure 128 bits of security, a failure probability $p_{\mathsf{fail}}\left(\mathcal{A}^{(\mathrm{CJP21})}\right), p_{\mathsf{fail}}\left(\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}\right) \leq 2^{-13.9}$ i.e., a standard score (Definition 22) of 4.

Remark 41 (Largest 2-Norm) *For a given message modulo β and carry-message modulo p one can find the worst 2-norm that they could encounter in the modular arithmetic defined in Section 2.4.1. Indeed, a fresh encoding is at worst $\beta - 1$, and the largest message one can consider before needing to empty the carry buffer is $p-1$, so the largest integer one can multiply a ciphertext with is $\left\lfloor \frac{p-1}{\beta-1} \right\rfloor$ which is the largest 2-norm.*

param ID	AP parameters		LWE		GLWE			LWE-to-LWE key switch		PBS		WoP-PBS compatible
	p	ν	n	$\log_2(\sigma)$	k	$\log_2(N)$	$\log_2(\sigma)$	$\log_2(\mathfrak{B})$	ℓ	$\log_2(\mathfrak{B})$	ℓ	
# 1	2^2	3	615	-13.38	4	9	-51.49	2	5	12	3	#8
# 2	2^4	5	702	-15.69	2	10	-51.49	2	7	9	4	#9
# 3	2^6	5	872	-20.21	1	12	-62.00	4	4	22	1	#10
# 4	2^2	3	667	-14.76	6	8	-37.88	4	3	18	1	$\varnothing$
# 5	2^4	5	784	-17.87	2	10	-51.49	4	3	23	1	$\varnothing$
# 6	2^8	17	983	-23.17	1	14	-62.00	4	5	15	2	$\varnothing$
# 7	2^6	9	838	-19.30	1	12	-62.00	3	5	15	2	$\varnothing$

Table 7.2: Optimized parameters for the AP of type $\mathcal{A}^{(\mathrm{CJP21})}$.

In Table 7.2, we provide seven parameter sets for $\mathcal{A}^{(\mathrm{CJP21})}$, each one with a bit of padding, a specific message modulus p and a specific 2-norm ν (Theorem 5). In Table 7.3, we provide five parameters sets for $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$, each one with a specific (carry-)message modulo p, a specific number of bits to extract per LWE ciphertext during the WoP-PBS, a specific number κ of input LWE ciphertexts to the WoP-PBS and a specific 2-norm ν. They do not have a bit of padding. In parameter IDs #11 and #12, the message modulus specifies the CRT base used and the corresponding number of bits to extract for each base.

Compatibility Between $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ and $\mathcal{A}^{(\mathbf{CJP21})}$. We generated couples of parameter sets that are compatible, one for $\mathcal{A}^{(\mathrm{CJP21})}$ and the other for $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$. By compatible,

param ID	AP parameters				LWE		GLWE			micro parameters		
	p	bit(s) to extract	κ	ν	n	$\log_2(\sigma)$	k	$\log_2(N)$	$\log_2(\sigma)$	operator	$\log_2(\mathfrak{B})$	ℓ
# 8	2^2	1	16	3	549	-11.62	2	10	-51.49	LWE-to-LWE key switch	2	5
										PBS	12	3
										packing key switch	17	2
compatible with CJP#1										circuit bootstrapping	13	1
# 9	2^4	2	8	5	534	-11.22	2	10	-51.49	LWE-to-LWE key switch	2	5
										PBS	12	3
										packing key switch	17	2
compatible with CJP#2										circuit bootstrapping	9	2
# 10	2^6	4	5	5	538	-11.33	4	10	-62.00	LWE-to-LWE key switch	1	10
										PBS	4	11
										packing key switch	20	2
compatible with CJP#3										circuit bootstrapping	7	4
# 11	$\begin{pmatrix}7\\8\\9\\11\\13\end{pmatrix}$	$\begin{pmatrix}3\\3\\4\\4\\4\end{pmatrix}$	5	5	696	-15.53	2	10	-51.49	LWE-to-LWE key switch	2	7
										PBS	9	4
										packing key switch	17	2
										circuit bootstrapping	7	3
# 12	$\begin{pmatrix}3\\11\\13\\19\\23\\29\\31\\32\end{pmatrix}$	$\begin{pmatrix}2\\4\\4\\5\\5\\5\\5\\5\end{pmatrix}$	8	2^5	781	-17.79	1	11	-51.49	LWE-to-LWE key switch	1	16
										PBS	5	8
										packing key switch	13	3
										circuit bootstrapping	6	4

Table 7.3: Optimized parameters for the AP of type $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$.

we mean that one can go from one to the other freely and smoothly. From $\mathcal{A}^{(\mathrm{CJP21})}$ to $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$, one needs to remove the bit of padding in the usual LUT of the PBS of $\mathcal{A}^{(\mathrm{CJP21})}$. From $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ to $\mathcal{A}^{(\mathrm{CJP21})}$, one needs to add a bit of padding in the LUT of the usual WoP-PBS of $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$. But we also need other guarantees to be able to freely compose atomic patterns $\mathcal{A}^{(\mathrm{CJP21})}$ and $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$. In particular, we need to guarantee that (i) each atomic pattern can absorb/deal with input noise either coming from $\mathcal{A}^{(\mathrm{CJP21})}$ or $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ and (ii) the input LWE dimensions of each atomic pattern are compatible i.e., the product of the GLWE dimension k by the polynomial size N must be equal in both APs. We could remove constraint (ii) by adding two key switching keys, one to go from $\mathcal{A}^{(\mathrm{CJP21})}$ to $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ and one to go from $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ to $\mathcal{A}^{(\mathrm{CJP21})}$: we leave it to future research.

To satisfy those two conditions, we decided to first solve the optimization problem on $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ and later on $\mathcal{A}^{(\mathrm{CJP21})}$ with more constraints. The first optimization gives us the product $k \cdot N$ and the output variance of $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$. Then we solve the optimization problem for $\mathcal{A}^{(\mathrm{CJP21})}$ with an additional constraint for the polynomial size N and the GLWE dimension k to satisfy (i) and using the maximum between the output noise of $\mathcal{A}^{(\mathrm{CJP21})}$ and $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ as the input noise of $\mathcal{A}^{(\mathrm{CJP21})}$ which satisfies (ii). This approach works well for parameter couples (#1,#8) and (#2,#9). But for the last parameter set couple (#3,#10), there is no solution for $\mathcal{A}^{(\mathrm{CJP21})}$ with the aforementioned constraints. For this special case, we reverse the order of the optimization and first solve the optimization problem for $\mathcal{A}^{(\mathrm{CJP21})}$ and then for $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ with the additional constraints mentioned above.

Experimental Results

The tables presented in this section contain timings related to 16 and 32-bit integer operations using the radix approach (Table 7.4), the CRT approach (Table 7.5) and the native CRT approach (Table 7.6). The benchmarks measure timings to compute homomorphic additions, multiplications, carry cleanings (apart from the native CRT approach) and LUT evaluations (only for 16-bits integers). As explained in Remark 37, it is not doable to evaluate LUTs on 32-bit integers.

Radix Approach. In Table 7.4, dedicated to the radix approach, we display two instances of 16-bit integers and three instances of 32-bit integers. The number of additions is bounded by the room available in the carry buffer, and once it is full, a carry cleaning is needed.

For 16-bit integers, it is possible to use both the $\mathcal{A}^{(\mathrm{CJP21})}$-based and the $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$-based operators. This means that for 16-bits integer, classical arithmetic uses the usual PBS ($\mathcal{A}^{(\mathrm{CJP21})}$), and LUT evaluation is done with the WoP-PBS ($\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$). We assume that the WoP-PBS is done over integers with free carry buffers (i.e., after a carry cleaning). The parameters have been generated as described in Paragraph 7.2.4. Note that the addition does not require any PBS to be computed (this is denoted with a star $+^*$), but is done accordingly to the parameters generated for the PBS.

For 32-bit integers, only arithmetic operations are possible. So, cryptographic parameters are optimized following $\mathcal{A}^{(\mathrm{CJP21})}$ only. Hence, some operations are computed faster for the 32-bit integers than for the 16-bit ones.

integer parameters				**PBS based operations**				**WoP-PBS based operations**	
Ω	p	carry modulus	κ	param ID	$+^*$	$\times$	carry cleaning	param ID	LUT evaluation
2^{16}	2^1	2^1	16	#1	12.8 µs	29.0 s	932 ms	#8	823 ms
2^{16}	2^2	2^2	8	#2	6.67 µs	5.73 s	657 ms	#9	1.80 s
2^{32}	2^1	2^1	32	#4	19.1 µs	43.8 s	685 ms	$\varnothing$	
2^{32}	2^2	2^2	16	#5	12.3 µs	9.60 s	514 ms		
2^{32}	2^4	2^4	8	#6	137 µs	25.0 s	6320 ms		

Table 7.4: Benchmarks for 16-bit and 32-bit homomorphic integers based on the radix approach. The star (*) means that a PBS is not required to compute the operation.

Remark 42 (Multiplication Failure Probability) *When* 32*-bit integers are represented with* 32 *blocks (i.e.,* $\kappa = 32$*), the number of AP of type* $\mathcal{A}^{(CJP21)}$ *required to compute a multiplication is quadratic in the number of blocks. Because the error probability* $\mathsf{p}_{\mathsf{fail}}$ *of this AP is bounded by* $2^{-13.9}$ *in our experiments, the error probability at the level of the multiplication will be increased greatly. To balance this, one can use the technique described in Section 4.2.5. Timings are clearly not in favor of this representation, and the probability of having an error is small enough for the other representations (with a smaller number of blocks). One solution is to keep the same value of* $\mathsf{p}_{\mathsf{fail}}$ *and consider a small enough* κ*, resulting in a better trade-off between running time and failure probability at the multiplication level (e.g., the one associated with the parameter ID#5). Another way of solving this problem would be to have another parameter set dedicated to the multiplication algorithm, with a smaller failure probability* $\mathsf{p}_{\mathsf{fail}}$ *but we leave that as an avenue for future work.*

CRT Approach. Table 7.5 is dedicated to the CRT approach. In this representation, each block has a dedicated basis, and we can perform some operations independently on each block. We display one instance for 16-bit integers and another one for 32-bit integers. For both of them we show the total time needed to compute the operations, as well as the amortized time when the implementation is multi-threaded. As for Table 7.4, the number of additions is bounded by the room available in the carry buffer, and once it is full, a carry cleaning in needed. Note that in the case of homomorphic evaluation of polynomial functions, using the CRT representation offers better timings, since it is sufficient to compute a PBS on each CRT residue. The timings are then the same as the ones of the carry cleaning when there is one block per residue, otherwise it means that we are considering the hybrid approach, and in that case, it is the cost of a LUT evaluation separately on each block.

For the 16-bit integers, the basis is given by $\Omega = 2^3 \cdot 3^2 \cdot 7 \cdot 11 \cdot 13 \approx 2^{16}$. As for 16-bit integers in radix representation, it is possible to use both the $\mathcal{A}^{(\mathrm{CJP21})}$ and the $\mathcal{A}^{(\mathsf{WoP\text{-}PBS})}$ operators. However, the major difference here is about the parameter optimization: in this case, the atomic pattern $\mathcal{A}^{(\mathrm{CJP21})}$ has been optimized first. Thus, the timings for evaluating a LUT using a WoP-PBS are way slower. By removing the constraint of compatibility, the performance should be closer to the one of Table 7.6. The WoP-PBS is parallelized by extracting bits for each block independently. Then, each LUT evaluation outputting one block (and taking all bits as input) is computed in parallel: note that this approach could also be applied in the case of the radix decomposition.

We consider the basis defined by $\Omega = 2^5 \cdot 3^5 \cdot 5^4 \cdot 7^4 \approx 2^{32}$ to represent 32-bit integers using the hybrid representation. For instance, to represent integers modulo 7^4, we use radix-based integers with 4 blocks and a message modulus equal to 7. Thanks to the CRT representation, by using this base, multiplications can be computed with the fast bi-variate PBS approach described in Section 2.4.1.

Ω	type of execution	**PBS based operations**				**WoP-PBS based operations**	
		param ID	$+^*$	$\times$	carry cleaning	param ID	LUT evaluation
$\approx 2^{16}$	sequential	#3	8.36 µs	401 ms	251 ms	#10	23.1 s
	5 threads		1.67 µs	80.3 ms	50.2 ms		4.61 s
$\approx 2^{32}$	sequential	#7	27.6 µs	5.17 s	2400 ms	$\varnothing$	
	4 threads		8.78 µs	1.82 s	729 ms		

Table 7.5: Benchmarks for 16-bit homomorphic integers based on the CRT approach and 32-bit integers are computed with a hybrid approach. We use the following CRT bases: $\Omega = 7 \cdot 8 \cdot 9 \cdot 11 \cdot 13 \approx 2^{16}$ and $\Omega = 2^5 \cdot 3^5 \cdot 5^4 \cdot 7^4 \approx 2^{32}$.

Native CRT Approach. Following up on what we present in Section 2.4.2, it is possible to have a fast version of CRT-encoded integers spread out across several ciphertexts. For example, to encode a residue $m_0 = 3 \mod 7$, its associated plaintext is $\left\lfloor 3\frac{q}{7} \right\rceil$. Additions and scalar multiplications are extremely fast in this context: they do not require any PBS and they can be computed independently and in parallel on each of the CRT residues with fast FHE operators. Indeed, to compute additions we use the LWE addition on each residue, and to compute a scalar multiplication by α, we decompose α with our CRT bases into smaller integers, and compute scalar multiplications with them.

To the best of our knowledge, until we completed this work, there were no efficient algorithm to compute a LUT in this context. However, one can use the new WoP-PBS to do so. They simply need to extract the first $\lceil \log_2(\beta_i) \rceil$ most significant bits for each residue, thanks to negacyclic sign functions. A bit extraction on a non-power-of-two encoded ciphertext has to be computed in the exact same manner.

The sign evaluation for an integer message m encoded as $\tilde{m} = m \cdot \frac{q}{p}$ (where both q and p are powers of two) is computed by adding to the input ciphertext $\frac{q}{2p}$, computing a PBS on it with a trivial encryption of $P(X) = \sum_{i=0}^{N-1} -\frac{q}{2p'} X^i$ and finally adding to the output $\frac{q}{2p'}$. The output is the encoding $b \cdot \frac{q}{p'}$ where b is the most significant bit of m. With p not being a power of two, one needs to replace $\frac{q}{2p}$ with $\left\lfloor \frac{q}{2p} \right\rceil$.

In Table 7.6, dedicated to the native CRT approach, we display one instance of 16-bit integers and another of 32-bit integers. We consider $\Omega = 7 \cdot 8 \cdot 9 \cdot 11 \cdot 13 \approx 2^{16}$ and respectively $\Omega = 3 \cdot 11 \cdot 13 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 32 \approx 2^{32}$. Since there is no carry buffer in this representation, there is no need for a carry cleaning. However, to avoid incorrect computations, the number of additions is bounded for these parameter sets by the value ν. Once this bound is reached, a WoP-PBS is required to reduce the noise.

We observe a slower timing for the multiplication with 32-bit integers, 36.8 seconds, which leads us to think that for a precision close to 32 bits, a hybrid approach is more efficient. Indeed, the native CRT approach requires to have in a single LWE ciphertext a small enough noise (after the bootstrapping) to preserve the message (with a size equal to the coprime modulus) and the room for the 2-norm ν needed to compute multiplications with known integers or additions between ciphertexts. So when one tries to build a large Ω, since small primes are very limited in number, they end up with large coprime residues and as a consequence, implies a higher 2-norm which means very slow parameter sets.

Ω	type of execution	**WoP-PBS based operations**				
		param ID	$+^*$		$\times$	LUT evaluation
			ν	time		
$\approx 2^{16}$	sequential	#11	5	4.32 µs	7.42 s	3.81 s
	5 threads			0.862 µs	1.65 s	0.761 s
$\approx 2^{32}$	sequential	#12	2^5	6.98 µs	36.8 s	$\varnothing$
	8 threads			0.873 µs	5.31 s	

Table 7.6: Benchmarks for 16-bit and 32-bit homomorphic integers based on the native CRT approach. We use the CRT bases: $\Omega = 7 \cdot 8 \cdot 9 \cdot 11 \cdot 13 \approx 2^{16}$ and $\Omega = 3 \cdot 11 \cdot 13 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 32 \approx 2^{32}$.

Conclusion In this chapter, we explained how to build two arithmetics. The first one is particularly efficient with small integers and leverage the LWE multiplication introduced in Chapter 5 and one of the WoP-PBS algorithms introduced in Chapter 6. The second one is dedicated to large integers and uses an encoding that generalizes both the CRT and radix approaches. The WoP-PBS introduced in Algorithm 30 and Theorem 30 is the core enabler of this arithmetic. We presented several benchmarks comparing the different encoding approaches for 16-bits and 32-bits integers.

Chapter 8

TFHE on New Problems

From Chapter 3 to Chapter 7, we improved TFHE by either finding optimal parameters or by introducing new and more efficient algorithms. Here, we adopt a different approach: we modify the security hypothesis on which TFHE relies to improve the efficiency of the state-of-the-art algorithms as well as the new algorithms introduced in this thesis. We now introduce two new kinds of secret keys in this chapter, alongside new algorithms especially efficient with the new hypothesis.

In Section 2.3, we introduced Limitation 9 which states that there is no fine-grained control over the size of a GLWE secret key. We also introduced Limitation 10 which states that there is a noise plateau for LWE dimensions greater than n_{plateau}. The partial secret keys introduced in this chapter are meant to overcome these limitations.

In the previous chapters, we saw that the key switch is important to reduce the LWE dimension before performing a PBS. This chapter introduces another, new type of secret key which is particularly efficient during key switches.

Remark 43 (Concurrent Work) *Lee et al. [Lee+23] introduced a new type of secret keys called block binary keys. Those are different from our contribution but concurrently exploit the advantage of having nested secret keys as introduced in this chapter under the name of shared randomness secret keys.*

In [LY23], Lee and Yoon describe a way to publicly transform a bootstrapping key encrypted under a traditional secret key to an extended version encrypted under a secret key containing zeros between secret coefficients. This extended bootstrapping key allows the authors to bootstrap messages with higher precisions, but it does not improve the noise growth. This contribution only involves traditional GLWE secret keys for encryption.

8.1 Partial GLWE Secret Keys

We introduced in Section 3.1 the concept of noise oracles (Definition 17) and we gave their formula in Equation (3.3) and Table 3.1. The formula shows that most of the time, when increasing the LWE dimension, we can decrease the encryption variance while keeping the same security. But, we cannot decrease the encryption variance at will. We need to enforce a minimal variance to be sure that ciphertexts contain non-zero noises as explained in Limitation 10. In Remark 16, we found that for LWE dimensions greater than $n_{\mathsf{plateau}} = 2443$, the noise standard deviation is constant and equal to 2^2.

Let us now focus on the choice of GLWE macro-parameters i.e., the polynomial size N and the GLWE dimension k. As long as the product $k \cdot N$ remains smaller that n_{plateau}, increasing N will increase the cost but at the same time, as we select a smaller encryption noise, we can potentially choose smaller values for other parameters (for instance the maximum level of the decomposition in a PBS). Unfortunately, when the product $k \cdot N$ is greater than n_{plateau}, we cannot decrease the encryption variance. Therefore, we will have a greater cost but without any benefit with respect to the noise. Interestingly, the new type of secret keys introduced in this section are beneficial to the noise even when the product $k \cdot N$ is greater than n_{plateau}.

A GLWE secret key usually contains kN random elements. Our first observation is that we can allow this secret key to contain only a number ϕ of random elements and allow the rest of them to be set to zero. We first formally define this notion of *partial secret keys* and list the different advantages and improvements that they offer.

A partial GLWE secret key is composed of two parts, the first one contains secret random elements (sampled from a distribution $\mathcal{D}$) and the second part is filled with *zeros at known positions*. As a simple example, we could define the following partial GLWE secret key:

$$\vec{S} = (S_0, S_1) \in \mathfrak{R}_{q,N}^2 \quad \text{with} \quad S_0 = \sum_{j=0}^{N-1} s_{0,j} X^j \quad \text{and} \quad S_1 = \sum_{j=0}^{N/2-1} s_{1,j} X^j \ ,$$

where $s_{0,0}, \cdots, s_{0,N-1}$ and $s_{1,0}, \cdots, s_{1,\frac{N}{2}-1}$ are sampled from $\mathcal{D}$, and the other coefficients are publicly known to be set to zero.

Definition 35 (GLWE Partial Secret Key) *A Partial GLWE secret key is a vector $\vec{S}^{[\phi]} \in \mathfrak{R}_{q,N}^k$ associated to a filling parameter ϕ such that $0 \leq \phi \leq kN$. This key will have ϕ random coefficients sampled from a distribution $\mathcal{D}$ and $kN - \phi$ known zeros. Both* *the locations of the random elements and the zeros are public. By convention, we fill the coefficients starting at coefficient $s_{0,0}$, then $s_{0,1}$ and so on, and when the first polynomial is entirely filled, we fill the second polynomial starting at $s_{1,0}$ and so on, until ϕ coefficients are determined, up to $s_{k-1,N-1}$.*

When we write $\mathsf{Var}\left(\vec{S}^{[\phi]}\right)$ (resp. $\mathbb{E}\left(\vec{S}^{[\phi]}\right)$), we refer to the variance (resp. the expectation) of $\mathcal{D}$ (either a uniform binary distribution, uniform ternary distribution, Gaussian distribution or small uniform distribution). When $\mathcal{D}$ is a uniform binary distribution, $\mathsf{Var}\left(\vec{S}^{[\phi]}\right) = 1/4$ and $\mathbb{E}\left(\vec{S}^{[\phi]}\right) = 1/2$.

In Definition 11, we introduced the flattened representation of a GLWE secret key. Now, we define the flattened representation of a partial GLWE secret key.

Definition 36 (Flattened Representation of a Partial GLWE Secret Key) *A partial GLWE secret key $\vec{S}^{[\phi]} = \left(S_0 = \sum_{j=0}^{N-1} s_{0,j}X^j, \cdots, S_{k-1} = \sum_{j=0}^{N-1} s_{k-1,j}X^j\right) \in \mathfrak{R}_{q,N}^k$ (Definition 35) can be viewed as a flattened LWE secret key $\vec{\bar{s}} = (\bar{s}_0, \cdots, \bar{s}_{\phi-1}) \in \mathbb{Z}^\phi$ in the following manner: $\bar{s}_{iN+j} := s_{i,j}$, for $0 \leq j < N$ and $0 \leq i < k$ with $iN + j < \phi$. This flattened representation contains only ϕ unknown coefficients.*

This type of keys seems to be a secure solution, taking into account the plateau limitation (Limitation 10). The hardness of partial keys is studied in [Ber+23b].

Impact of Partial Keys on the Noise Distribution. Regarding the security of the partial secret key and the different attacks presented in Section 2.1.2, we can use the security oracle (Definition 17 in Section 3.1) to find out the smallest noise variance σ^2 for an $\mathsf{LWE} \in \mathbb{Z}_q^{\phi+1}$ guarantying the desired level of security λ. By using this same σ^2 for $\mathsf{GLWE} \in \mathfrak{R}_{q,N}^{k+1}$ with partial secret key $S^{[\phi]}$ we obtain the same level of security λ.

8.1.1 Advantages of Partial GLWE Secret Keys

Partial GLWE secret keys enable, in many contexts, to have a smaller computational cost for certain algorithms and/or to have a smaller noise growth. This will lead to faster parameter sets (for a given failure probability and security level) after optimization. More details are provided in Section 8.3.

Advantages for Sample Extraction

In Section 2.3, we introduced the sample extract (Algorithm 5 and Theorem 9). In Algorithms 35 and 36, we explain how to compute a sample extract (i.e., transforming a GLWE ciphertext into an LWE ciphertext) in the context of a partial GLWE secret key. They are generalizations of the same algorithm used for *traditional* secret keys. Indeed, a traditional secret key is captured when $\phi = k \cdot N$. We prove the correctness of those algorithms in Appendix A.4.

```
Algorithm 35: $\mathsf{ct_{out}} \leftarrow \mathsf{ConstantSampleExtract}(\mathsf{CT_{in}})$

Context: $\vec{S}^{[\phi]} \in \mathfrak{R}_{q,N}^k$ : a partial secret key (Definition 35)
         $(k-1)N+1 \le \phi \le kN$ : filling parameter for the partial secret key
         $\vec{\bar{s}} \in \mathbb{Z}^\phi$ : the flattened version of $\vec{S}^{[\phi]}$ (Definition 36)
         $P := \sum_{i=0}^{N-1} p_i X^i \in \mathfrak{R}_{q,N}$
         $\mathsf{CT_{in}} = \left(\sum_{i=0}^{N-1} a_{0,i}X^i, \cdots, \sum_{i=0}^{N-1} a_{k-1,i}X^i, \sum_{i=0}^{N-1} b_i X^i\right) \in \mathfrak{R}_{q,N}^{k+1}$
Input: $\mathsf{CT_{in}} \in \mathsf{GLWE}_{\vec{S}^{[\phi]}}(P)$ : a GLWE encryption of the plaintext $P$
Output: $\mathsf{ct_{out}} \in \mathsf{LWE}_{\vec{\bar{s}}}(p_0)$ : an LWE encryption of the plaintext $p_0$
1 for $i \in [\![0; \phi-1]\!]$ do
2     set $\alpha := \lfloor \frac{i}{N} \rfloor$, $\beta := (N-i) \bmod N$ and $\gamma := 1 - (\beta == 0)$
3     set $a_{\mathsf{out},i} := (-1)^\gamma \cdot a_{\alpha,\beta}$
4 return $\mathsf{ct_{out}} := (a_{\mathsf{out},0}, \cdots, a_{\mathsf{out},\phi-1}, b_0) \in \mathbb{Z}_q^{\phi+1}$
```

```
Algorithm 36: $\mathsf{ct_{out}} \leftarrow \mathsf{SampleExtract}(\mathsf{CT_{in}}, \alpha)$

Context: $\vec{S}^{[\phi]} \in \mathfrak{R}_{q,N}^k$ : a partial secret key (Definition 35)
         $(k-1)N+1 \le \phi \le kN$ : filling parameter for the partial secret key
         $\vec{\bar{s}} \in \mathbb{Z}^\phi$ : the flattened version of $\vec{S}^{[\phi]}$ (Definition 36)
         $P := \sum_{i=0}^{N-1} p_i X^i \in \mathfrak{R}_{q,N}$
Input: $\mathsf{CT_{in}} \in \mathsf{GLWE}_{\vec{S}^{[\phi]}}(P)$ : a GLWE encryption of the plaintext $P$
       $0 \le \alpha \le N-1$ : the coefficient to extract
Output: $\mathsf{ct_{out}} \in \mathsf{LWE}_{\vec{\bar{s}}}(p_\alpha)$ : an LWE encryption of the plaintext $p_\alpha$
  /* Rotation of the GLWE ciphertext                                    */
1 set $\mathsf{CT} := X^{-\alpha} \cdot \mathsf{CT_{in}}$
  /* Call to Algorithm 35                                               */
2 return $\mathsf{ct_{out}} := \mathsf{ConstantSampleExtract}(\mathsf{CT})$
```

Remark 44 (Noise and Cost of Sample Extract) *A sample extract, whether it includes a partial secret key or not, does not add any noise to the plaintext as explained in Theorem 9. With partial keys, the cost of the sample extraction stays roughly the same and it is negligible.*

Advantages for the GLWE Key Switch

Algorithm 37: $\mathsf{CT}_{\mathsf{out}} \leftarrow \mathsf{GlweKeySwitch}(\mathsf{CT}_{\mathsf{in}}, \mathsf{KSK})$

Context: $\begin{cases} (k_{\mathsf{in}}-1)N < \phi_{\mathsf{in}} \leq k_{\mathsf{in}}N \text{ and } (k_{\mathsf{out}}-1)N < \phi_{\mathsf{out}} \leq k_{\mathsf{out}}N \\ \vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]} \in \mathfrak{R}_{q,N}^{k_{\mathsf{in}}} : \text{the input partial secret key (Definition 35)} \\ \vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]} = (S_{\mathsf{in},0}, \cdots, S_{\mathsf{in},k_{\mathsf{in}}-1}) \\ \vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]} \in \mathfrak{R}_{q,N}^{k_{\mathsf{out}}} : \text{the output partial secret key (Definition 35)} \\ \ell \in \mathbb{N} : \text{ the number of levels of the decomposition} \\ \mathfrak{B} \in \mathbb{N} : \text{ the base of the decomposition} \end{cases}$

Input: $\begin{cases} \mathsf{CT}_{\mathsf{in}} = (A_0, \cdots, A_{k_{\mathsf{in}}-1}, B) \in \mathsf{GLWE}_{\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}}(P) \subseteq \mathfrak{R}_{q,N}^{k_{\mathsf{in}}+1}, \text{ with } P \in \mathfrak{R}_{q,N} \\ \mathsf{KSK} = \{\mathsf{KSK}_i = \{\mathsf{KSK}_{i,j}\}_{0\leq j \leq \ell-1}\}_{0 \leq i \leq k_{\mathsf{in}}-1}, \text{ with} \\ \quad \mathsf{KSK}_{i,j} \in \mathsf{GLWE}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}\left(\frac{q}{\mathfrak{B}^j} \cdot S_{\mathsf{in},i}\right), \text{ for } 0 \leq i \leq k_{\mathsf{in}}-1 \text{ and } 0 \leq j \leq \ell - 1 \end{cases}$

Output: $\mathsf{CT}_{\mathsf{out}} \in \mathsf{GLWE}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}(P)$

```
  /* Keep the $B$ part                                              */
1 Set $\mathsf{CT}_{\mathsf{out}} := (0, \cdots, 0, B) \in \mathfrak{R}_{q,N}^{k_{\mathsf{out}}+1}$
2 for $i \in [\![0; k_{\mathsf{in}}-1]\!]$ do
      /* Decompose the mask                                         */
3     Update $\mathsf{CT}_{\mathsf{out}} = \mathsf{CT}_{\mathsf{out}} - \left\langle \vec{K}_i, \mathsf{dec}^{(\mathfrak{B},\ell)}(A_i) \right\rangle$
4 return $\mathsf{CT}_{\mathsf{out}}$
```

A GLWE-to-GLWE key switching with $N \neq 1$, as described in Algorithm 37, takes as input a GLWE ciphertext $\mathsf{CT}_{\mathsf{in}} \in \mathfrak{R}_{q,N}^{k_{\mathsf{in}}+1}$ encrypting the plaintext $P \in \mathfrak{R}_{q,N}$ under the secret key $\vec{S}^{[\phi_{\mathsf{in}}]} \in \mathfrak{R}_{q,N}^{k_{\mathsf{in}}}$, and outputs $\mathsf{CT}_{\mathsf{out}} \in \mathfrak{R}_{q,N}^{k_{\mathsf{out}}+1}$ encrypting the plaintext $P + E_{\mathsf{KS}} \in \mathfrak{R}_{q,N}$ under the secret key $\vec{S}^{[\phi_{\mathsf{out}}]} \in \mathfrak{R}_{q,N}^{k_{\mathsf{out}}}$. The noise E_{KS} added during this procedure, is composed of a rounding error plus a linear combination of the noise from the key switching key ciphertexts. The larger ϕ_{in}, the more significant the rounding error.

Theorem 31 (Noise of GLWE Key Switch) *After performing a key switching (Algorithm 37) taking as input a* GLWE *ciphertext* $\mathsf{CT}_{\mathsf{in}} \in \mathfrak{R}_{q,N}^{k_{\mathsf{in}}+1}$ *under the secret key* $\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]} \in \mathfrak{R}_{q,N}^{k_{\mathsf{in}}}$ *and a key switching key with noise variance* σ_{KSK}^2*, and outputting a* GLWE *ciphertext* $\mathsf{CT}_{\mathsf{out}} \in \mathfrak{R}_{q,N}^{k_{\mathsf{out}}+1}$ *under the secret key* $\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]} \in \mathfrak{R}_{q,N}^{k_{\mathsf{out}}}$*, the variance of the noise of each coefficient of the output can be estimated by*

$$\begin{aligned}\mathsf{Var}\left(\mathsf{CT}_{\mathsf{out}}\right) = \sigma_{\mathsf{in}}^2 + \phi_{\mathsf{in}}\left(\frac{q^2 - \mathfrak{B}^{2\ell}}{12\mathfrak{B}^{2\ell}}\right)\left(\mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right) + \mathbb{E}^2\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)\right)\\ + \frac{\phi_{\mathsf{in}}}{4}\mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right) + \ell k_{\mathsf{in}} N \sigma_{\mathsf{ksk}}^2 \frac{\mathfrak{B}^2 + 2}{12}\end{aligned} \tag{8.1}$$

where $\mathfrak{B}$ *and* ℓ *are the decomposition base and level respectively.*

Proof 31 (Theorem 31) *The proof of Theorem 31 is provided in Appendix A.5.*

Note that when $\phi_{\mathsf{in}} = k_{\mathsf{in}} \cdot N$ *we end up with the same formula than the one for the classical GLWE to GLWE keyswitch (Algorithm 2). In the proof, we decrypt the output ciphertext, extract the error and analyze its variance.*

□

Remark 45 (Cost of a GLWE Key Switch) *We recall that the cost of a GLWE-to-GLWE key switch, which remains the same whether it involves partial secret keys or not is*

$$\begin{aligned}\mathsf{Cost}\left(\mathsf{FftLweKeySwitch}\right) = k_{\mathsf{in}}\ell \cdot \mathsf{Cost}\left(\mathbf{FFT}_N\right) + \left(k_{\mathsf{out}} + 1\right) \cdot \mathsf{Cost}\left(\mathbf{iFFT}_N\right)\\ + N k_{\mathsf{in}}\ell \cdot \left(k_{\mathsf{out}} + 1\right) \cdot \mathsf{Cost}\left(\times_{\mathbb{C}}\right)\\ + N \cdot \left(k_{\mathsf{in}}\ell - 1\right) \cdot \left(k_{\mathsf{out}} + 1\right) \cdot \mathsf{Cost}\left(+_{\mathbb{C}}\right)\end{aligned} \tag{8.2}$$

where $+_{\mathbb{C}}$ *and* $\times_{\mathbb{C}}$ *represent a double-complex addition and multiplication (in the FFT domain) respectively, and* $\mathbf{FFT}_N$ *(resp.* $\mathbf{iFFT}_N$*) the Fast Fourier Transform (resp. inverse FFT).*

Advantages for the Secret Product GLWE Key Switch

A GLWE-to-GLWE key switch computing a product with a secret polynomial, as described in Algorithm 38, follows the exact same definition as above, except that the output ciphertext encrypts $Q \cdot P + E_{\mathsf{KS}}$ where $Q \in \mathfrak{R}_{q,N}$ is the secret polynomial hidden in the keyswitching key. The added noise E_{KS} also depends on the input secret key $\vec{S}^{[\phi_{\mathsf{in}}]}$ and its filling parameter ϕ_{in}. Indeed, this term is the product between the rounding term (dependent on ϕ_{in}) and the polynomial Q.

Algorithm 38: $\mathsf{CT}_{\mathsf{out}} \leftarrow \mathsf{SecretProductGlweKeySwitch}(\mathsf{CT}_{\mathsf{in}}, \mathsf{KSK})$

Context:
$$\begin{cases} \vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]} \in \mathfrak{R}_{q,N}^{k_{\mathsf{in}}} : \text{the input partial secret key (Definition 35)} \\ \vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]} = (S_{\mathsf{in},0}, \cdots, S_{\mathsf{in},k_{\mathsf{in}}-1}) \\ \vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]} \in \mathfrak{R}_{q,N}^{k_{\mathsf{out}}} : \text{the output partial secret key (Definition 35)} \\ (k_{\mathsf{in}}-1)N < \phi_{\mathsf{in}} \leq k_{\mathsf{in}}N \text{ and } (k_{\mathsf{out}}-1)N < \phi_{\mathsf{out}} \leq k_{\mathsf{out}}N \\ Q \in \mathfrak{R}_{q,N} \\ \mathsf{CT}_{\mathsf{in}} = (A_0, \cdots, A_{k_{\mathsf{in}}-1}, B) \in \mathfrak{R}_{q,N}^{k_{\mathsf{in}}+1} \\ \mathsf{CT}_{i,j} \in \mathsf{GLWE}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}\left(\frac{q}{\mathfrak{B}^j} \cdot Q \cdot S_{\mathsf{in},i}\right), \text{ for } 0 \leq i \leq k_{\mathsf{in}}-1 \text{ and } 0 \leq j \leq \ell-1 \\ \mathsf{CT}_{k_{\mathsf{in}},j} \in \mathsf{GLWE}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}\left(\frac{q}{\mathfrak{B}^j} \cdot Q\right), \text{ for } 0 \leq j \leq \ell-1 \\ \ell \in \mathbb{N} : \text{ the number of levels of the decomposition} \\ \mathfrak{B} \in \mathbb{N} : \text{ the base of the decomposition} \end{cases}$$

Input: $\begin{cases} \mathsf{CT}_{\mathsf{in}} \in \mathsf{GLWE}_{\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}}(P), \text{ with } P \in \mathfrak{R}_{q,N} \\ \mathsf{KSK} = \left\{\vec{K}_i = (\mathsf{CT}_{i,0}, \cdots, \mathsf{CT}_{i,\ell-1})\right\}_{0 \leq i \leq k_{\mathsf{in}}} \end{cases}$

Output: $\mathsf{CT}_{\mathsf{out}} \in \mathsf{GLWE}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}(Q \cdot P)$

```
  /* Decompose the $B$ part                                   */
1 Set $\mathsf{CT}_{\mathsf{out}} = \left\langle \vec{K}_{k_{\mathsf{in}}}, \mathsf{Decomp}^{(\mathfrak{B},\ell)}(B) \right\rangle$
2 for $i \in [\![0; k-1]\!]$ do
      /* Decompose the mask                                   */
3     Update $\mathsf{CT}_{\mathsf{out}} = \mathsf{CT}_{\mathsf{out}} - \left\langle \vec{K}_i, \mathsf{Decomp}^{(\mathfrak{B},\ell)}(A_i) \right\rangle$
4 return $\mathsf{CT}_{\mathsf{out}}$
```

Theorem 32 (Noise of Secret-Product GLWE Key Switch) *After performing a Secret-Product key switching (Algorithm 38) taking as input a* GLWE *ciphertext* $\mathsf{CT}_{\mathsf{in}} \in \mathfrak{R}_{q,N}^{k_{\mathsf{in}}+1}$ *under the secret key* $\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]} \in \mathfrak{R}_{q,N}^{k_{\mathsf{in}}}$ *and a key switching key with noise variance* σ_{KSK}^2 *encrypting a secret message* Q*, and outputting a* GLWE *ciphertext* $\mathsf{CT}_{\mathsf{out}} \in \mathfrak{R}_{q,N}^{k_{\mathsf{out}}+1}$ *under the secret key* $\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]} \in \mathfrak{R}_{q,N}^{k_{\mathsf{out}}}$*, the noise variance of each coefficient of the output can be estimated by*

$$\begin{aligned}\mathsf{Var}\left(\mathsf{CT}_{\mathsf{out}}\right) &= \ell(k_{\mathsf{in}}+1)N\sigma_{\mathsf{KSK}}^2\frac{\mathfrak{B}^2+2}{12}\\ &+ \|Q\|_2^2 \cdot \left(\sigma_{\mathsf{in}}^2 + \left(\frac{q^2-\mathfrak{B}^{2\ell}}{12\mathfrak{B}^{2\ell}}\right)\left(1+\phi_{\mathsf{in}}\left(\mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)+\mathbb{E}^2\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)\right)\right)+\frac{\phi_{\mathsf{in}}}{4}\mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)\right) .\end{aligned}$$

Proof 32 (Theorem 32) *The proof of this theorem is very similar to the proof of Theorem 31, it is given in Appendix A.6.*

□

Advantages for an External Product

A GLWE external product is a special case of a secret-product GLWE-to-GLWE key switch, where the input secret key and the output secret key are the same. The external product was reminded in Section 2.3 (Algorithm 7 and Theorem 11). It is pretty easy to establish what noise this procedure will add. The cost of computing a GLWE external product whether it includes a partial secret key or not, is the same.

Theorem 33 (Noise of GLWE External Product) *The external product algorithm is the same as the algorithm of a secret-product* GLWE *key switch (Algorithm 38). The only difference is that the external product uses the same key* $\vec{S}^{[\phi]} \in \mathfrak{R}_{q,N}^{k}$ *as input and as output, and the key switching key is now seen as a* GGSW *ciphertext of message* M_2 *encrypted with noise variance* σ_2^2*. For each coefficient of the output* $\mathsf{CT}_{\mathsf{out}}$*, the noise variance can be estimated by*

$$\begin{aligned}\mathsf{Var}\left(\mathsf{CT}_{\mathsf{out}}\right) &= \ell(k+1)N\sigma_2^2\frac{\mathfrak{B}^2+2}{12}\\ &+ \|M_2\|_2^2 \cdot \left(\sigma_{\mathsf{in}}^2 + \left(\frac{q^2-\mathfrak{B}^{2\ell}}{12\mathfrak{B}^{2\ell}}\right)\left(1+\phi\left(\mathsf{Var}\left(\vec{S}^{[\phi]}\right)+\mathbb{E}^2\left(\vec{S}^{[\phi]}\right)\right)\right)+\frac{\phi}{4}\mathsf{Var}\left(\vec{S}^{[\phi]}\right)\right) .\end{aligned}$$

Proof 33 (Theorem 33) *This proof is the same as the proof of Theorem 32, noting that in this case, we have* $k = k_{\mathsf{in}} = k_{\mathsf{out}}$ *and* $\vec{S}^{[\phi]} = \vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]} = \vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}$*.*

□

Advantages for TFHE's PBS. Using a partial GLWE secret key to encrypt a bootstrapping key for TFHE's programmable bootstrapping enables two convenient features. On the one hand, the output LWE ciphertext is smaller, with less than $k \cdot N + 1$ coefficients, and on the other hand it offers a smaller noise growth in each external product (see Appendix A.7). As explained in Section 2.3, the external product is the main operation used in the cmuxes (Theorem 12 and Algorithm 8) of the blind rotation (Theorem 13 and Algorithm 9). The direct consequence of having smaller output ciphertexts is the fact that we can perform smaller LWE-to-LWE key switches before the next PBS. Furthermore, when $k \cdot N$ is large enough to reach the noise plateau (as explained in Limitation 10), partial secret keys enable to avoid adding unnecessary noise to the bootstrapping.

8.1.2 LWE-to-LWE Key Switch

Partial GLWE secret keys can be used to design a new LWE-to-LWE key switching that is FFT-based. The idea is an adaptation of the work done by Chen *et al.* [Che+20] that was recall in Algorithm 4, but now exploits the use of partial GLWE secret keys. First, one casts the input LWE ciphertext into a GLWE ciphertext using Algorithm 40 so we can apply to it a GLWE-to-GLWE key switching with Algorithm 37 to go to a partial GLWE secret key, and finally compute a sample extract (Algorithm 35). Indeed, the GLWE-to-GLWE key switch can exploit the speed-up coming from the FFT to compute polynomial multiplications. Details about this new LWE-to-LWE key switch are provided in Algorithm 39.

Remark 46 (Inverse Constant Sample Extraction) *Algorithm 40 trivially casts an LWE ciphertext of size $n+1$ into a GLWE ciphertext of size $k+1$ and with polynomials of size N. We obviously need $n \leq kN$. If $n = kN$, the output is a GLWE ciphertext under a traditional secret key, otherwise it is a GLWE ciphertext under a partial GLWE secret key. Note that the constant term of the output GLWE plaintext is exactly the plaintext of the input LWE ciphertext, however the rest of the coefficients of the output GLWE ciphertext are filled with uniformly random values.*

We have the property that for all $p \in \mathbb{Z}_q$, for all $\vec{s} \in \mathbb{Z}_q^n$, for all $\mathsf{ct} \in \mathsf{LWE}_{\vec{s}}(p) \subseteq \mathbb{Z}_q^{n+1}$ and for all $(k, N) \in \mathbb{N}^2$ such that $n \leq kN$:

$$\mathsf{ct} = \mathsf{ConstantSampleExtract}\left(\mathsf{ConstantSampleExtract}^{-1}(\mathsf{ct}, k, N)\right).$$

Algorithm 39: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{FftLweKeySwitch}(\mathsf{ct}_{\mathsf{in}}, \mathsf{KSK})$

```
Context: { $n_{\mathsf{in}} \leq k_{\mathsf{in}} \cdot N, \quad n_{\mathsf{out}} \leq k_{\mathsf{out}} \cdot N$
         { $\vec{s}_{\mathsf{in}} = (s_0, \cdots, s_{n_{\mathsf{in}}-1}) \in \mathbb{Z}_q^{n_{\mathsf{in}}}$ : the input LWE secret key
         { $\vec{s}_{\mathsf{out}} = \left(s'_0, \cdots, s'_{n_{\mathsf{out}}-1}\right) \in \mathbb{Z}_q^{n_{\mathsf{out}}}$ : the output LWE secret key
         { $\ell \in \mathbb{N}$ : the number of levels of the decomposition
         { $\mathfrak{B} \in \mathbb{N}$ : the base of the decomposition
         { $\vec{S}^{[n_{\mathsf{in}}]}_{\mathsf{in}} = \left(S_{\mathsf{in},0}, \cdots, S_{\mathsf{in},k_{\mathsf{in}}-1}\right) \in \mathfrak{R}_{q,N}^{k_{\mathsf{in}}}$ : a partial secret
         {       key (Definition 35) such that its flattened version is $\vec{s}_{\mathsf{in}}$
         { $\vec{S}^{[n_{\mathsf{out}}]}_{\mathsf{out}} = \left(S_{\mathsf{out},0}, \cdots, S_{\mathsf{out},k_{\mathsf{out}}-1}\right) \in \mathfrak{R}_{q,N}^{k_{\mathsf{out}}}$ : a partial secret
         {       key (Definition 35) such that its flattened version is $\vec{s}_{\mathsf{out}}$
Input: { $\mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}_{\mathsf{in}}}(p) \subseteq \mathbb{Z}_q^{n_{\mathsf{in}}+1}$, with $p \in \mathbb{Z}_q$
       { $\mathsf{KSK} = \{\mathsf{KSK}_i\}_{0 \leq i < k_{\mathsf{in}}}$, with $\mathsf{KSK}_i \in \mathsf{GLev}_{\vec{S}^{[\phi_{\mathsf{out}}]}_{\mathsf{out}}}(S_{\mathsf{in},i})$ : a key switching key
Output: $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s}_{\mathsf{out}}}(p) \subseteq \mathbb{Z}_q^{n_{\mathsf{out}}+1}$
   /* Inverse of a constant sample extraction (Algorithm 40)                    */
1  Set $\mathsf{CT} \leftarrow \mathsf{ConstantSampleExtraction}^{-1}(\mathsf{ct}_{\mathsf{in}}, k_{\mathsf{in}}, N) \in \mathfrak{R}_{q,N}^{k_{\mathsf{in}}+1}$
   /* GLWE-to-GLWE key switch based on the FFT (Algorithm 37)                   */
2  Set $\mathsf{CT}' \leftarrow \mathsf{GlweKeySwitch}(\mathsf{CT}, \mathsf{KSK}) \in \mathfrak{R}_{q,N}^{k_{\mathsf{out}}+1}$
   /* Constant sample extraction (Algorithm 35)                                 */
3  Set $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{ConstantSampleExtract}(\mathsf{CT}') \in \mathbb{Z}_q^{n_{\mathsf{out}}+1}$
4  return $\mathsf{ct}_{\mathsf{out}}$
```

Algorithm 40: $\mathsf{CT}_{\mathsf{out}} \leftarrow \mathsf{ConstantSampleExtract}^{-1}(\mathsf{ct}_{\mathsf{in}}, k, N)$

```
Context: { $\vec{s} \in \mathbb{Z}_q^n$ : the input LWE secret key
         { $\vec{S}^{[n]} \in \mathfrak{R}_{q,N}^k$ : a partial secret key (Definition 35)
         {      such that its flattened version is $\vec{s}$ (Definition 36)
         { $R := \sum_{i=1}^{N-1} r_i \cdot X^i \in \mathfrak{R}_{q,N}$, where $r_i$ are uniformly random
         { $\mathsf{ct}_{\mathsf{in}} = (a_0, \cdots, a_{n-1}, b) \in \mathbb{Z}_q^{n+1}$
         { $p \in \mathbb{Z}_q$
Input: { $\mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}}(p)$ : an LWE encryption of the plaintext $p$
       { $k \in \mathbb{N}$ : the output GLWE dimension
       { $N \in \mathbb{N}$ : the output polynomial size
Output: $\mathsf{CT}_{\mathsf{out}} \in \mathsf{GLWE}_{\vec{S}^{[n]}}(p_0 + R)$ : a GLWE encryption
   /* put the $b$ part in a polynomial                                          */
1  set $B' := b \in \mathfrak{R}_{q,N}$
   /* put the rest in polynomials                                               */
2  for $i \in [\![0; k \cdot N]\!]$ do
3      set $\alpha := \left\lfloor \frac{i}{N} \right\rfloor$, $\beta := (N - i) \mod N$ and $\gamma := 1 - (\beta == 0)$
4      if $i \leq \phi - 1$ then
5          set $a'_{\alpha,\beta} := (-1)^{\gamma} \cdot a_i$
6      else
7          set $a'_{\alpha,\beta} := 0$
8  return $\mathsf{CT}_{\mathsf{out}} := \left(A'_0 := \sum_{j=0}^{N-1} a'_{0,j} X^j, \cdots, A'_{k-1} := \sum_{j=0}^{N-1} a'_{k-1,j} X^j, B'\right) \in \mathfrak{R}_{q,N}^{k+1}$
```

Theorem 34 (Noise & Cost of FFT-Based LWE Key Switch) *We consider the new LWE-to-LWE key switch as described in Algorithm 39. Its cost is the same as the cost of a GLWE-to-GLWE key switch as introduced in Remark 45 i.e,* $\mathsf{Cost}\left(\mathsf{FftLweKeySwitch}\right) = \mathsf{Cost}\left(\mathsf{GlweKeySwitch}\right)$.

The output noise can be expressed from the noise formula of the GLWE-to-GLWE key switch (Theorem 31). To sum up, the output noise is:

$$\mathsf{Var}\left(\mathsf{FftLweKeySwitch}\right) = \mathsf{FftError}\left(k_{\mathsf{max}}, N, \beta, \ell\right) + \mathsf{Var}\left(\mathsf{GlweKeySwitch}\right)$$

with $\phi_{\mathsf{in}} = n_{\mathsf{in}}$*,* $\phi_{\mathsf{out}} = n_{\mathsf{out}}$*,* $k_{\mathsf{max}} = \max\left(k_{\mathsf{in}}, k_{\mathsf{out}}\right)$ *and* $\mathsf{FftError}_{k_{\mathsf{max}},N,\beta,\ell}$ *being the error added by the FFT conversions (see Section 3.3).*

Proof 34 (Theorem 34) *Expressing the cost is quite straightforward, since we can neglect the complexity of the sample extraction and its inverse. The estimation of the variance of the error is immediate as well. We use the corrective formula introduced in Section 3.3 to estimate an upper bound on the FFT error. Indeed, it is easy to see that the FFT-based LWE key switch with* k_{in} *and* k_{out} *is a special case of an external product with* $k = \max\left(k_{\mathsf{in}}, k_{\mathsf{out}}\right)$ *where some of the ciphertexts composing the* GGSW *are trivial encryptions of 0 or 1 (no noise, all mask elements set to zero and the plaintext put in the b/B part).*

$\square$

Practical Improvement. The use of partial secret keys brings a significant improvement to homomorphic computations. Figure 8.2 presents a comparison of our techniques and the state of the art [CJP21]. More details on the experiments are reported in Section 8.3.

8.2 Shared Randomness

In the previous chapters, we introduced several types of atomic patterns (Definition 28) and we showed in Section 4.3 by comparing an atomic pattern of type $\mathcal{A}^{(\text{CJP21})}$ (Definition 30) to an atomic pattern of type $\mathcal{A}^{(\text{KS-free})}$ that the key switch was very important to obtain an efficient bootstrap. In this context, a key switch changes the secret key of a ciphertext to a smaller secret key. Intuitively, a key switch performs an homomorphic decryption of the ciphertext by removing the dot product between the mask and the secret key. In practice, it means that the cost and the noise of a key switch (Theorem 6) depends on the input LWE dimension and the output LWE dimension (or polynomial size and GLWE dimension).

With the new keys introduced in this section, the cost and the noise of a key switch will no longer be dependent on the input LWE dimension and the output LWE dimension but on the output LWE dimension and the difference between the input LWE dimension and the output LWE dimension. In the special case where the output LWE dimension is larger than the input LWE dimension, the key switch will come for free: it will add no noise and will have no cost.

Instead of sampling every secret key independently, we may also consider generating a list of α nested GLWE keys with the same level of security λ. It is then public knowledge that all the secret coefficients of a smaller key will be included into a larger secret key.

To give a simple example, consider three integers $1 < n_0 < n_1 < n_2$ and a secret key $\vec{s}^{(2)} \in \mathbb{Z}_q^{n_2}$ generated in the traditional manner (either sampled from an uniform binary/ternary, or a small Gaussian). Let us write it as a concatenation of 3 vectors: $\vec{s}^{(2)} = \vec{r}^{(0)}||\vec{r}^{(1)}||\vec{r}^{(2)}$. We can now build two smaller secret keys out of $\vec{s}^{(2)}$ such that for all pairs of nested keys, the small one is included in the large one, as its first coefficients:

$$\vec{s}^{(0)} = \vec{r}^{(0)} \in \mathbb{Z}_q^{n_0} \quad \text{and} \quad \vec{s}^{(1)} = \vec{r}^{(0)}||\vec{r}^{(1)} \in \mathbb{Z}_q^{n_1}.$$

This form of secret keys is extremely useful for the key switching and bootstrapping procedures. Note that each of those secret keys uses a different variance for the noise added during encryption; the smaller the secret key, the bigger the variance, so they can all guarantee the same level of security λ.

Definition 37 (GLWE Secret Keys with Shared Randomness) *Two GLWE secret keys $\vec{S} \in \mathfrak{R}_{q,N}^k$ and $\vec{S}' \in \mathfrak{R}_{q,N'}^{k'}$, with $kN \leq k'N'$, are said to be with shared randomness if we have that for all $0 \leq i < kN$, $\bar{s}_i = \bar{s}'_i$, where $\bar{s}_i$ and $\bar{s}'_i$ respectively come from the flattened view (Definition 11) of $\vec{S}$ and $\vec{S}'$. We note by $\vec{S} \prec \vec{S}'$ this relationship between secret keys.*

The security of the shared randomness secret keys is detailed in [Ber+23b] and fells outside of the scope of this thesis.

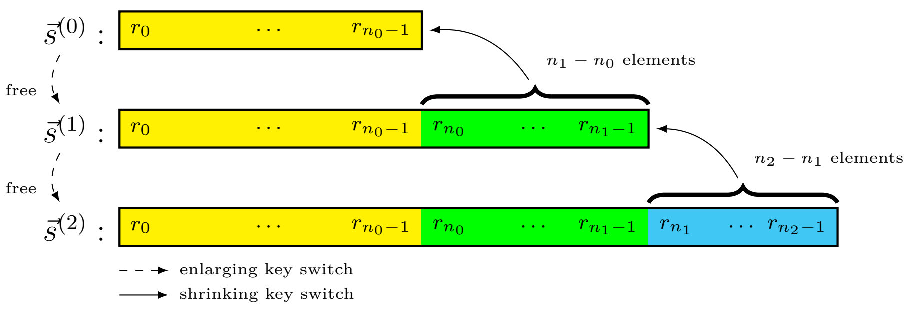


Figure 8.1: Illustration of simplified key switch procedures between three LWE secret keys with shared randomness.

Using shared randomness secret keys enables to speed up homomorphic computations and reduce the amount of noise added by these operations. This is particularly useful for LWE-to-LWE key switch procedures.

8.2.1 Advantages for LWE-to-LWE Key Switch

Secret keys with shared randomness enable to key switch more efficiently and add less noise during the procedure. Figure 8.1 illustrates key switching processes between three LWE secret keys with shared randomness. A key switch from a key to a bigger key is represented with dotted arrows and is called *enlarging key switch*. A key switch from a key to a smaller key is represented with solid arrows and is called *shrinking key switch*.

Enlarging Key Switch. When we consider a ciphertext $\mathsf{ct}_{\mathsf{in}} = (a_0, \cdots, a_{n_1-1}, b) \in \mathsf{LWE}_{\vec{s}^{(1)}}(m) \subseteq \mathbb{Z}_q^{n_1+1}$ under the secret key $\vec{s}^{(1)} \in \mathbb{Z}_q^{n_1}$ and want to key switch it to the secret key $\vec{s}^{(2)} \in \mathbb{Z}_q^{n_2}$, where $\vec{s}^{(1)} \prec \vec{s}^{(2)}$, the algorithm translates into simply adding zeros at the end of the ciphertext:

$$\mathsf{ct}_{\mathsf{out}} := (a_0, \cdots, a_{n_1-1}, 0, \cdots, 0, b) \in \mathsf{LWE}_{\vec{s}^{(2)}}(m) \subseteq \mathbb{Z}_q^{n_2+1}$$

Algorithm 41 describes this procedure in detail. In this thesis, we only use this algorithm with LWE ciphertexts, but it can trivially be extended to GLWE ciphertexts as well.

To sum up, with secret keys with shared randomness, the enlarging key switchess are basically zero-cost operations and they do not require the use of a public key. They also add no noise, instead of adding a linear combination of freshly encrypted ciphertexts

```
Algorithm 41: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{EnlargingKeySwitch}(\mathsf{ct}_{\mathsf{in}})$
Context: $\begin{cases} \vec{s}_{\mathsf{in}} \in \mathbb{Z}_q^{n_{\mathsf{in}}} : \text{the input secret key} \\ \vec{s}_{\mathsf{out}} \in \mathbb{Z}_q^{n_{\mathsf{out}}} : \text{the output secret key} \\ \vec{s}_{\mathsf{in}} \prec \vec{s}_{\mathsf{out}} : \text{ secret keys with shared randomness (Definition 37)} \\ p \in \mathbb{Z}_q \\ \mathsf{ct}_{\mathsf{in}} = (a_0, \cdots, a_{n_{\mathsf{in}}-1}, b) \in \mathbb{Z}_q^{n_{\mathsf{in}}+1} \end{cases}$
Input: $\mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}_{\mathsf{in}}}(p)$
Output: $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s}_{\mathsf{out}}}(p)$
/* Pad with zeros between the mask and the $b$ part */
1 Set $\mathsf{ct}_{\mathsf{out}} := (a_0, \cdots, a_{n-1}, 0, \cdots, 0, b) \in \mathbb{Z}_q^{n_{\mathsf{out}}+1}$
2 return $\mathsf{ct}_{\mathsf{out}}$
```

under $\vec{s}^{(2)}$.

Theorem 35 (Cost & Noise of Enlarging Key Switching) *When working with secret keys with shared randomness, the cost of an enlarging key switching (Algorithm 41) is reduced to zero, and the noise in the output is the same as the one in the input.*

Proof 35 (Theorem 35) *The proof of this theorem is trivial.*

□

Shrinking Key Switch. When we consider a ciphertext $\mathsf{ct}_{\mathsf{in}} = (a_0, \cdots, a_{n_1-1}, b) \in \mathbb{Z}_q^{n_1+1}$ under the secret key $\vec{s}^{(1)} \in \mathbb{Z}_q^{n_1}$ and we want to key switch it to the secret key $\vec{s}^{(0)} \in \mathbb{Z}_q^{n_0}$, where $\vec{s}^{(0)} \prec \vec{s}^{(1)}$ and $\vec{s}^{(1)} = \vec{s}^{(0)}||\vec{r}^{(1)}$, the algorithm is simplified precisely because of the shared randomness property:

1. the parts $(a_0, \cdots, a_{n_0-1})$ and b do not need to be processed but simply reorganized into a temporary ciphertext: $\mathsf{ct} = (a_0, \cdots, a_{n_0-1}, b) \in \mathbb{Z}_q^{n_0+1}$,
2. the part $(a_{n_0}, \cdots, a_{n_1-1})$ has to be key switched, which can be somehow viewed as a traditional key switching algorithm: i.e., key switching the ciphertext $(a_{n_0}, \cdots, a_{n_1-1}, 0) \in \mathbb{Z}_q^{n_1-n_0+1}$ with a key switching key going from the secret key $\vec{r}^{(1)}$ to $\vec{s}^{(0)}$, and at the end, adding it to ct and returning the result.

Algorithm 42 describes this procedure in detail. In this thesis, we only use this algorithm with LWE ciphertexts, but it can also be trivially extended to GLWE ciphertexts.

To sum up, with secret keys with shared randomness, the shrinking key switch requires smaller key switching keys: their size becomes proportional to $n_2 - n_1$ instead of n_2. As

Algorithm 42: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{ShrinkingKeySwitch}(\mathsf{ct}_{\mathsf{in}}, \mathsf{KSK})$

```
Context: $\begin{cases} \vec{s}_{\mathsf{in}} = (s_0, \cdots, s_{n_{\mathsf{in}}-1}) \in \mathbb{Z}_q^{n_{\mathsf{in}}} : \text{the input secret key} \\ \vec{s}_{\mathsf{out}} \in \mathbb{Z}_q^{n_{\mathsf{out}}} : \text{the output secret key} \\ n_{\mathsf{out}} < n_{\mathsf{in}} \\ \vec{s}_{\mathsf{out}} \prec \vec{s}_{\mathsf{in}} : \text{ secret keys with shared randomness (Definition 37)} \\ p \in \mathbb{Z}_q \\ \mathsf{ct}_{\mathsf{in}} = (a_0, \cdots, a_{n_{\mathsf{in}}-1}, b) \in \mathbb{Z}_q^{n_{\mathsf{in}}+1} \\ \overline{\mathsf{CT}}_i \in \mathsf{GLev}_{\vec{s}}^{\mathfrak{B},\ell}(s_i), \text{ for } n_{\mathsf{out}} \leq i \leq n_{\mathsf{in}} - 1 \\ \ell \in \mathbb{N} : \text{ the number of levels of the decomposition} \\ \mathfrak{B} \in \mathbb{N} : \text{ the base of the decomposition} \end{cases}$
Input: $\begin{cases} \mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}_{\mathsf{in}}}(p) \\ \mathsf{KSK} = \left\{\overline{\mathsf{CT}}_i\right\}_{n_{\mathsf{out}} \leq i \leq n_{\mathsf{in}}-1} \end{cases}$
Output: $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s}_{\mathsf{out}}}(p)$
/* Keep the beginning of the mask and the $b$ part */
1 Set $\mathsf{ct}_{\mathsf{out}} := (a_0, \cdots, a_{n_{\mathsf{out}}-1}, b) \in \mathbb{Z}_q^{n_{\mathsf{out}}+1}$
2 for $i \in [\![n_{\mathsf{out}}; n_{\mathsf{in}} - 1]\!]$ do
      /* Decompose the rest of the mask */
3     Update $\mathsf{ct}_{\mathsf{out}} = \mathsf{ct}_{\mathsf{out}} - \left\langle \overline{\mathsf{CT}}_i, \mathsf{dec}^{(\mathfrak{B},\ell)}(a_i) \right\rangle$
4 return $\mathsf{ct}_{\mathsf{out}}$
```

a consequence, the computation is faster, equivalent to key switch a ciphertext of size $n_2 - n_1 + 1$ instead of $n_2 + 1$. Finally, the noise in the output is also smaller because the algorithm involves a smaller linear combination of freshly encrypted ciphertexts under $\vec{s}^{(1)}$.

Theorem 36 (Cost & Noise of Shrinking Key Switching) *Consider two secret keys with shared randomness $\vec{s}^{(0)} \prec \vec{s}^{(1)}$ with $\vec{s}^{(0)} \in \mathbb{Z}_q^{n_0}$, $\vec{s}^{(1)} \in \mathbb{Z}_q^{n_1}$ and $1 < n_0 < n_1$. Let $\mathfrak{B} \in \mathbb{N}^*$ and $\ell \in \mathbb{N}^*$ be the decomposition base and level used in key switching. The cost of our shrinking key switching (Algorithm 42) is $\ell (n_1 - n_0)(n_0 + 1)$ integer multiplications and $(\ell (n_1 - n_0) - 1)(n_0 + 1)$ integer additions.*

The noise added by the procedure satisfies

$$\mathsf{Var}(\mathsf{ShrinkingKeySwitch}) = (n_1 - n_0)\left(\frac{q^2 - \mathfrak{B}^{2\ell}}{12\mathfrak{B}^{2\ell}}\right)\left(\mathsf{Var}(\vec{s}_{\mathsf{in}}) + \mathbb{E}^2(\vec{s}_{\mathsf{in}})\right) + \frac{(n_1 - n_0)}{4}\mathsf{Var}(\vec{s}_{\mathsf{in}}) + \ell \cdot (n_1 - n_0) \cdot \frac{\mathfrak{B}^2 + 2}{12}\sigma_{\mathsf{KSK}}^2 .$$

The size of the shrinking key switching key can be obtained using the formula for the *size of the classical keyswitching key with $n = n_{\mathsf{in}} - n_{\mathsf{out}}$.*

Proof 36 (Theorem 36) *The details of this proof can be found in Appendix A.8.*

□

8.2.2 Stair Key Switch

In Section 8.2.1, we saw that when one uses different secret keys within an FHE circuit, it is convenient to make use of the secret keys with shared randomness. However, this concept can also be used locally inside a key switch procedure to explore a cost/noise trade-off.

For simplicity, let us consider an FHE use case where there are only two LWE secret keys, and only a key switch from the big one to the small one. We start by setting the two secret keys as ones with shared randomness. The idea here is to add one or several secret keys with shared randomness, only during the key switch procedure.

For example, let us assume a fixed decomposition base $\mathfrak{B}$, a fixed number of levels ℓ and let $\vec{s}^{(2)}$ be our big secret key and $\vec{s}^{(0)}$ be our small (as defined in Section 8.2.1). To key switch from $\vec{s}^{(2)}$ to $\vec{s}^{(0)}$, this time we will add one intermediate secret key with shared randomness $\vec{s}^{(1)}$ and compute first a key switch from $\vec{s}^{(2)}$ to $\vec{s}^{(1)}$ and then a key switch from $\vec{s}^{(1)}$ to $\vec{s}^{(0)}$. This algorithm will be more costly than a key switch with secret key with shared randomness as presented in Algorithm 42 because its first part will be a linear combination of $(n_2 - n_1)$ ciphertexts of size $n_1 + 1$, and its second part a linear combination of $(n_1 - n_0)$ ciphertexts of smaller size $n_0 + 1$, instead of having a single linear combination of $n_2 - n_0$ ciphertexts of size $n_0 + 1$. The total number of ciphertexts in the linear combination and in the key switching key has not changed ($n_2 - n_1 + n_1 - n_0 = n_2 - n_0$ as in the key switch from $\vec{s}^{(2)}$ to $\vec{s}^{(0)}$), but the linear combinations are slightly more costly and the ciphertexts composing the key switching keys slightly larger. On the other hand, this algorithm produces less noise: indeed its first part has ciphertexts with lower noise because they are encrypted under a bigger secret key.

Here is the trade-off we want to study. The extreme is to go from $\vec{s}^{(\mathsf{nb})}$ to $\vec{s}^{(0)}$ by key switching one element of the key in each key switch, meaning that we will have a total number of $\mathsf{nb} = n_{\mathsf{nb}} - n_0$ shrinking key switches (Algorithm 42) to perform. So nb corresponds to the steps in the staircase. This means considering a total number of shared keys equal to $\mathsf{nb} + 1$, including the secret keys $\vec{s}^{(\mathsf{nb})}$ and $\vec{s}^{(0)}$ which are the end points of the staircase. We call the added keys between $\vec{s}^{(\mathsf{nb})}$ and $\vec{s}^{(0)}$ intermediate secret keys, so we have a total of $\mathsf{nb} - 1$ intermediate secret keys. In practice, we start with coefficient $a_{n_{\mathsf{nb}}-1}$ and key switch it to the secret key with $n_{\mathsf{nb}} - 1$ elements, add it to the rest, and do the same with the next last element, and so on until we reach the desired secret key, one coefficient at a time. The other extreme case is when we key switch directly from $\vec{s}^{(1)}$ and $\vec{s}^{(0)}$ without intermediary key switches, so $\mathsf{nb} = 1$ as presented in Algorithm 42.

Algorithm 43 gives details about this procedure. It is important to point out that there are now nb pairs of decomposition parameters $(\mathfrak{B}_\alpha, \ell_\alpha)$ for $0 \le \alpha \le \mathsf{nb} - 1$, one for each step of the staircase. Note that we could also allow to have more than one such couple per step as well.

Algorithm 43: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{StairKeySwitch}\left(\mathsf{ct}_{\mathsf{in}}, \{\mathsf{KSK}_\alpha\}_{0\le\alpha\le\mathsf{nb}-1}\right)$

```
Context: { nb ∈ ℕ : the number of steps of the algorithm
         { $n_0 < n_1 < \cdots < n_{\mathsf{nb}}$
         { $\vec{s}^{(\mathsf{nb})} \in \mathbb{Z}_q^{n_{\mathsf{nb}}}$ : the input secret key
         { $\vec{s}^{(0)} \in \mathbb{Z}_q^{n_0}$ : the output secret key
         { $\vec{s}^{(\alpha)} \in \mathbb{Z}_q^{n_\alpha}, \forall 1 \le \alpha \le \mathsf{nb}-1$ : intermediate secret keys
         { $\vec{s}^{(0)} \prec \vec{s}^{(1)} \prec \cdots \prec \vec{s}^{(\mathsf{nb})}$ : shared randomness secret keys (Definition 37)
Input: { $\mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}_{\vec{s}^{(\mathsf{nb})}}(p) \subseteq \mathbb{Z}_q^{n_{\mathsf{nb}}+1}$, with $p \in \mathbb{Z}_q$
       { $\{\mathsf{KSK}_\alpha\}_{0\le\alpha\le\mathsf{nb}-1}$ : intermediate key switching key as in Algorithm 42
       {                                   where $\mathsf{KSK}_\alpha$ switches from $\vec{s}^{(\alpha+1)}$ to $\vec{s}^{(\alpha)}$
Output: $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s}^{(0)}}(p) \subseteq \mathbb{Z}_q^{n_0+1}$
  /* Set the counter to go from $\mathsf{nb}-1$ to $0$ */
1 Set $\alpha := \mathsf{nb} - 1$
  /* Set the initial ciphertext */
2 Set $\mathsf{ct} := \mathsf{ct}_{\mathsf{in}}$
3 while $\alpha >= 0$ do
      /* Call to Algorithm 42 */
4     Update $\mathsf{ct} \leftarrow \mathsf{ShrinkingKeySwitch}(\mathsf{ct}, \mathsf{KSK}_\alpha) \in \mathsf{LWE}_{\vec{s}^{(\alpha)}}(p) \subseteq \mathbb{Z}_q^{n_\alpha+1}$
5     $\alpha := \alpha - 1$
6 return $\mathsf{ct}_{\mathsf{out}} := \mathsf{ct}$
```

Theorem 37 (Cost & Noise of Stair Shrinking Key Switching)
Consider the stair key switch as detailed in Algorithm 43. Its cost amounts to $\sum_{\alpha=0}^{\mathsf{nb}-1} \ell_\alpha (n_{\alpha+1} - n_\alpha)(n_\alpha + 1)$ *integer multiplications and* $\sum_{\alpha=0}^{\mathsf{nb}-1} (\ell_\alpha (n_{\alpha+1} - n_\alpha) - 1)(n_\alpha + 1)$ *integer additions.*

The noise added by the procedure satisfies

$$\mathsf{Var}(\mathsf{StairShrinkKS}) = \sum_{\alpha=0}^{\mathsf{nb}-1} (n_{\alpha+1} - n_\alpha) \left(\frac{q^2 - \mathfrak{B}_\alpha^{2\ell_\alpha}}{12\mathfrak{B}_\alpha^{2\ell_\alpha}} \right) \left(\mathsf{Var}\left(\vec{s}^{(\alpha+1)}\right) + \mathbb{E}^2\left(\vec{s}^{(\alpha+1)}\right) \right) + \frac{(n_{\alpha+1} - n_\alpha)}{4} \mathsf{Var}\left(\vec{s}^{(\alpha+1)}\right) + \ell_\alpha \cdot (n_{\alpha+1} - n_\alpha) \cdot \frac{\mathfrak{B}_\alpha^2 + 2}{12} \sigma^2_{\mathsf{KSK}_\alpha} \ .$$

The size (in bits) of the Stair keyswitching key $\mathsf{Size}\left(\mathsf{Stair\text{-}KS}\right)$ *with* 2 *steps is*

$$\mathsf{Size}\left(\mathsf{Stair\text{-}KS}\right) := \left((n_{\mathsf{KS}} + 1) \cdot \ell_{\mathsf{KS}_1} \cdot (\phi - n_{\mathsf{KS}}) + (n+1) \cdot \ell_{\mathsf{KS}_2} \cdot (n_{\mathsf{KS}} - n) \right) \cdot \lceil \log_2(q) \rceil \ .$$

Proof 37 (Theorem 37) *The cost and noise of the stair shrinking key switch is trivially deduced from Theorem 36. At step α of the loop in Algorithm 43, the cost of the shrinking key switching is $\ell_\alpha \left(n_{\alpha+1} - n_\alpha\right)\left(n_\alpha + 1\right)$ integer multiplications and $\left(\ell_\alpha \left(n_{\alpha+1} - n_\alpha\right) - 1\right)\left(n_\alpha + 1\right)$ integer additions.*

The variance of the noise added at step α is

$$\mathsf{Var}(\mathsf{ShrinkKS}_\alpha) = (n_{\alpha+1} - n_\alpha) \left(\frac{q^2 - \mathfrak{B}_\alpha^{2\ell_\alpha}}{12\mathfrak{B}_\alpha^{2\ell_\alpha}} \right) \left(\mathsf{Var}\left(\vec{s}^{(\alpha+1)}\right) + \mathbb{E}^2\left(\vec{s}^{(\alpha+1)}\right) \right) + \frac{(n_{\alpha+1} - n_\alpha)}{4} \mathsf{Var}\left(\vec{s}^{(\alpha+1)}\right) + \ell_\alpha \cdot (n_{\alpha+1} - n_\alpha) \cdot \frac{\mathfrak{B}_\alpha^2 + 2}{12} \sigma^2_{\mathsf{KSK}_\alpha} \ .$$

To obtain the total cost of the algorithm and the total variance of the noise added, we simply iterate from $\alpha = 0$ to $\mathsf{nb} - 1$.

□

Remark 47 (Staircase in the Blind Rotation.) *A similar process can be introduced in the blind rotation algorithm. The idea would be, during the blind rotation, to progressively use GLWE partial secret keys (Definition 35) with a smaller filling parameter ϕ which will reduce the output noise of the blind rotate. As with the stair shrinking key switch, we could use different bases and levels in the external products, thus potentially offering an overall speed-up. We leave this problem as a topic for future work.*

Practical Improvement. The use of shared secret keys brings a significant improvement to homomorphic computations. Figure 8.3 presents a comparison of our techniques to the state of the art [CJP21]. More detailed experiments are reported in Section 8.3.5.

8.3 Combining Partial Keys & Shared Randomness

In this section, we start by providing details on FHE algorithms that benefit from having secret keys that are both partial (Section 8.1) and shared randomness (Section 8.2). Later on, we describe some useful applications of this new type of secret keys.

8.3.1 Combining Both Techniques

Partial GLWE secret keys with shared randomness are simply a list of partial GLWE secret keys (Section 8.1) with shared coefficients in the exact same way as in Section 8.2. This type of keys is a combination of secret keys with shared randomness and partial secret keys, offering advantages of both types.

It is possible to design a faster shrinking key switch (Algorithm 39) which uses partial secret keys (Definition 35). This means that for this faster algorithm, we use both partial secret keys and secret keys with shared randomness. Details about this new procedure are given in Algorithm 44.

```
Algorithm 44: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{FftShrinkingKeySwitch}(\mathsf{ct}_{\mathsf{in}}, \mathsf{KSK})$
Context: $\begin{cases} n_{\mathsf{out}} < n_{\mathsf{in}},\ \ n_{\mathsf{in}} - n_{\mathsf{out}} \leq k_{\mathsf{KSK,in}} \cdot N_{\mathsf{KSK}} \text{ and } n_{\mathsf{out}} \leq k_{\mathsf{KSK,out}} \cdot N_{\mathsf{KSK}} \\ \vec{s}_{\mathsf{out}} \prec \vec{s}_{\mathsf{in}} : \text{ secret keys with shared randomness (Definition 37)} \\ \vec{s}_{\mathsf{out}} \in \mathbb{Z}_q^{n_{\mathsf{out}}} : \text{the output LWE secret key} \\ \vec{s} = (s_{n_{\mathsf{out}}}, \cdots, s_{n_{\mathsf{in}}-1}) \in \mathbb{Z}_q^{n_{\mathsf{in}}-n_{\mathsf{out}}} \\ \vec{s}_{\mathsf{in}} = \vec{s}_{\mathsf{out}} || \vec{s} \in \mathbb{Z}_q^{n_{\mathsf{in}}} : \text{the input LWE secret key} \end{cases}$
Input: $\begin{cases} \mathsf{ct}_{\mathsf{in}} = (a_0, \cdots, a_{n_{\mathsf{in}}-1}, b) \in \mathsf{LWE}_{\vec{s}_{\mathsf{in}}}(p) \subseteq \mathbb{Z}_q^{n_{\mathsf{in}}+1}, \text{ where } p \in \mathbb{Z}_q \\ \mathsf{KSK} : \text{ the key switching key suited for Algorithm 39} \end{cases}$
Output: $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}_{\vec{s}_{\mathsf{out}}}(p)$
   /* Split the input LWE ciphertext into two parts: one related to $\vec{s}_{\mathsf{out}}$, and the
      rest                                                                          */
1  Set $\mathsf{ct}_0 := (a_0, \cdots, a_{n_{\mathsf{out}}-1}, b) \in \mathbb{Z}_q^{n_{\mathsf{out}}+1}$
2  Set $\mathsf{ct}_1 := (a_{n_{\mathsf{out}}}, \cdots, a_{n_{\mathsf{in}}-1}, 0) \in \mathbb{Z}_q^{n_{\mathsf{in}}-n_{\mathsf{out}}+1}$
   /* Call Algorithm 39                                                             */
3  Set $\mathsf{ct}'_1 \leftarrow \mathsf{FftLweKeySwitch}(\mathsf{ct}_1, \mathsf{KSK}) \in \mathbb{Z}_q^{n_{\mathsf{out}}+1}$
4  return $\mathsf{ct}_{\mathsf{out}} = \mathsf{ct}_0 + \mathsf{ct}'_1$
```

Theorem 38 (Noise & Cost of the FFT-Based Shrinking Key Switch) *We consider the FFT-based LWE shrinking key switch as detailed in Algorithm 44. Its cost can be expressed from the cost of a GLWE-to-GLWE key switch (Remark 45) since we neglect the* *costs of sample extraction and its inverse. The cost is then* $\mathsf{Cost}\left(\mathsf{FftShrinkingKeySwitch}\right) = \mathsf{Cost}\left(\mathsf{GlweKeySwitch}\right)$*. Note that* k_{in} *is smaller thanks to the shared randomness property of the secret keys, which leads to a faster procedure.*

The added noise can be expressed from the noise formula of the GLWE-to-GLWE key switch (Theorem 31) which gives $\mathsf{Var}\left(\mathsf{FftShrinkingKeySwitch}\right) = \mathsf{FftError}\left(k_{\mathsf{max}}, N, \mathfrak{B}, \ell\right) + \mathsf{Var}\left(\mathsf{GlweKeySwitch}\right)$ *with* $\phi_{\mathsf{in}} = n_{\mathsf{out}} - n_{\mathsf{in}}$ *and* $k_{\mathsf{max}} = \max\left(k_{\mathsf{in}}, k_{\mathsf{out}}\right)$*.*

The size (in bits) of the FFT-Shrinking keyswitching key $\mathsf{Size}\left(\mathsf{FFT\text{-}KS}\right)$ *is*

$$\mathsf{Size}\left(\mathsf{FFT\text{-}KS}\right) := \left(k_{\mathsf{out}} + 1\right) \cdot \ell_{\mathsf{KS}} \cdot k_{\mathsf{in}} \cdot N \cdot \lceil \log_2(q) \rceil \ .$$

Proof 38 (Theorem 38) *The estimation of the variance of the error is immediate. For the FFT error, we refer to Section 3.3 and Proof 34.*

□

We can also adapt the GLWE keyswitch introduced in Algorithm 2 with partial and shared randomness key as detailed in Algorithm 45.

Theorem 39 (Noise of GLWE Key Switching With Partial Keys With Shared Randomness) *Perform a key switch (Algorithm 45) from* $\mathsf{CT}_{\mathsf{in}} \in \mathfrak{R}_{q,N}^{k_{\mathsf{in}}+1}$ *under the secret key* $\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]} \in \mathfrak{R}_{q,N}^{k_{\mathsf{in}}}$*, to* $\mathsf{CT}_{\mathsf{out}} \in \mathfrak{R}_{q,N}^{k_{\mathsf{out}}+1}$ *under the secret key* $\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]} \in \mathfrak{R}_{q,N}^{k_{\mathsf{out}}}$*, where* $\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]} \prec \vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}$*. Each coefficient of the output has added noise estimated as*

$$\begin{aligned}\mathsf{Var}(\mathsf{GlweKeySwitch}') = (\phi_{\mathsf{in}} - \phi_{\mathsf{out}}) \left(\frac{q^2 - \mathfrak{B}^{2\ell}}{12\mathfrak{B}^{2\ell}}\right) \left(\mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right) + \mathbb{E}^2\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)\right) \\ + \frac{\phi_{\mathsf{in}} - \phi_{\mathsf{out}}}{4} \mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right) + \ell(k_{\mathsf{in}} - k_{\mathsf{out}}) N \sigma_{\mathsf{ksk}}^2 \frac{\mathfrak{B}^2 + 2}{12} \ .\end{aligned}$$

Proof 39 (Theorem 39) *The proof of this theorem can be found in Appendix A.9.*

□

8.3.2 Some Higher Level Applications

Through Sections 8.1.1, 8.2 and 8.3.1, we discussed the many advantages of using partial and/or secret keys with shared randomness in FHE operations. We now discuss the advantages at a somewhat high level.

Algorithm 45: $\mathsf{CT}_{\mathsf{out}} \leftarrow \mathsf{GlweKeySwitch}'(\mathsf{CT}_{\mathsf{in}}, \mathsf{KSK})$

Context: $\begin{cases} \vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]} \in \mathfrak{R}_{q,N}^{k_{\mathsf{in}}} : \text{the input partial secret key (Definition 35)} \\ \vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]} = (S_{\mathsf{in},0}, \cdots, S_{\mathsf{in},k_{\mathsf{in}}-1}) \\ \vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]} \in \mathfrak{R}_{q,N}^{k_{\mathsf{out}}} : \text{the output partial secret key (Definition 35)} \\ \vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]} = (S_{\mathsf{out},0}, \cdots, S_{\mathsf{out},k_{\mathsf{out}}-1}) \\ (k_{\mathsf{in}}-1)N < \phi_{\mathsf{in}} \leq k_{\mathsf{in}}N \text{ and } (k_{\mathsf{out}}-1)N < \phi_{\mathsf{out}} \leq k_{\mathsf{out}}N \\ \vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]} \prec \vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]} : \text{ secret keys with shared randomness (Definition 37)} \\ \vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]} \neq \vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]} \text{ and } k_{\mathsf{out}} \leq k_{\mathsf{in}} \\ k \in \{k_{\mathsf{out}}-1, k_{\mathsf{out}}\} \text{ such that } \forall 0 \leq i < k,\ S_{\mathsf{in},i} = S_{\mathsf{out},i} \\ P \in \mathfrak{R}_{q,N} \\ \mathsf{CT}_{\mathsf{in}} = (A_0, \cdots, A_{k_{\mathsf{in}}-1}, B) \in \mathfrak{R}_{q,N}^{k_{\mathsf{in}}+1} \\ \mathsf{CT}_{i,j} \in \mathsf{GLWE}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}\left(\frac{q}{\beta^j} \cdot S_{\mathsf{in},i}\right), \text{ for } k_{\mathsf{out}} \leq i < k_{\mathsf{in}} \text{ and } 0 \leq j \leq \ell - 1 \\ \text{if } k = k_{\mathsf{out}} - 1 : \\ \qquad \mathsf{CT}_{k,j} \in \mathsf{GLWE}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}\left(\frac{q}{\beta^j} \cdot (S_{\mathsf{in},k} - S_{\mathsf{out},k})\right), \text{ for } 0 \leq j \leq \ell - 1 \\ \ell \in \mathbb{N} : \text{ the number of levels of the decomposition} \\ \beta \in \mathbb{N} : \text{ the base of the decomposition} \end{cases}$

Input: $\begin{cases} \mathsf{CT}_{\mathsf{in}} \in \mathsf{GLWE}_{\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}}(P) \\ \mathsf{KSK} = \left\{\overline{\mathsf{CT}}_i = (\mathsf{CT}_{i,0}, \cdots, \mathsf{CT}_{i,\ell-1})\right\}_{k \leq i < k_{\mathsf{in}}} \end{cases}$

Output: $\mathsf{CT}_{\mathsf{out}} \in \mathsf{GLWE}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}(P)$

```
  /* Keep the $B$ part and the first part of the mask                          */
1 Set $\mathsf{CT}_{\mathsf{out}} := (A_0, \cdots, A_{k_{\mathsf{out}}-1}, B) \in \mathfrak{R}_{q,N}^{k_{\mathsf{out}}+1}$
  /* Different public material for this potential partial-shared secret key
     polynomial                                                                 */
2 if $k = k_{\mathsf{out}} - 1$ then
3     Update $\mathsf{CT}_{\mathsf{out}} = \mathsf{CT}_{\mathsf{out}} - \left\langle \overline{\mathsf{CT}}_k, \mathsf{dec}^{(\beta,\ell)}(A_k) \right\rangle$
  /* Same process as in Algorithm 2                                             */
4 for $i \in [\![k_{\mathsf{out}}; k_{\mathsf{in}} - 1]\!]$ do
      /* Decompose the mask                                                     */
5     Update $\mathsf{CT}_{\mathsf{out}} = \mathsf{CT}_{\mathsf{out}} - \left\langle \overline{\mathsf{CT}}_i, \mathsf{dec}^{(\beta,\ell)}(A_i) \right\rangle$
6 return $\mathsf{CT}_{\mathsf{out}}$
```

Key Switching Key Compression. When one deploys an FHE instance using secret keys with the shared randomness property, the total amount of public material for keyswitching keys is reduced. Indeed, they only need to generate all the shrinking keyswitching keys (Algorithm 42), from the largest key to the smallest. All of these shrinking key switching keys are way smaller than the sum of all the traditional key switching keys that are usually needed. Note that it is possible to provide more levels in some of the keyswitching keys, and only use the ones that are needed at a moment for a given noise constraint.

Compressed Bootstrapping Keys. In the same manner, secret keys with shared randomness allow to reduce the total amount of bootstrapping keys. When the polynomial size N is shared, since a bootstrapping key is a list of GGSW ciphertexts, each one encrypting a secret key coefficient of the input LWE secret key, one can only provide the GGSW ciphertexts for the largest LWE secret key of the instance. Whenever bootstrapping an LWE ciphertext with a smaller dimension is required, one will only use the first part of the bootstrapping key. In the same spirit, additional levels can be added, and only used when strictly needed.

Easier Parameter Set Conversion. Section 7.2.4 considers use-cases where there are a couple of coexisting parameter sets, and it is necessary to move from one parameter set to the other. Using shared (and partial) secret keys helps converting in a more efficient way ciphertexts between the two (or more) parameter sets and adds less noise during the process. Without the shared randomness property, this requires lots of keyswitching keys and additional, unnecessary computation.

Multikey Compatibility. Both the partial and shared randomness properties are preserved in the MK-FHE (such as [Kim+22; KMS22]) and threshold-FHE approaches. Indeed, summing two partial secret keys results in another partial secret key, and summing together two pairs of secret keys with shared randomness results in a new pair of secret keys with shared randomness. Those new secret keys could improve the performance of MK-FHE and threshold-FHE, which is in general less attractive that the one of (single key) FHE, as well as reduce the size of the total public key material.

Other FHE Schemes. Partial keys and secret keys with shared randomness could be used in other FHE schemes such as FHEW [DM15] or NTRU-based schemes (such as [Bon+22]). These types of keys could also be used in either BFV [Bra12; FV12] or CKKS [Che+17] when larger polynomials are required for the same modulus q, for instance.

Combination With Fixed Hamming Weight. Both partial keys and secret keys with shared randomness could be instantiated with a fixed Hamming weight if needed. We do not explore this topic any further here.

LWE Encryption Public Key With GLWE Material. If one wants to take advantage of the FFT to encrypt fresh LWE ciphertexts with a secret key $\vec{s} \in \mathbb{Z}_q^n$, and/or shrinks the size of ciphertexts with partial GLWE secret key, it is possible to provide a GLWE encryption public key for a partial GLWE secret key $\vec{S}^{[\phi=n]} \in \mathfrak{R}_{q,N}^k$ such that its flattened version is actually $\vec{s}$. In this case, one uses GLWE encryption and applies sample extraction right after that to obtain the desired LWE ciphertext.

8.3.3 Parameters & Benchmarks

In this section, we describe how to generate FHE parameters for all our experiments. We use the procedure introduced in Chapter 4 (see Section 4.3) to compare the different approaches. To demonstrate the impact of partial keys and/or secret keys with shared randomness, we use the Atomic Pattern (AP) called CJP (Definition 30).

We explain below, how to optimize parameters for the different experiments and show the different improvements (both in computational time and size of public material) brought forward by each of the new procedures introduced in this paper.

Real life applications use additions and multiplications by public integers (i.e. a dot product) between two consecutive bootstrappings. Formally, given a list of ciphertexts $\{\mathsf{ct}_i\}_{i\in[\![1,\alpha]\!]} \in (\mathsf{LWE}_{\vec{s}_{\mathsf{in}}})^\alpha$ (with independent noise values) and a list of integers $\{\omega_i\}_{i\in[\![1,\alpha]\!]} \in \mathbb{Z}^\alpha$, one computes $\sum_{i=1}^{\alpha} \omega_i \cdot \mathsf{ct}_i$. In that case, we have $\nu^2 = \sum_{i=1}^{\alpha} \omega_i^2$ and ν is used to fully describe the noise growth during a dot product following the formalization of Chapter 4. In our experiments, we set $\nu = 2^p$ with p the precision of the message.

For every experiment below, the probability of failure is set to $p_{\mathsf{fail}} \le 2^{-13.9}$ (see Theorem 16).

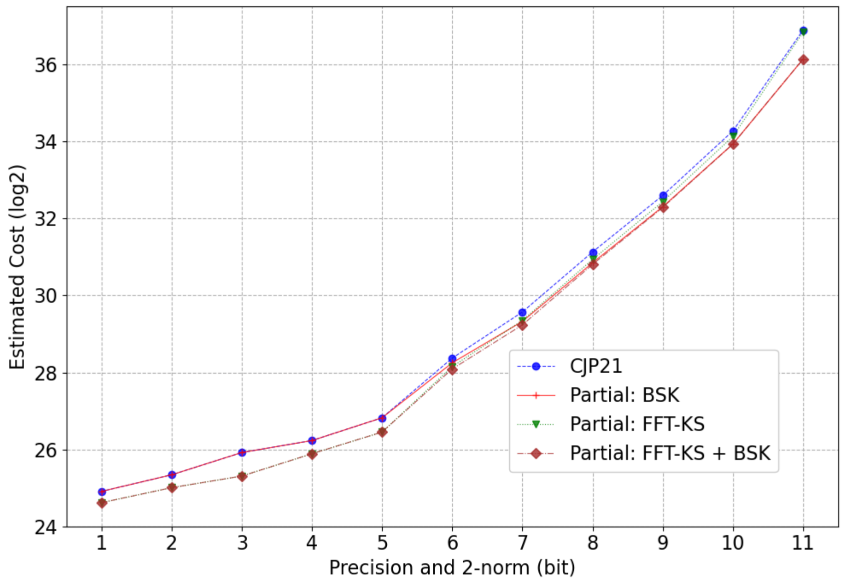


Figure 8.2: Comparison in terms of estimated computation time using the optimization framework introduced in Chapter 4 (see Section 4.3), of the traditional CJP atomic pattern (Definition 30), our baseline, with three variants of CJP based on partial secret keys. Details can be found in Section 8.3.4 and exact plotted values can be found in Tables A.4, A.5 and A.6 in Appendix A.10.

8.3.4 Partial GLWE Secret Key

We conduct three experiments with partial GLWE secret keys (Definition 35), and we plot the results predicted with the optimizer in Figure 8.2. The X axis represents the precision p and the Y axis reflects the base 2 log of the cost estimated by the optimizer. Our baseline is CJP, the blue dashed curve with the $\bullet$ symbol.

The first experiment focuses on the CJP atomic pattern where we allow the GLWE secret key to be partial, with a filling parameter ϕ. On the figure, it is the red solid curve with the $+$ symbol. During optimization, we set ϕ to the minimum between $k \cdot N$ and the value n_{plateau} discussed in Limitation 10. As expected, we observe an improvement mostly with larger precisions, starting at $p = 6$ where the plateau is actually reached.

The second experiment considers the CJP atomic pattern where the traditional LWE-to-LWE key switch is replaced with the FFT-base LWE key switch that we introduced in Algorithm 39. On the figure, it is the green dotted curve with the $\blacktriangledown$ symbol. During the optimization, we had to introduce new FHE parameters for this particular key switch: an input GLWE dimension k_{in}, an output GLWE dimension k_{out} and a polynomial size N_{KS}. We observe a significant improvement for all precisions when using this key switch, but it is more visible with smaller precisions, in the range $[1;6]$.

The third and last experiment is the combination of the two first ones: we allow the GLWE secret key to be partial (when the plateau is reached) and use the FFT-based LWE key switch (Algorithm 39). On the figure, it is the brown dashed curve with the ◆ symbol. As expected, the curve stands below the two others because it is indeed exploiting the best of both improvements. We can see a significant improvement for all precisions.

Note that there is no way to build an LWE-to-LWE key switch based on the FFT without partial secret keys, so no comparison with our results can be done.

8.3.5 Secret Keys with Shared Randomness

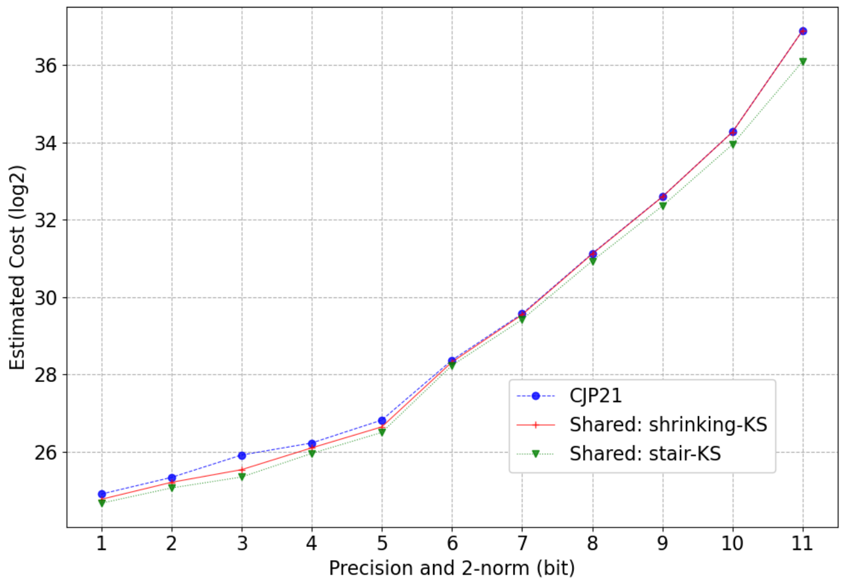


Figure 8.3: Comparison in terms of estimated computation time using the optimization framwork introduced in Chapter 4 (see Section 4.3), of traditional CJP, our baseline, with two variants of CJP based on secret keys with shared randomness. Details can be found in Section 8.3.5 and exact plotted values can be found in Tables A.4, A.5 and A.6 in Appendix A.10.

We conduct two experiments with secret keys with shared randomness (Definition 37), and plot the results predicted by our optimizer in Figure 8.3. This figure follows the same logic as the previous one. Our baseline is still CJP, the blue dashed curve with the • symbol.

The first experiment is the CJP atomic pattern where we allow the secret keys to share their randomness using the shrinking LWE key switch described in Algorithm 42. On the figure, it is the red solid curve with the + symbol. We observe a significant improvement with small precisions up to $p = 6$.

The second and last experiment is the CJP atomic pattern where we allow the secret keys to share their randomness, using the 2-steps stair LWE key switch from Algorithm 43. On the figure, it is the green dotted curve with the ▼ symbol. We see a significant improvement at all precisions.

Note that if one tries to trivially have a 2-steps stair key switch without any shared randomness, the computational cost is basically the same as in CJP.

8.3.6 Combining Both

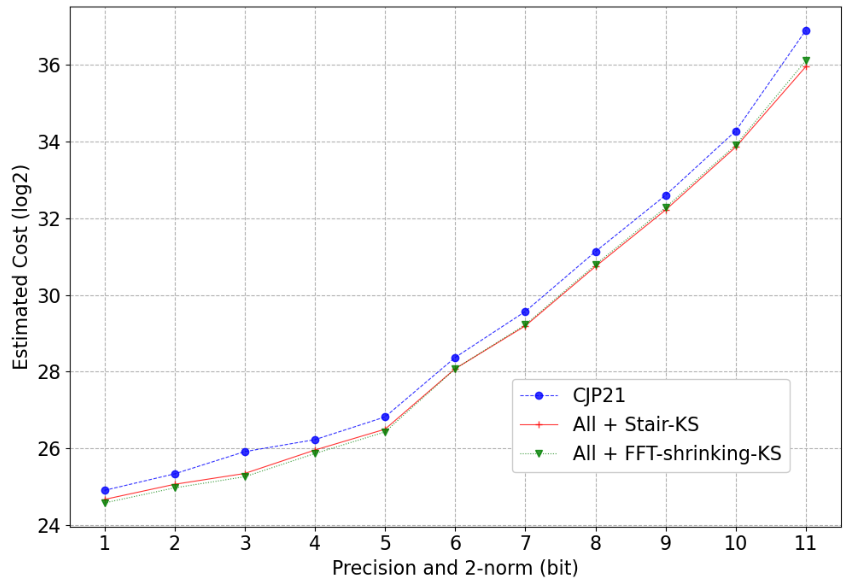


Figure 8.4: Comparison in terms of estimated computation, between traditional CJP, our baseline, and two variants of CJP based on both partial secret keys and secret keys with shared randomness. Details can be found in Section 8.3.6 and exact plotted values can be found in Tables A.4, A.5 and A.6 in Appendix A.10.

We conduct two experiments with both partial keys (Definition 35) and secret keys with shared randomness (Definition 37). We plot first the predicted computational cost in Figure 8.4 obtained by our optimizer. This figure follows the same logic as the previous ones. Our baseline is again CJP, the blue dashed curve with the ● symbol.

The first experiment is the CJP atomic pattern where we allow the secret keys to be partial and to share their randomness. We use the 2-steps stair LWE key switch from Algorithm 43 and we allow the GLWE secret key to be partial (when the plateau is reached). On the figure, it is the red solid curve with the + symbol. We see a significant improvement at all precisions.

The second and last experiment also focuses on the CJP atomic pattern where we allow secret keys to be partial and to share their randomness. We allow the GLWE secret key to be partial (when the plateau is reached), and use the FFT-based LWE key switch (Algorithm 44) since our secret keys also share randomness. On the figure, it is the green dotted curve with the ▼ symbol. We see a similar improvement at all precisions.

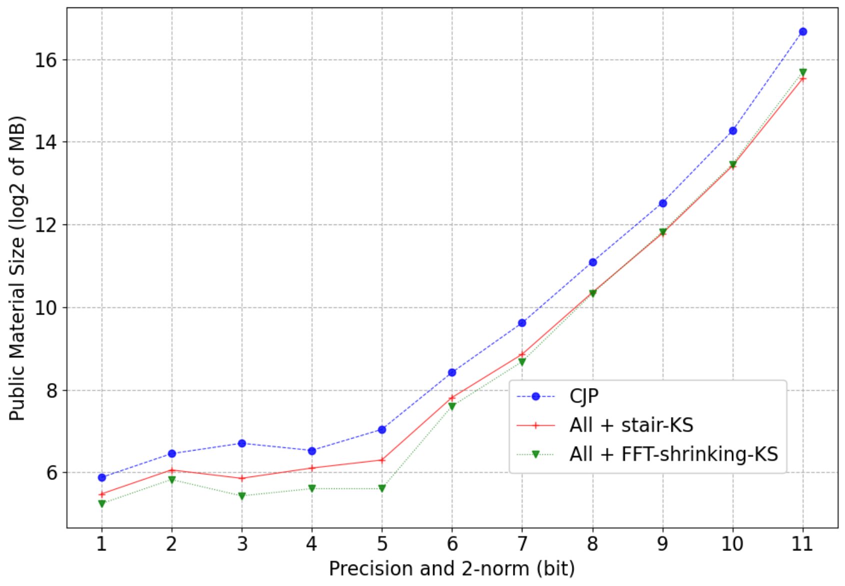


Figure 8.5: Comparison in terms of size of total public key material, between traditional CJP, our baseline, and two variants of CJP based on both partial secret keys and secret keys with shared randomness. Details can be found in Section 8.3.6 and exact plotted values can be found in Tables A.4, A.5 and A.6 in Appendix A.10.

For those experiments, we also plot the size of the total public key material needed in Figure 8.5 to demonstrate their benefit in this matter. The legend corresponding to the experiments is the same as the one of Figure 8.4. Both the stair key switch curve and the FFT shrinking key switch curve stand below our baseline. They actually follow pretty much the curves of the predictions plotted in Figure 8.4.

We finally plot the timings obtained with benchmarks in Figure 8.6 and Figure 8.7

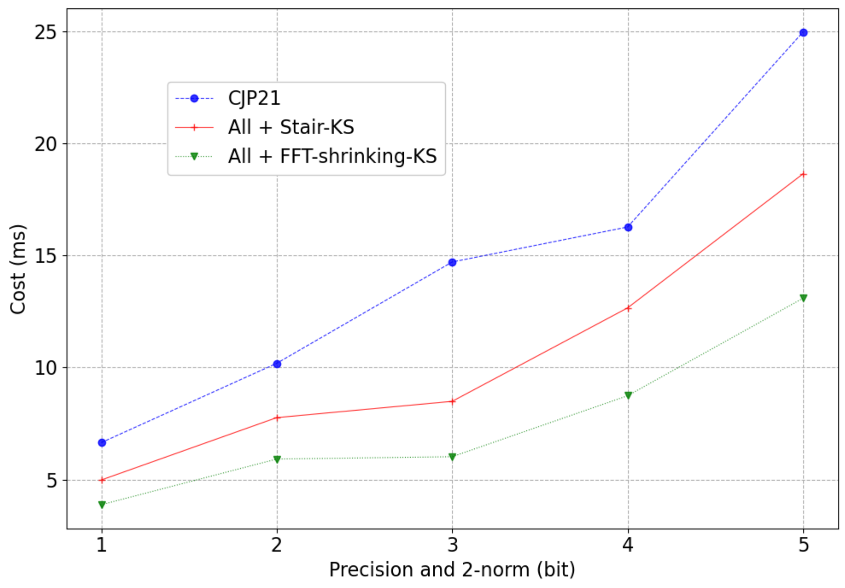


Figure 8.6: Comparison in terms of computing time, between traditional CJP, our baseline, and two variants of CJP based on both partial secret keys and secret keys with shared randomness. Details can be found in Section 8.3.6 and exact plotted values can be found in Tables A.4, A.5 and A.6 in Appendix A.10.

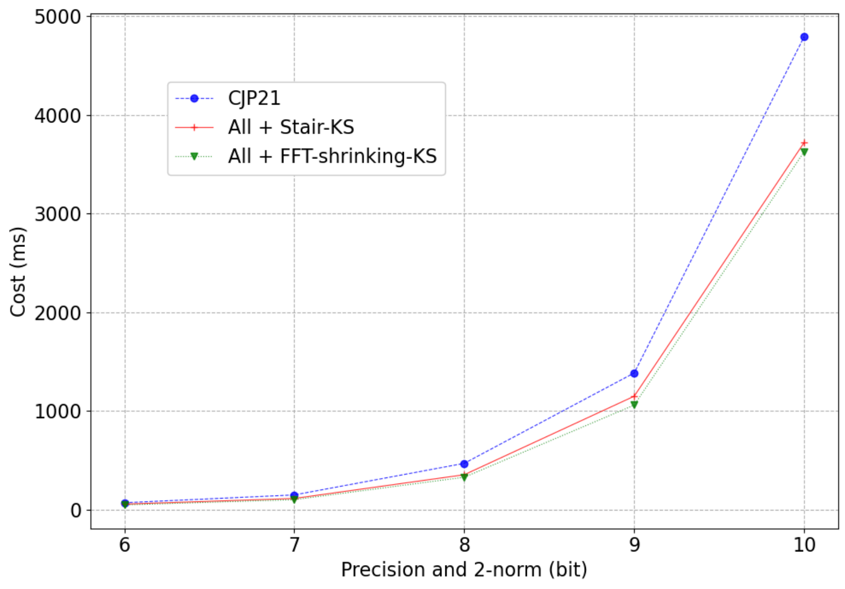


Figure 8.7: Comparison in terms of computing time, between traditional CJP, our baseline, and two variants of CJP based on both partial secret keys and secret keys with shared randomness. Details can be found in Section 8.3.6 and exact plotted values can be found in Tables A.4, A.5 and A.6 in Appendix A.10.

to validate our predictions. The legend corresponding to the experiments is the same as described above for Figure 8.4, except for the Y axis, which is not logarithmic anymore, so one can easily read the timings. Both the stair key switch curve and the FFT shrinking key switch curve are below our baseline as predicted, and we have even better results with the FFT shrinking key switch than expected. Note that at precision $p = 3$ we have a 2.4 speed-up factor compared to the baseline (Figure 8.6).

All of our experiments have been carried out on AWS with a m6i.metal instance Intel Xeon 8375C (Ice Lake) at 3.5 GHz, with 128 vCPUs and 512.0 GiB of memory using the TFHE-rs library [Zam22b].

In Tables A.4, A.5 and A.6 in Appendix A.10, we provide all the parameter sets used to estimate the cost curves of Figure 8.4, their benchmarks and their total public key size.

Conclusion. In this chapter, we introduced new types of secret keys and new algorithms. We proved through comparisons with our FHE optimizer and with extensive benchmarks that these new techniques systematically improve the state of the art in all contexts. When combining both types of secret keys, we observed significant improvements for both small and higher precisions. In particular, we show computational speed-up factors ranging between 1.3 and 2.4 while keeping the same level of security and failure probability. Furthermore, the size of the public material (i.e., key switching and bootstrapping keys) is also reduced by factors ranging from 1.5 to 2.7.

Chapter 9

Conclusion

In the introduction, we asked ourselves the following question:

How can we make FHE more practical ?

This question has been the common thread of this thesis and we gave answers all along this manuscript. We also identified limitations of TFHE and explained how our work alleviates them partially or totally. Let us summarize our different contributions.

In Chapter 3, we introduced the first components of our optimization framework: the security oracle and the concept of noise model. A security oracle estimates a minimal noise variance to be used at encryption time that satisfies some security level given an LWE dimension, a ciphertext modulus, a noise distribution and a secret key distribution. A noise model is a collection of noise formulae, functions that predicts the noise distribution after an FHE operator. This model is used to track the noise all along a computation and is crucial to ensure the correctness. We also explained how to correct the noise formula of the PBS by taking into account the impact of the FFT on the noise.

In Chapter 4, we introduced an optimization framework that removes Limitations 14 and 15. Those limitations state that there is no tool to automatically select the best parameters for an FHE computation and that it is hard to compare different FHE algorithms associated with the same plaintext operations as they satisfy different noise and cost trade-offs. This solves one of the main blocking factors in the widespread adoption of FHE.

In the end, we succeeded in translating the FHE optimization problem into a more classical optimization problem (simplified along the way), where already known and powerful optimization techniques, such as the branch-and-bound algorithm, finally take over.

Roughly speaking, our optimization framework takes as input a graph of mathematical operators, such as additions, multiplications or LUT evaluations, and a list of *translation* *rules*, i.e., various ways to transform these cleartext operations into FHE operations. A ciphertext in this graph is associated with some metadata regarding the precision of the message and whether or not it should be encrypted. The output provided by the optimization framework is an *optimal graph* of FHE operators along with the *optimal parameter set* for this graph.

By finding the fastest set of parameters for different contexts, one can truly compare FHE operators computing the same plaintext function, thus removing Limitation 15. An accurate comparison of two FHE operators needs to take into account the failure probability of both algorithms and their respective impacts on noise growth. Some of the comparisons we made in this thesis shed some light on the relationship between the precision of encrypted integers, and the time needed to compute over them. In particular, it is clear that if we want to work with large precisions in TFHE, it is more efficient to use several ciphertexts with a radix or a CRT encoding to encrypt a single message, instead of considering huge parameters to make it fit into a single ciphertext.

We also demonstrated that adding more bootstrappings can speed up some homomorphic computations and we proved that the position of a key switch operator has a non-negligible impact on the efficiency of a circuit. Finally, we described ways to use our framework to take into account other constraints such as having a consensus-friendly FHE evaluation, or allowing an optimization for more than a single pair of bootstrapping and keyswitching keys.

In Chapter 5, we introduced several new FHE algorithms. For every algorithm, we carried out a noise analysis to get an analytical formula that models its noise growth and we provided a cost function that can be used to select efficient parameters.

First, we *generalized* the PBS of TFHE so it can evaluate *several functions at once* without additional computation or noise. This approach is possible when the message to bootstrap is small enough. It overcomes Limitation 6 and enables to compute a single generalized PBS when computing a circuit bootstrapping instead of ℓ PBSs, overcoming Limitation 8. Circuit bootstrapping is particularly interesting in the leveled evaluation of a lookup table, as shown in [Chi+17].

Then, we introduced a way to compute a lookup table on a rounded input using a rounded-PBS which contributes to remove the limitation of small precisions in the PBS (Limitation 2).

Next, we thoroughly studied the noise growth of a tensor product followed by a relinearization (i.e., the BFV-like multiplication) and found *parameters compatible with TFHE*. Our noise analysis is also valid for BFV-like schemes and can help estimate the noise growth in those schemes. Using the GLWE multiplication, we were able to build a native LWE multiplication. This multiplication is efficient and does not require a PBS, which overcomes Limitation 4. We also proposed a packed variant of this algorithm to compute several LWE products at once or a sum of several LWE products at once.

In Chapter 6, we presented three new algorithms that remove the constraint of the bit of padding in TFHE's PBS (Limitation 1). From the new LWE multiplication, we defined two *new PBSs* that do not require the MSB to be set to zero, overcoming Limitation 1. These new procedures are composed of a few generalized PBSs that can be computed in parallel, which makes them more compatible with multi-threading (Limitation 3).

From these two new Without-Padding PBS (WoP-PBS), we were able to *homomorphically decompose* a plaintext from a single ciphertext into several ciphertexts encrypting blocks of the input plaintext, thus overcoming Limitation 5.

From this new decomposition algorithm and the Tree-PBS [GBA21], we were able to create a *fast PBS for larger input messages*, overcoming Limitation 2. We can also in an even faster manner, refresh the noise in a ciphertext from this new decomposition algorithm.

Next, we introduced another WoP-PBS. It computes a bootstrap on one or several ciphertexts, which do not need to have a known bit of padding (Limitation 1). Using this new algorithm, it is possible to evaluate in an efficient manner a generic lookup table on a radix, CRT, or hybrid integer representation, which was not possible before (Limitation 13). This new PBS scales particularly well with high precisions (Limitation 2), can be multithreaded (Limitation 3) and can be adapted to compute several lookup tables (with a slight cost overhead but without additional noise), which also contributes to remove Limitation 6.

In Chapter 7, we studied how to build integer arithmetics with FHE. First, we defined a new way to build a Boolean circuit with TFHE that does not need a PBS at each Boolean gate. To do so, we leveraged the LWE multiplication introduced in Chapter 5. We extended this method to work with small integers and explained how to compute the basics operations (addition, multiplication and LUT evaluation).

Then, we explained how to implement an integer arithmetic for high-precision integers with the help of the last WoP-PBS introduced in Chapter 6. We detailed how to use in practice the radix and the CRT encodings and we introduced a hybrid approach that takes the best of the CRT and radix approaches. We also provided extensive *benchmarks* to show the practicality of our approach.

In Chapter 8, we introduced two new types of secret keys. The partial keys are especially useful when the product of the GLWE dimension by the polynomial size is large. When it is large enough, the minimal encryption noise reaches its minimal value and we end up in a scenario where we have huge parameters, which impacts both the cost and the noise. Thanks to our new secret keys, having a large product between the polynomial size and the GLWE dimension still impacts the cost but impacts the noise less than when using classical secret keys.

Thanks to the second type of secret keys, the cost of key switches is greatly decreased. By sharing parts of the coefficients between secret keys, we can decrease the noise added during the key switches and also improve the cost. Since in real use-cases, lots of key switches are involved, this new type of secret keys clearly improves the resulting execution time.

Every new algorithm and its variants are backed up by extensive benchmarks and analysis using our optimization framework.

Learnings & Open Questions In the course of this work, we learned that efficient cryptographic algorithms are *not enough*. Many algorithms have degrees of freedom and these have a huge impact on the cost and on the noise. We need tools that select the best parameters to minimize the cost of a computation while guaranteeing its correctness.

To pick fast parameters, we need to use realistic cost models. In this thesis, we use algorithmic complexities as a surrogate for the execution time. In the future, we shall test other cost models to determine which one gives the best parameters in practice. Building accurate cost models is a very difficult task but it must be undertaken as it yields significant improvements.

To guarantee the correctness of an FHE computation, we need to have tight noise formulae as they are used to track the noise all along the computation. The correctness of a computation is tightly related to the noise distributions involved in it but we do not have access to the actual distribution; we use noise formulae that model that the noise distribution. These approximations must be as realistic as possible to guarantee that we are not underestimating the failure probability of the computation nor that we are overestimating it too much. The former leads to incorrect results and the latter to sub-optimal computations. During the noise analysis presented in this thesis, we made some hypothesis and some of them can be removed, resulting in tighter formulae. More generally, we need the tightest formula for each and every FHE algorithm, not only for TFHE.

An optimizer *must be used* to guide the research in FHE. There are too many trade-offs to consider, we cannot compare algorithms just by looking at the sequences of operations nor by benchmarking a finite set of parameters. As different algorithms satisfy different trade-offs, we cannot analytically compare them. To truly select the best one, we need to put them all in the same context i.e., in graphs that perform the same plaintext operations up to the same failure probability, optimize every possible graphs and find out which algorithm is the best. Most of the time, no algorithm is better than every other in every use cases. As they satisfy different cost-noise trade-offs, they will be efficient in different scenarios. A thorough analysis of every algorithm has begun in this thesis but it is still missing comparisons with some late improvements of TFHE's PBS. This work would be of overriding interest for the community and to guide future research. In the future, every contribution must come with its associated noise formula to be easily compared with other methods.

The new types of secret keys introduced in Chapter 8 bring significant improvements to TFHE. As with many new assumptions, a more complete security analysis is needed. It will be also interesting to look more closely at the stair key switch (Algorithm 43) which improves the state of the art by a significant factor. We studied a special case, the 2-step key switch. The interest of this algorithm for a larger number of steps and the method to chose those steps is still an open problem.

Some FHE schemes offer approximate arithmetics instead of exact computations up to a given failure probability. An approximate arithmetic with the power of TFHE's PBS could be very interesting for machine learning use-cases.

In this thesis, we showed that FHE algorithms by themselves were not sufficient to make FHE practical. We need to leverage optimization techniques to find the best parameter set for a given use case. Without these techniques, we can not truly compare different algorithms performing the same plaintext operation. We need to introduce new security hypothesis that we can leverage to improve the efficiency of existing algorithms. We also need to find new algorithms to efficiently support very large messages e.g., more than 32-bits. If the FHE community is up to the task, the future will see protocols involving FHE in combination with other privacy-preserving technologies such as *Multi-Party Computation* (MPC) and *Zero Knowledge Proofs*.

BIBLIOGRAPHY

Lattice Security

[Alb17a] Martin Albrecht, "On Dual Lattice Attacks Against Small-Secret LWE and Parameter Choices in HElib and SEAL", *in*: Apr. 2017, pp. 103–129, ISBN: 978-3-319-56613-9, DOI: 10.1007/978-3-319-56614-6_4.

[Alb+17a] Martin Albrecht, Florian Göpfert, Fernando Virdia, and Thomas Wunderer, "Revisiting the Expected Cost of Solving uSVP and Applications to LWE", *in*: Nov. 2017, pp. 297–322, ISBN: 978-3-319-70693-1, DOI: 10.1007/978-3-319-70694-8_11.

[Alb+14] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, and Ludovic Perret, *Algebraic Algorithms for LWE*, Cryptology ePrint Archive, Paper 2014/1018, https://eprint.iacr.org/2014/1018, 2014, URL: https://eprint.iacr.org/2014/1018.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott, "On the concrete hardness of learning with errors", *in*: *Journal of Mathematical Cryptology* 9.*3* (2015), https://bitbucket.org/malb/lwe-estimator/src/master/, pp. 169–203, DOI: 10.1515/jmc-2015-0016.

[Alk+16b] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe, "Post-quantum Key Exchange—A New Hope", *in*: *25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX: USENIX Association, Aug. 2016, pp. 327–343, ISBN: 978-1-931971-32-4, URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/alkim.

[AG11] Sanjeev Arora and Rong Ge, "New Algorithms for Learning in Presence of Errors", *in*: *Automata, Languages and Programming*, ed. by Luca Aceto, Monika Henzinger, and Jiří Sgall, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 403–415, ISBN: 978-3-642-22006-7.

[BG14] Shi Bai and Steven Galbraith, "Lattice Decoding Attacks on Binary LWE", *in*: July 2014, ISBN: 978-3-319-08343-8, DOI: `10.1007/978-3-319-08344-5_21`.

[Che+19a] Jung Cheon, Minki Hhan, Seungwan Hong, and Yongha Son, "A Hybrid of Dual and Meet-in-the-Middle Attack on Sparse and Ternary Secret LWE", *in*: *IEEE Access* PP (June 2019), pp. 1–1, DOI: `10.1109/ACCESS.2019.2925425`.

[Che+19b] Jung Hee Cheon, Minki Hhan, Seungwan Hong, and Yongha Son, *A Hybrid of Dual and Meet-in-the-Middle Attack on Sparse and Ternary Secret LWE*, Cryptology ePrint Archive, Paper 2019/1114, `https://eprint.iacr.org/2019/1114`, 2019, URL: `https://eprint.iacr.org/2019/1114`.

[Dac+20a] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi, "LWE with Side Information: Attacks and Concrete Security Estimation", *in*: Aug. 2020, pp. 329–358, ISBN: 978-3-030-56879-5, DOI: `10.1007/978-3-030-56880-1_12`.

[Duc13] Léo Ducas-Binda, "Signatures fondées sur les réseaux euclidiens: attaques, analyses et optimisations", PhD thesis, PhD thesis, École Normale Supérieure Paris, 2013. http://cseweb. ucsd. edu …, 2013.

[EJK20] Thomas Espitau, Antoine Joux, and Natalia Kharchenko, "On a Dual/Hybrid Approach to Small Secret LWE - A Dual/Enumeration Technique for Learning with Errors and Application to Security Estimates of FHE Schemes", *in*: *International Conference on Cryptology in India*, 2020, URL: `https://api.semanticscholar.org/CorpusID:219332962`.

[GJS15b] Qian Guo, Thomas Johansson, and Paul Stankovski, "Coded-BKW: Solving LWE Using Lattice Codes.", English, *in*: *Advances in Cryptology – CRYPTO 2015/Lecture notes in computer science*, ed. by Rosario Gennaro and Matthew Robshaw, vol. 9215, 35th Annual Cryptology Conference ; Conference date: 16-08-2015 Through 20-08-2015, Germany: Springer, 2015, pp. 23–42, ISBN: 978-3-662-47988-9, DOI: `10.1007/978-3-662-47989-6_2`.

[How01] Nick Howgrave-Graham, "Approximate Integer Common Divisors", *in*: *Cryptography and Lattices*, ed. by Joseph H. Silverman, Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 51–66, ISBN: 978-3-540-44670-5.

[KF15a] Paul Kirchner and Pierre-Alain Fouque, "An Improved BKW Algorithm for LWE with Applications to Cryptography and Lattices", *in*: *IACR Cryptol. ePrint Arch.* 2015 (2015), p. 552, URL: `https://api.semanticscholar.org/CorpusID:1957267`.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev, "On Ideal Lattices and Learning with Errors over Rings", *in*: *Advances in Cryptology - EUROCRYPT 2010*, ed. by Henri Gilbert, vol. 6110, Lecture Notes in Computer Science, Springer, 2010.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev, "On ideal lattices and learning with errors over rings", *in*: *Journal of the ACM* 60.*6* (2013), Earlier version in EUROCRYPT 2010, 43:1–43:35, DOI: `10.1145/2535925`.

[MR09a] Daniele Micciancio and Oded Regev, "Lattice-based Cryptography", *in*: *Post-Quantum Cryptography*, ed. by Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 147–191, ISBN: 978-3-540-88702-7, DOI: `10.1007/978-3-540-88702-7_5`, URL: `https://doi.org/10.1007/978-3-540-88702-7_5`.

[Pla18] Rachel Player, "Parameter selection in lattice-based cryptography", *in*: 2018, URL: `https://api.semanticscholar.org/CorpusID:216024218`.

[PV20] Eamonn W. Postlethwaite and Fernando Virdia, "On the Success Probability of Solving Unique SVP via BKZ", *in*: *IACR Cryptology ePrint Archive*, 2020, URL: `https://api.semanticscholar.org/CorpusID:226207574`.

[Reg05] Oded Regev, "On lattices, learning with errors, random linear codes, and cryptography", *in*: *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, 2005*, ed. by Harold N. Gabow and Ronald Fagin, ACM, 2005.

[Reg09] Oded Regev, "On lattices, learning with errors, random linear codes, and cryptography", *in*: *Journal of the ACM* 56.*6* (2009), Earlier version in STOC 2005, 34:1–34:40, DOI: `10.1145/1568318.1568324`.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", *in*: *Commun. ACM* 21.*2* (Feb. 1978), pp. 120–126, ISSN: 0001-0782, DOI: `10.1145/359340.359342`, URL: `https://doi.org/10.1145/359340.359342`.

[Sch03] Claus Schnorr, “Lattice Reduction by Random Sampling and Birthday Methods”, *in*: vol. 2607, Apr. 2003, ISBN: 978-3-540-00623-7, DOI: `10.1007/3-540-36494-3_14`.

[Ste+09] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa, “Efficient public key encryption based on ideal lattices”, *in*: *Advances in Cryptology – ASIACRYPT 2009*, vol. 5912, Lecture Notes in Computer Science, Springer, 2009, pp. 617–635, DOI: `10.1007/978-3-642-10366-7_36`.

Other FHE Schemes

[AN16] Seiko Arita and Shota Nakasato, “Fully homomorphic encryption for point numbers”, *in*: *International Conference on Information Security and Cryptology*, Springer, 2016.

[Baj+16] Jean-Claude Bajard, Julien Eynard, M Anwar Hasan, and Vincent Zucca, “A full RNS variant of FV like somewhat homomorphic encryption schemes”, *in*: *International Conference on Selected Areas in Cryptography*, Springer, 2016.

[Bon+22] Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder V. L. Pereira, and Nigel P. Smart, “FINAL: Faster FHE Instantiated with NTRU and LWE”, *in*: *Advances in Cryptology – ASIACRYPT 2022*, ed. by Shweta Agrawal and Dongdai Lin, Cham: Springer Nature Switzerland, 2022, pp. 188–215, ISBN: 978-3-031-22966-4.

[BST20] Florian Bourse, Olivier Sanders, and Jacques Traoré, “Improved secure integer comparison via homomorphic encryption”, *in*: *CT-RSA*, Springer, 2020.

[Bra12] Zvika Brakerski, “Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP”, *in*: *IACR Cryptology ePrint Archive* 2012 (2012), URL: `http://eprint.iacr.org/2012/078`.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan, “(Leveled) fully homomorphic encryption without bootstrapping”, *in*: *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, 2012.

[Che+20] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song, “Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts”, *in*: *IACR Cryptol. ePrint Arch.* (2020), URL: `https://eprint.iacr.org/2020/015`.

[CH18] Hao Chen and Kyoohyung Han, “Homomorphic lower digits removal and improved FHE bootstrapping”, *in*: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2018.

[Che+18a] Hao Chen, Kim Laine, Rachel Player, and Yuhou Xia, “High-precision arithmetic in homomorphic encryption”, *in*: *Cryptographers’ Track at the RSA Conference*, Springer, 2018.

[Che+18b] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song, "A full RNS variant of approximate homomorphic encryption", *in*: *International Conference on Selected Areas in Cryptography*, Springer, 2018.

[Che+17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers", *in*: *Advances in Cryptology - ASIACRYPT 2017*, 2017.

[Dij+10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan, "Fully Homomorphic Encryption over the Integers", *in*: *Advances in Cryptology – EUROCRYPT 2010*, ed. by Henri Gilbert, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 24–43, ISBN: 978-3-642-13190-5.

[FV12] Junfeng Fan and Frederik Vercauteren, "Somewhat Practical Fully Homomorphic Encryption", *in*: *IACR Cryptology ePrint Archive* 2012 (2012), URL: `http://eprint.iacr.org/2012/144`.

[Gen09] Craig Gentry, "Fully homomorphic encryption using ideal lattices", *in*: *41st Annual ACM Symposium on Theory of Computing*, ACM Press, 2009, pp. 169–178, DOI: `10.1145/1536414.1536440`.

[Gen10] Craig Gentry, "Computing arbitrary functions of encrypted data", *in*: *Communications of the ACM* 53.*3* (2010), Earlier version in STOC 2009, pp. 97–105, DOI: `10.1145/1666420.1666444`.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based", *in*: *Advances in Cryptology – CRYPTO 2013, Part I*, vol. 8042, Lecture Notes in Computer Science, Springer, 2013, pp. 75–92, DOI: `10.1007/978-3-642-40041-4_5`.

[HPS19] Shai Halevi, Yuriy Polyakov, and Victor Shoup, "An improved RNS variant of the BFV homomorphic encryption scheme", *in*: *Cryptographers' Track at the RSA Conference*, Springer, 2019.

[HS15] Shai Halevi and Victor Shoup, "Bootstrapping for helib", *in*: *Annual International conference on the theory and applications of cryptographic techniques*, Springer, 2015.

[Lee+20b] Yongwoo Lee, Joonwoo Lee, Young-Sik Kim, HyungChul Kang, and Jong-Seon No, *High-Precision and Low-Complexity Approximate Homomorphic Encryption by Error Variance Minimization*, Cryptology ePrint Archive, Report 2020/1549, `https://eprint.iacr.org/2020/1549`, 2020.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan, “On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption”, *in*: *Proceedings of the Annual ACM Symposium on Theory of Computing* (May 2012), DOI: `10.1145/2213977.2214086`.

[RAD78] Ronald L. Rivest, Len Adleman, and Michael L. Detouzos, “On data banks and privacy homomorphisms”, *in*: *Foundations of Secure Computation*, Available at `https://people.csail.mit.edu/rivest/pubs.html`, Academic Press, 1978, pp. 165–179.

[SV14] Nigel P. Smart and Frederik Vercauteren, “Fully homomorphic SIMD operations”, *in*: *Des. Codes Cryptography* 71.*1* (2014).

TFHE

[ACS20] Pascal Aubry, Sergiu Carpov, and Renaud Sirdey, “Faster Homomorphic Encryption is not Enough: Improved Heuristic for Multiplicative Depth Minimization of Boolean Circuits”, *in*: *Topics in Cryptology - CT-RSA 2020 - The Cryptographers’ Track at the RSA Conference 2020, San Francisco, CA, USA, February 24-28, 2020, Proceedings*, ed. by Stanislaw Jarecki, vol. 12006, Lecture Notes in Computer Science, Springer, 2020, pp. 345–363, DOI: `10.1007/978-3-030-40186-3\_15`, URL: `https://doi.org/10.1007/978-3-030-40186-3%5C_15`.

[Bou+20] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev, “CHIMERA: Combining Ring-LWE-based Fully Homomorphic Encryption Schemes”, *in*: *J. Math. Cryptol.* 14.*1* (2020).

[CAS17] Sergiu Carpov, Pascal Aubry, and Renaud Sirdey, “A Multi-start Heuristic for Multiplicative Depth Minimization of Boolean Circuits”, *in*: *Combinatorial Algorithms - 28th International Workshop, IWOCA 2017, Newcastle, NSW, Australia, July 17-21, 2017, Revised Selected Papers*, ed. by Ljiljana Brankovic, Joe Ryan, and William F. Smyth, vol. 10765, Lecture Notes in Computer Science, Springer, 2017, pp. 275–286, DOI: `10.1007/978-3-319-78825-8\_23`, URL: `https://doi.org/10.1007/978-3-319-78825-8%5C_23`.

[CIM19] Sergiu Carpov, Malika Izabachène, and Victor Mollimard, “New techniques for multi-value input homomorphic evaluation and applications”, *in*: *Cryptographers’ Track at the RSA Conference*, Springer, 2019.

[CCR19] Hao Chen, Ilaria Chillotti, and Ling Ren, “Onion Ring ORAM: Efficient Constant Bandwidth Oblivious RAM from (Leveled) TFHE”, *in*: *CCS 2019*, ACM, 2019, URL: `https://doi.org/10.1145/3319535.3354226`.

[Che+21] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song, “Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts”, *in*: *Applied Cryptography and Network Security*, ed. by Kazue Sako and Nils Ole Tippenhauer, Cham: Springer International Publishing, 2021, pp. 460–479.

[Chi+16a] 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*, 2016.

[Chi+17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène, “Faster Packed Homomorphic Operations and Efficient Circuit Bootstrapping for TFHE”, *in*: *Advances in Cryptology - ASIACRYPT 2017*, 2017.

[Chi+20a] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène, “TFHE: Fast fully homomorphic encryption over the torus”, *in*: *Journal of Cryptology* 33.*1* (2020), Earlier versions in ASIACRYPT 2016 and 2017, pp. 34–91, DOI: `10.1007/s00145-019-09319-x`.

[CJP21] Ilaria Chillotti, Marc Joye, and Pascal Paillier, “Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks”, *in*: *Cyber Security Cryptography and Machine Learning - 5th International Symposium, CSCML 2021*, vol. 12716, Lecture Notes in Computer Science, Springer, 2021.

[Cle+22] 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, Report 2022/149, `https://ia.cr/2022/149`, 2022.

[DM15] Léo Ducas and Daniele Micciancio, “FHEW: Bootstrapping homomorphic encryption in less than a second”, *in*: *Advances in Cryptology – EUROCRYPT 2015, Part I*, vol. 9056, Lecture Notes in Computer Science, Springer, 2015, pp. 617–640, DOI: `10.1007/978-3-662-46800-5_24`.

[GBA21] Antonio Guimarães, Edson Borin, and Diego F. Aranha, “Revisiting the functional bootstrap in TFHE”, *in*: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.*2* (2021).

[Joy21] Marc Joye, “Balanced Non-adjacent Forms”, *in*: *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part III*, ed. by Mehdi Tibouchi and Huaxiong Wang, vol. 13092, Lecture Notes in Computer Science, Springer, 2021, pp. 553–576, DOI: `10.1007/978-3-030-92078-4\_19`, URL: `https://doi.org/10.1007/978-3-030-92078-4%5C_19`.

[JP22] Marc Joye and Pascal Paillier, “Blind Rotation in Fully Homomorphic Encryption with Extended Keys”, *in*: *Cyber Security, Cryptology, and Machine Learning*, ed. by Shlomi Dolev, Jonathan Katz, and Amnon Meisels, Cham: Springer International Publishing, 2022, pp. 1–18, ISBN: 978-3-031-07689-3.

[Kim+22] Taechan Kim, Hyesun Kwak, Dongwon Lee, Jinyeong Seo, and Yongsoo Song, “Asymptotically Faster Multi-Key Homomorphic Encryption from Homomorphic Gadget Decomposition”, *in*: *IACR Cryptol. ePrint Arch.* (2022), p. 347, URL: `https://eprint.iacr.org/2022/347`.

[KO22] Jakub Klemsa and Melek Onen, *Parallel Operations over TFHE-Encrypted Multi-Digit Integers*, Cryptology ePrint Archive, Report 2022/067, 2022, URL: `https://ia.cr/2022/067`.

[KS21] Kamil Kluczniak and Leonard Schild, “FDFB: Full Domain Functional Bootstrapping Towards Practical Fully Homomorphic Encryption”, *in*: *CoRR* (2021), URL: `https://arxiv.org/abs/2109.02731`.

[KMS22] Hyesun Kwak, Seonhong Min, and Yongsoo Song, “Towards Practical Multikey TFHE: Parallelizable, Key-Compatible, Quasi-linear Complexity”, *in*: *IACR Cryptol. ePrint Arch.* (2022), p. 1460, URL: `https://eprint.iacr.org/2022/1460`.

[Lee+23] Changmin Lee, Seonhong Min, Jinyeong Seo, and Yongsoo Song, “Faster TFHE Bootstrapping with Block Binary Keys”, *in*: *ACM ASIACCS 2023*, 2023.

[LY23] Kang Hoon Lee and Ji Won Yoon, “Discretization Error Reduction for High Precision Torus Fully Homomorphic Encryption”, *in*: *Public-Key Cryptography–PKC 2023: 26th IACR International Conference on Practice and Theory of Public-Key Cryptography, Atlanta, GA, USA, May 7–10, 2023, Proceedings, Part II*, Springer, 2023, pp. 33–62.

[LMP21] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov, *Large-Precision Homomorphic Sign Evaluation using FHEW/TFHE Bootstrapping*, Cryptology ePrint Archive, Report 2021/1337, `https://ia.cr/2021/1337`, 2021.

[Mat+21] Kotaro Matsuoka, Yusuke Hoshizuki, Takashi Sato, and Song Bian, “Towards Better Standard Cell Library: Optimizing Compound Logic Gates for TFHE”, *in*: *WAHC ’21: Proceedings of the 9th on Workshop on Encrypted*

Computing & Applied Homomorphic Cryptography, Virtual Event, Korea, 15 November 2021, WAHC@ACM, 2021, URL: `https://doi.org/10.1145/3474366.3486927`.

[Zho+18] Tanping Zhou, Xiaoyuan Yang, Longfei Liu, Wei Zhang, and Ningbo Li, "Faster Bootstrapping With Multiple Addends", *in*: *IEEE Access* 6 (2018), pp. 49868–49876, DOI: `10.1109/ACCESS.2018.2867655`.

Implementations

[CDS15] Sergiu Carpov, Paul Dubrulle, and Renaud Sirdey, “Armadillo: a compilation chain for privacy preserving applications”, *in*: *Proceedings of the 3rd International Workshop on Security in Cloud Computing*, 2015.

[Chi+16b] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène, “TFHE: Fast fully homomorphic encryption library”, `https://tfhe.github.io/tfhe/`, Aug. 2016.

[DM17] Léo Ducas and Daniele Micciancio, *FHEW: A Fully Homomorphic Encryption library*, 2017.

[FJ05] Matteo Frigo and Steven G. Johnson, “The design and implementation of FFTW3”, *in*: *Proceedings of the IEEE* 93.*2* (2005), Special issue on “Program Generation, Optimization, and Platform Adaptation” `http://www.fftw.org/`, pp. 216–231.

[Har+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant, “Array programming with NumPy”, *in*: *Nature* 585.*7825* (Sept. 2020), pp. 357–362, DOI: `10.1038/s41586-020-2649-2`, URL: `https://doi.org/10.1038/s41586-020-2649-2`.

[20] “ISO/IEC/IEEE International Standard - Floating-point arithmetic”, *in*: *ISO/IEC 60559:2020(E) IEEE Std 754-2019* (2020), pp. 1–86, DOI: `10.1109/IEEESTD.2020.9091348`.

[Pad11] David Padua, “FFTW”, *in*: *Encyclopedia of Parallel Computing*, ed. by David Padua, Boston, MA: Springer US, 2011, pp. 671–671, ISBN: 978-0-387-09766-4, DOI: `10.1007/978-0-387-09766-4_397`, URL: `https://doi.org/10.1007/978-0-387-09766-4_397`.

[VG13] Joachim Von Zur Gathen and Jürgen Gerhard, *Modern computer algebra*, Cambridge university press, 2013.

[Zam22a] Zama, *Concrete: TFHE Compiler that converts python programs into FHE equivalent*, `https://github.com/zama-ai/concrete`, 2022.

[Zam22b] Zama, *TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data*, `https://github.com/zama-ai/tfhe-rs`, 2022.

Optimization for FHE

[Chi+18] Eduardo Chielle, Oleg Mazonka, Homer Gamil, Nektarios Georgios Tsoutsos, and Michail Maniatakos, *E3: A Framework for Compiling C++ Programs with Encrypted Operands*, Cryptology ePrint Archive, Paper 2018/1013, 2018.

[Dat+20] Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine, and Madan Musuvathi, "EVA: an encrypted vector arithmetic language and compiler for efficient homomorphic computation", *in*: *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, ACM, June 2020, DOI: `10.1145/3385412.3386023`, URL: `https://doi.org/10.1145%2F3385412.3386023`.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart, "Homomorphic Evaluation of the AES Circuit", *in*: *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, ed. by Reihaneh Safavi-Naini and Ran Canetti, vol. 7417, Lecture Notes in Computer Science, Springer, 2012, pp. 850–867, DOI: `10.1007/978-3-642-32009-5\_49`, URL: `https://doi.org/10.1007/978-3-642-32009-5%5C_49`.

[Gor+21] Shruthi Gorantala, Rob Springer, Sean Purser-Haskell, William Lam, Royce J. Wilson, Asra Ali, Eric P. Astor, Itai Zukerman, Sam Ruth, Christoph Dibak, Phillipp Schoppmann, Sasha Kulankhina, Alain Forget, David Marn, Cameron Tew, Rafael Misoczki, Bernat Guillén, Xi Ye, Dennis Kraft, Damien Desfontaines, Aishe Krishnamurthy, Miguel Guevara, Irippuge Milinda Perera, Iurii Sushko, and Bryant Gipson, "A General Purpose Transpiler for Fully Homomorphic Encryption", *in*: *IACR Cryptol. ePrint Arch.* 2021 (2021), p. 811, URL: `https://api.semanticscholar.org/CorpusID:235436123`.

[GKT22] Charles Gouert, Rishi Khan, and Nektarios Georgios Tsoutsos, *Optimizing Homomorphic Encryption Parameters for Arbitrary Applications*, Cryptology ePrint Archive, Paper 2022/575, `https://eprint.iacr.org/2022/575`, 2022, URL: `https://eprint.iacr.org/2022/575`.

[KM78] Ravindran Kannan and Clyde L. Monma, “On the Computational Complexity of Integer Programming Problems”, *in*: *Optimization and Operations Research*, ed. by Rudolf Henn, Bernhard Korte, and Werner Oettli, Berlin, Heidelberg: Springer Berlin Heidelberg, 1978, pp. 161–172.

[Kle22] Jakub Klemsa, “Hitchhiker’s Guide to a Practical Automated TFHE Parameter Setup for Custom Applications”, *in*: *IACR Cryptol. ePrint Arch.* (2022), p. 1315, URL: `https://eprint.iacr.org/2022/1315`.

[Lee+20a] DongKwon Lee, Woosuk Lee, Hakjoo Oh, and Kwangkeun Yi, “Optimizing homomorphic evaluation circuits by program synthesis and term rewriting”, *in*: *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, ed. by Alastair F. Donaldson and Emina Torlak, ACM, 2020, pp. 503–518, DOI: `10.1145/3385412.3385996`, URL: `https://doi.org/10.1145/3385412.3385996`.

[Mon+22] Johannes Mono, Chiara Marcolla, Georg Land, Tim Güneysu, and Najwa Aaraj, “Finding and Evaluating Parameters for BGV”, *in*: *IACR Cryptol. ePrint Arch.* (2022), p. 706, URL: `https://eprint.iacr.org/2022/706`.

[Via+22] Alexander Viand, Patrick Jattke, Miro Haller, and Anwar Hithnawi, “HECO: Fully Homomorphic Encryption Compiler”, *in*: *USENIX Security Symposium*, 2022, URL: `https://api.semanticscholar.org/CorpusID:257365804`.

[VJH21] Alexander Viand, Patrick Jattke, and Anwar Hithnawi, “SoK: Fully Homomorphic Encryption Compilers”, *in*: *CoRR* (2021), URL: `https://arxiv.org/abs/2101.07078`.

My Contributions

[Ber+23a] Loris Bergerat, Anas Boudi, Quentin Bourgerie, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap, "Parameter Optimization and Larger Precision for (T)FHE", *in*: *Journal of Cryptology* 36.*3* (2023), p. 28, DOI: 10.1007/s00145-023-09463-5, URL: https://doi.org/10.1007/s00145-023-09463-5.

[Ber+23b] Loris Bergerat, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, Adeline Roux-Langlois, and Samuel Tap, *Faster Secret Keys for (T)FHE*, Cryptology ePrint Archive, Paper 2023/979, 2023, URL: https://eprint.iacr.org/2023/979.

[Chi+20b] Ilaria Chillotti, Marc Joye, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap, "CONCRETE: Concrete operates on ciphertexts rapidly by extending TfhE", *in*: *WAHC 2020*, 2020.

[Chi+21] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap, "Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for tfhe", *in*: *ASIACRYPT 2021*, Springer, 2021.

[Dah+23] Morten Dahl, Daniel Demmler, Sarah El Kazdadi, Arthur Meyre, Jean-Baptiste Orfila, Dragos Rotaru, Nigel P. Smart, Samuel Tap, and Michael Walter, "Noah's Ark: Efficient Threshold-FHE Using Noise Flooding", *in*: *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC '23, Copenhagen, Denmark: Association for Computing Machinery, 2023, pp. 35–46, ISBN: 9798400702556, DOI: 10.1145/3605759.3625259, URL: https://doi.org/10.1145/3605759.3625259.

Appendix A

Appendix

A.1 Noise Analysis of the GLWE Multiplication

In this section, we provide details for the noise analysis of the multiplication described in Algorithm 20. We start by providing some basic notations and analysis for the distributions composing the different elements that are called in the multiplication. We then provide a noise analysis for the two parts of the multiplication, i.e. the **tensor product** in Section A.1.4 and the **relinearization** in Section A.1.6.

A.1.1 Notation.

Let q be a ciphertext modulus such that q is a power-of-two. We use the notation $[\cdot]_q$ to indicate a centered modular reduction, i.e., modulo q with representatives chosen in $[\![-q/2, q/2[\![\subset \mathbb{Z}$. When we modulus switch an integer value $x \in \mathbb{Z}$ from q to Q with $q < Q \in \mathbb{N}$, the following conversions are used:

$$\begin{cases} [x]_q \xrightarrow{q \to Q} [x]_Q \\ [x]_Q \xrightarrow{Q \to q} [x]_q + q \cdot U \quad \text{where} \quad U \in \mathbb{Z} \end{cases}$$

Then, we have the following properties for polynomials $P(X) \in \mathfrak{R}$:

$$\begin{cases} [P(X)]_q \xrightarrow{q \to Q} [P(X)]_Q \\ [P(X)]_Q \xrightarrow{Q \to q} [P(X)]_q + q \cdot U(X) \quad \text{where} \quad U(X) \in \mathfrak{R} \end{cases}$$

When doing operations between polynomials:

$$\sum_{i=1}^{n} \left[P^{(i)}(X)\right]_q = \left[\sum_{i=1}^{n} P^{(i)}(X)\right]_q + q \cdot U^{+}(X)$$

$$\prod_{i=1}^{n} \left[P^{(i)}(X)\right]_q = \left[\prod_{i=1}^{n} P^{(i)}(X)\right]_q + q \cdot U^{*}(X)$$

Note that U^+ and U^* are integer polynomials. Their distribution depends on the distribution of polynomials $P^{(1)}, \cdots, P^{(n)}$.

A.1.2 Uniform distributions in a fixed interval.

Let Δ be a scaling factor such that $\Delta/2$ is an integer. Consider the variance and expectation of random variables from uniform intervals:

- Let a_i be a uniform variable in $[\![-\Delta/2, \Delta/2[\![\subset \mathbb{Z}$. Then:

$$a_i \in \mathcal{U}([\![-\Delta/2, \Delta/2[\![) \qquad \begin{cases} \mathbb{E}(a_i) = -1/2 \\ \mathsf{Var}(a_i) = (\Delta^2 - 1)/12 \end{cases}$$

- Let a_i be a uniform variable in $[\![-q/2, q/2[\![\subset \mathbb{Z}$. Then:

$$a_i \in \mathcal{U}([\![-q/2, q/2[\![) \qquad \begin{cases} \mathbb{E}(a_i) = -1/2 \\ \mathsf{Var}(a_i) = (q^2 - 1)/12 \end{cases}$$

A.1.3 Secret keys probability distributions.

We analyze the probability distributions of the secret keys and their combinations that appear in the multiplication. We start by observing some basic probability distributions for uniform binary, uniform ternary and Gaussian keys in Table A.1.

	Binary	Ternary	Gaussian
$\mathsf{Var}(s_i)$	$1/4$	$2/3$	σ^2
$\mathbb{E}(s_i)$	$1/2$	0	0
$\mathsf{Var}(s_i^2)$	$1/4$	$2/9$	$2\sigma^4$
$\mathbb{E}(s_i^2)$	$1/2$	$2/3$	σ^2
$\mathsf{Var}(s_i s_j)$	$3/16$	$4/9$	σ^4
$\mathbb{E}(s_i s_j)$	$1/4$	0	0

Table A.1: Variance and expectation of s_i, s_i^2 and $s_i s_j$ with s_i and s_j independently drawn from the distribution $\mathcal{D}$ and $\mathcal{D}$ is either uniform binary, uniform ternary or Gaussian.

Distribution of $S_i(X)S_j(X)$: $i = j$ case. We consider a polynomial $S(X) = \sum_{i=0}^{N-1} s_i \cdot X^i \in \mathfrak{R}_q$ and $S'(X) = S^2(X) = \sum_{\alpha=0}^{N-1} s'_\alpha \cdot X^\alpha = \sum_{k=0}^{(N-2)/2} s'_{2k} \cdot X^{2k} + \sum_{k=0}^{(N-2)/2} s'_{2k+1} \cdot X^{2k+1} \in \mathfrak{R}_q$. We have each s_i independently sampled from the same distribution $\mathcal{D}$ of variance $\sigma_{\mathcal{D}}^2$ and expectation $\mu_{\mathcal{D}}$.

$$\begin{cases} \mathbb{E}(s'_\alpha) = \mathbb{E}\left(s_i \cdot s_j\right) \cdot (2\alpha - N + 2) \\ \mathsf{Var}(s'_{2k}) = 2 \cdot \mathsf{Var}\left(s_k^2\right) + 2 \cdot (N-2) \cdot \mathsf{Var}\left(s_i \cdot s_j\right) \\ \mathsf{Var}(s'_{2k+1}) = 2 \cdot N \cdot \mathsf{Var}\left(s_i \cdot s_j\right) \end{cases}$$

Proof 40 *Let's start by focusing on the even terms:*

$$\begin{aligned} s'_\alpha := s'_{2k} &= \sum_{i+j=2k} s_i \cdot s_j \\ &= s_k^2 - s_{\frac{N}{2}+k}^2 + \sum_{i+j=2k}^{i\neq j} s_i \cdot s_j = s_k^2 - s_{\frac{N}{2}+k}^2 + \sum_{i+j=2k<N}^{i\neq j} s_i \cdot s_j - \sum_{i+j=2k\geq N}^{i\neq j} s_i \cdot s_j \\ &= s_k^2 - s_{\frac{N}{2}+k}^2 + \sum_{i+j=2k<N}^{i\neq j, i<j} 2 \cdot s_i \cdot s_j - \sum_{i+j=2k\geq N}^{i\neq j, i<j} 2 \cdot s_i \cdot s_j \end{aligned}$$

Observe that all the terms are independent since the pairs (i, j) are exclusive in each sum. The variance is:

$$\begin{aligned} \mathsf{Var}(s'_\alpha) := \mathsf{Var}(s'_{2k}) &= \mathsf{Var}\left(s_k^2 - s_{\frac{N}{2}+k}^2 + \sum_{i+j=2k<N}^{i\neq j, i<j} 2 \cdot s_i \cdot s_j - \sum_{i+j=2k\geq N}^{i\neq j, i<j} 2 \cdot s_i \cdot s_j\right) \\ &= \mathsf{Var}\left(s_k^2\right) + \mathsf{Var}\left(s_{\frac{N}{2}+k}^2\right) + \mathsf{Var}\left(\sum_{i+j=2k<N}^{i\neq j, i<j} 2 \cdot s_i \cdot s_j\right) + \mathsf{Var}\left(\sum_{i+j=2k\geq N}^{i\neq j, i<j} 2 \cdot s_i \cdot s_j\right) \\ &= 2 \cdot \mathsf{Var}\left(s_k^2\right) + \underbrace{\sum_{i+j=2k<N}^{i\neq j, i<j} 4 \cdot \mathsf{Var}\left(s_i \cdot s_j\right)}_{k \textit{ terms}} + \underbrace{\sum_{i+j=2k\geq N}^{i\neq j, i<j} 4 \cdot \mathsf{Var}\left(s_i \cdot s_j\right)}_{\frac{N-2}{2}-k \textit{ terms}} \\ &= 2 \cdot \mathsf{Var}\left(s_k^2\right) + \underbrace{\sum_{i+j=2k}^{i\neq j, i<j} 4 \cdot \mathsf{Var}\left(s_i \cdot s_j\right)}_{\frac{N-2}{2} \textit{ terms}} = 2 \cdot \mathsf{Var}\left(s_k^2\right) + \frac{N-2}{2} \cdot 4 \cdot \mathsf{Var}\left(s_i \cdot s_j\right) \\ &= 2 \cdot \mathsf{Var}\left(s_k^2\right) + 2 \cdot (N-2) \cdot \mathsf{Var}\left(s_i \cdot s_j\right) \ . \end{aligned}$$

The expectation is:

$$
\begin{aligned}
\mathbb{E}(s'_\alpha) := \mathbb{E}(s'_{2k}) &= \mathbb{E}\left(s_k^2 - s^2_{\frac{N}{2}+k} + \sum_{i+j=2k<N}^{i\neq j, i<j} 2\cdot s_i \cdot s_j - \sum_{i+j=2k\geq N}^{i\neq j, i<j} 2\cdot s_i\cdot s_j\right)\\
&= \cancel{\mathbb{E}\left(s_k^2\right)} - \cancel{\mathbb{E}\left(s^2_{\frac{N}{2}+k}\right)} + \mathbb{E}\left(\sum_{i+j=2k<N}^{i\neq j, i<j} 2\cdot s_i\cdot s_j\right) - \mathbb{E}\left(\sum_{i+j=2k\geq N}^{i\neq j, i<j} 2\cdot s_i\cdot s_j\right)\\
&= \underbrace{\sum_{i+j=2k<N}^{i\neq j, i<j} 2\cdot \mathbb{E}\left(s_i\cdot s_j\right)}_{k \text{ terms}} - \underbrace{\sum_{i+j=2k\geq N}^{i\neq j, i<j} 2\cdot\mathbb{E}\left(s_i\cdot s_j\right)}_{\frac{N-2}{2}-k \text{ terms}}\\
&= 2\cdot k\cdot \mathbb{E}\left(s_i\cdot s_j\right) - 2\cdot\left(\frac{N-2}{2}-k\right)\cdot\mathbb{E}\left(s_i\cdot s_j\right)\\
&= 2\cdot\mathbb{E}\left(s_i\cdot s_j\right)\cdot(k - \frac{N-2}{2}+k) = \mathbb{E}\left(s_i\cdot s_j\right)\cdot(4k-N-2)\\
&= \mathbb{E}\left(s_i\cdot s_j\right)\cdot(2\alpha - N+2)\ .
\end{aligned}
$$

Now, let's focus on the odd coefficients.

$$
\begin{aligned}
s'_\alpha := s'_{2k+1} &= \sum_{i+j=2k+1<N} s_i\cdot s_j - \sum_{i+j=2k+1\geq N} s_i\cdot s_j\\
&= \sum_{i+j=2k+1<N}^{i<j} 2s_i\cdot s_j - \sum_{i+j=2k+1\geq N}^{i<j} 2s_i\cdot s_j\ .
\end{aligned}
$$

Observe again that all the terms are independent since the couples (i, j) are exclusive in each sum. The variance is

$$
\begin{aligned}
\mathsf{Var}(s'_\alpha) := \mathsf{Var}(s'_{2k+1}) &= \mathsf{Var}\left(\sum_{i+j=2k+1<N}^{i<j} 2s_i\cdot s_j - \sum_{i+j=2k+1\geq N}^{i<j} 2s_i\cdot s_j\right)\\
&= \mathsf{Var}\left(\sum_{i+j=2k+1<N}^{i<j} 2s_i\cdot s_j\right) + \mathsf{Var}\left(\sum_{i+j=2k+1\geq N}^{i<j} 2s_i\cdot s_j\right)\\
&= \underbrace{4\cdot\sum_{i+j=2k+1<N}^{i<j}\mathsf{Var}\left(s_i\cdot s_j\right)}_{k+1 \text{ terms}} + \underbrace{4\cdot\sum_{i+j=2k+1\geq N}^{i<j}\mathsf{Var}\left(s_i\cdot s_j\right)}_{\frac{N}{2}-(k+1) \text{ terms}}\\
&= 4\cdot(k+1)\cdot\mathsf{Var}\left(s_i\cdot s_j\right) + 4\cdot(\frac{N}{2}-k-1)\cdot\mathsf{Var}\left(s_i\cdot s_j\right)\\
&= 4\cdot\mathsf{Var}\left(s_i\cdot s_j\right)\cdot(k+1+\frac{N}{2}-k-1) = 4\cdot\mathsf{Var}\left(s_i\cdot s_j\right)\cdot\frac{N}{2}\\
&= 2\cdot N\cdot\mathsf{Var}\left(s_i\cdot s_j\right)\ .
\end{aligned}
$$

The expectation is

$$
\begin{aligned}
\mathbb{E}(s'_\alpha) := \mathbb{E}(s'_{2k+1}) &= \mathbb{E}\left(\sum_{i+j=2k+1<N}^{i<j} 2s_i \cdot s_j - \sum_{i+j=2k+1\geq N}^{i<j} 2s_i \cdot s_j\right) \\
&= \mathbb{E}\left(\sum_{i+j=2k+1<N}^{i<j} 2s_i \cdot s_j\right) - \mathbb{E}\left(\sum_{i+j=2k+1\geq N}^{i<j} 2s_i \cdot s_j\right) \\
&= \underbrace{2 \cdot \sum_{i+j=2k+1<N}^{i<j} \mathbb{E}\left(s_i \cdot s_j\right)}_{k+1 \ terms} \underbrace{-2 \cdot \sum_{i+j=2k+1\geq N}^{i<j} \mathbb{E}\left(s_i \cdot s_j\right)}_{\frac{N}{2}-(k+1) \ terms} \\
&= 2 \cdot (k+1) \cdot \mathbb{E}\left(s_i \cdot s_j\right) - 2 \cdot \left(\frac{N}{2} - k - 1\right) \cdot \mathbb{E}\left(s_i \cdot s_j\right) \\
&= 2 \cdot \mathbb{E}\left(s_i \cdot s_j\right)\left(k + 1 - \frac{N}{2} + k + 1\right) \\
&= \mathbb{E}\left(s_i \cdot s_j\right)(4k + 4 - N) = \mathbb{E}\left(s_i \cdot s_j\right)(2\alpha + 2 - N) \ .
\end{aligned}
$$

$\square$

In particular, in the case of a uniform binary, uniform ternary or Gaussian distribution, the squared secret keys have coefficients distributed as shown in Table A.2.

	Binary	Ternary	Gaussian
$\mathbb{E}(s'_\alpha)$	$\frac{1}{4} \cdot (2\alpha - N + 2)$	0	0
$\mathbb{E}^2(s'_{\mathsf{mean}}) = \mathsf{mean}(\{\mathbb{E}^2(s'_\alpha)\}_{\alpha=0}^{N-1})$	$\frac{(N^2+2)}{48}$	0	0
$\mathsf{Var}(s'_{2k})$	$\frac{3}{8} \cdot N - \frac{1}{4}$	$(2N-3) \cdot \frac{4}{9}$	$2N \cdot \sigma^4$
$\mathsf{Var}(s'_{2k+1})$	$\frac{3}{8} \cdot N$	$N \cdot \frac{8}{9}$	$2N \cdot \sigma^4$

Table A.2: General formula applied to polynomials with binary, ternary and Gaussian distributions. These formulae are true for N a power of 2, $N \neq 1$.

For the Binary case, the following proof shows that:

$$
\sum_{\alpha=0}^{N-1} \mathbb{E}^2(s'_\alpha) = \left(\sum_{i=0}^{N/2-1} \mathbb{E}^2(s'_{2i+1}) + \sum_{i=0}^{N/2-1} \mathbb{E}^2(s'_{2i})\right) = N \cdot \frac{(N^2+2)}{48}
$$

This means that in average, the expectation of a coefficient of the squared binary key is $\sqrt{(N^2+2)/48}$.

Proof 41

$$\begin{aligned}
\left(\sum_{i=0}^{N/2-1}\mathbb{E}^2(s'_{2i+1})+\sum_{i=0}^{N/2-1}\mathbb{E}^2(s'_{2i})\right) &= \sum_{k=0}^{N-1}\mathbb{E}^2(s'_k)\\
&= \sum_{k=0}^{N-1}\left(\frac{k+1}{2}-\frac{N}{4}\right)^2\\
&= \sum_{k=0}^{N-1}\left(\frac{(k+1)^2}{4}-2\cdot\frac{k+1}{2}\cdot\frac{N}{4}+\frac{N^2}{16}\right)\\
&= \sum_{k=0}^{N-1}\left(\frac{k^2}{4}+\frac{1}{4}+\frac{2k}{4}-\frac{kN}{4}-\frac{N}{4}+\frac{N^2}{16}\right)\\
&= N\cdot\left(\frac{1}{4}-\frac{N}{4}+\frac{N^2}{16}\right)+\sum_{k=0}^{N-1}\left(\frac{k^2}{4}+\frac{k}{2}-\frac{kN}{4}\right)\\
&= \frac{N}{4}-\frac{N^2}{4}+\frac{N^3}{16}+\frac{1}{4}\cdot\sum_{k=0}^{N-1}k^2+\left(\frac{1}{2}-\frac{N}{4}\right)\cdot\sum_{k=0}^{N-1}k\\
&= \frac{N}{4}-\frac{N^2}{4}+\frac{N^3}{16}+\\
&\quad+\frac{1}{4}\cdot\frac{(N-1)N(2(N-1)+1)}{6}+\left(\frac{1}{2}-\frac{N}{4}\right)\cdot\frac{(N-1)N}{2}\\
&= \frac{N(N^2+2)}{48}
\end{aligned}$$

□

Distribution of $S_i(X)S_j(X)$: $i \neq j$ case. We consider two polynomials $S_i(X) = \sum_{k=0}^{N-1} S_{i,k}\cdot X^k \in \mathfrak{R}_q$ and $S_j(X) = \sum_{k=0}^{N-1} S_{j,k}\cdot X^k \in \mathfrak{R}_q$. We note as $S''(X) = S_i(X)\cdot S_j(X)$.

We have:

$$\begin{aligned}
S''(X) = S_i(X)\cdot S_j(X) &= \sum_{\alpha=0}^{N-1} S''_\alpha\cdot X^\alpha\\
&= \sum_{\alpha=0}^{N-1}\left(\sum_{h+k=\alpha<N} S_{i,h}\cdot S_{j,k}-\sum_{h+k=\alpha\geq N} S_{i,h}\cdot S_{j,k}\right)\cdot X^\alpha \in \mathfrak{R}_q\ .
\end{aligned}$$

We have each coefficient of the two secret keys independently sampled from the same distribution $\mathcal{D}$ of variance $\sigma^2_{\mathcal{D}}$ and expectation $\mu_{\mathcal{D}}$.

$$\begin{cases}\mathbb{E}(S''_\alpha) = \mathbb{E}\left(S_{i,h}\cdot S_{j,k}\right)\cdot(2\alpha+2-N)\\ \mathsf{Var}(S''_\alpha) = N\cdot\mathsf{Var}\left(S_{i,h}\cdot S_{j,k}\right)\end{cases}$$

Proof 42 *Let $S_i, S_j \in \mathfrak{R}$ be two independent keys following the same distribution (binary, ternary or Gaussian). Then*

$$S_i \cdot S_j = \sum_{\alpha=1}^{N} S''_\alpha \cdot X^\alpha = \sum_{\alpha=1}^{N} \left(\sum_{h+k=\alpha<N} S_{i,h} \cdot S_{j,k} - \sum_{h+k=\alpha\geq N} S_{i,h} \cdot S_{j,k} \right) \cdot X^\alpha$$

Observe that all the terms in the sum are independent. The variance is

$$\begin{aligned}
\mathsf{Var}(S''_\alpha) &= \mathsf{Var}\left(\sum_{h+k=\alpha<N} S_{i,h} \cdot S_{j,k} - \sum_{h+k=\alpha\geq N} S_{i,h} \cdot S_{j,k} \right) \\
&= \mathsf{Var}\left(\sum_{h+k=\alpha<N} S_{i,h} \cdot S_{j,k} \right) + \mathsf{Var}\left(\sum_{h+k=\alpha\geq N} S_{i,h} \cdot S_{j,k} \right) \\
&= \sum_{h+k=\alpha<N} \mathsf{Var}\left(S_{i,h} \cdot S_{j,k}\right) + \sum_{h+k=\alpha\geq N} \mathsf{Var}\left(S_{i,h} \cdot S_{j,k}\right) \\
&= \sum_{h+k=\alpha[N]} \mathsf{Var}\left(S_{i,h} \cdot S_{j,k}\right) \\
&= N \cdot \mathsf{Var}\left(S_{i,h} \cdot S_{j,k}\right) \ .
\end{aligned}$$

The expectation is

$$\begin{aligned}
\mathbb{E}(S''_\alpha) &= \mathbb{E}\left(\sum_{h+k=\alpha<N} S_{i,h} \cdot S_{j,k} - \sum_{h+k=\alpha\geq N} S_{i,h} \cdot S_{j,k} \right) \\
&= \mathbb{E}\left(\sum_{h+k=\alpha<N} S_{i,h} \cdot S_{j,k} \right) - \mathbb{E}\left(\sum_{h+k=\alpha\geq N} S_{i,h} \cdot S_{j,k} \right) \\
&= \underbrace{\sum_{h+k=\alpha<N} \mathbb{E}\left(S_{i,h} \cdot S_{j,k}\right)}_{\alpha+1 \ \textit{terms}} - \underbrace{\sum_{h+k=\alpha\geq N} \mathbb{E}\left(S_{i,h} \cdot S_{j,k}\right)}_{N-(\alpha+1) \ \textit{terms}} \\
&= (\alpha+1) \cdot \mathbb{E}\left(S_{i,h} \cdot S_{j,k}\right) - (N-(\alpha+1)) \cdot \mathbb{E}\left(S_{i,h} \cdot S_{j,k}\right) \\
&= \mathbb{E}\left(S_{i,h} \cdot S_{j,k}\right) \cdot (\alpha+1-N+\alpha+1) \\
&= \mathbb{E}\left(S_{i,h} \cdot S_{j,k}\right) \cdot (2\alpha+2-N) \ .
\end{aligned}$$

□

In particular, in the case of a uniform binary, uniform ternary or Gaussian distribution, the product secret keys have coefficients distributed as shown in Table A.3.

For the Binary case, we can observe that:

$$\sum_{\alpha=0}^{N-1} \mathbb{E}^2(S''_\alpha) = N \cdot \frac{(N^2+2)}{48}$$

This means that in average, the expectation of a coefficient of the squared binary key is $\sqrt{(N^2+2)/48}$.

	Binary	Ternary	Gaussian
$\mathbb{E}(S''_\alpha)$	$\frac{1}{4}(2\alpha+2-N)$	0	0
$\mathbb{E}^2(S''_{\mathsf{mean}}) = \mathsf{mean}(\{\mathbb{E}^2(S''_\alpha)\}_{\alpha=0}^{N-1})$	$\frac{N^2+2}{48}$	0	0
$\mathsf{Var}(S''_\alpha)$	$\frac{3}{16}\cdot N$	$\frac{4}{9}\cdot N$	$\sigma^4\cdot N$

Table A.3: General formula applied to polynomials with binary, ternary and Gaussian distributions.

Proof 43

$$\begin{aligned}
\sum_{\alpha=0}^{N-1}\mathbb{E}^2(S''_\alpha) &= \sum_{\alpha=0}^{N-1}\left(\frac{1}{4}(2\alpha+2-N)\right)^2\\
&= \frac{1}{16}\cdot\sum_{\alpha=0}^{N-1}(2\alpha+2-N)^2\\
&= \frac{1}{16}\cdot\sum_{\alpha=0}^{N-1}\left(4\alpha^2+4+8\alpha+N^2-4N\alpha-4N\right)\\
&= \frac{1}{16}\cdot\left(4N+N^3-4N^2+4\sum_{\alpha=0}^{N-1}\alpha^2+(8-4N)\sum_{\alpha=0}^{N-1}\alpha\right)\\
&= \frac{1}{16}\cdot\left(4N+N^3-4N^2+4\frac{(N-1)N(2(N-1)+1)}{6}+(8-4N)\frac{(N-1)N}{2}\right)\\
&= \frac{N(N^2+2)}{48}
\end{aligned}$$

□

A.1.4 Tensor product

We perform a GLWE multiplication with dense messages having different scaling factors. The inputs are two GLWE ciphertexts modulo q:

$$\mathsf{CT}^{(1)} = (A_1^{(1)},\cdots,A_k^{(1)},B^{(1)} = \sum_{i=1}^{k}A_i^{(1)}\cdot S_i + E_1 + P_1)\in\mathfrak{R}_q^{k+1}$$
$$\mathsf{CT}^{(2)} = (A_1^{(2)},\cdots,A_k^{(2)},B^{(2)} = \sum_{i=1}^{k}A_i^{(2)}\cdot S_i + E_2 + P_2)\in\mathfrak{R}_q^{k+1}$$

such that

- $(S_1,\cdots,S_k)\in\mathfrak{R}^k$ is the secret key made of polynomials with coefficients either sampled from a uniform binary, uniform ternary or gaussian distribution,
- $\{A_i^{(1)}\}_{i=1}^k$ and $\{A_i^{(2)}\}_{i=1}^k$ are polynomials in $\mathfrak{R}_q$ with coefficients sampled from $\mathcal{U}(\llbracket -q/2, q/2\rrbracket)$,

- E_1, E_2 are error polynomials in $\mathfrak{R}_q$ such that their coefficients are sampled from Gaussian distributions $\chi_{\sigma_1}, \chi_{\sigma_2}$ respectively,
- $P_1 = \lfloor \Delta_1 \cdot M_1 \rceil_q$ and $P_2 = \lfloor \Delta_2 \cdot M_2 \rceil_q$, with $M_1, M_2 \in \mathbb{T}_N[X]$ and Δ_1 and Δ_2 the scaling factors.

The first step (modulus switching, tensor product, rescale, round and modulo) is to compute:

$$\begin{aligned}
T'_i &= \left[\left\lfloor \frac{\left[A_i^{(1)} \cdot A_i^{(2)}\right]_Q}{\Delta} \right\rceil\right]_q && \text{depending on } S_i^2 && k \text{ terms}\\
R'_{i,j} &= \left[\left\lfloor \frac{\left[A_i^{(1)} \cdot A_j^{(2)} + A_j^{(1)} \cdot A_i^{(2)}\right]_Q}{\Delta} \right\rceil\right]_q && \text{depending on } S_i \cdot S_j && \frac{k(k-1)}{2} \text{ terms}\\
A'_i &= \left[\left\lfloor \frac{\left[A_i^{(1)} \cdot B^{(2)} + B^{(1)} \cdot A_i^{(2)}\right]_Q}{\Delta} \right\rceil\right]_q && \text{depending on } S_i && k \text{ terms}\\
B' &= \left[\left\lfloor \frac{[B_1 \cdot B_2]_Q}{\Delta} \right\rceil\right]_q && \text{constant term} && 1 \text{ term}
\end{aligned}$$

where $\Delta = \mathsf{min}(\Delta_1, \Delta_2)$. The intermediate result of this step are the polynomials:

$$\begin{aligned}
[T_i]_Q &= A_i^{(1)} \cdot A_i^{(2)} = \left[A_i^{(1)} \cdot A_i^{(2)}\right]_q + q \cdot U_{T_i} && \in \mathfrak{R}_Q\\
[R_{i,j}]_Q &= A_i^{(1)} \cdot A_j^{(2)} + A_j^{(1)} \cdot A_i^{(2)} = \left[A_i^{(1)} \cdot A_j^{(2)} + A_j^{(1)} \cdot A_i^{(2)}\right]_q + q \cdot U_{R_{i,j}} && \in \mathfrak{R}_Q\\
[A_i]_Q &= A_i^{(1)} \cdot B^{(2)} + B^{(1)} \cdot A_i^{(2)} = \left[A_i^{(1)} \cdot B^{(2)} + B^{(1)} \cdot A_i^{(2)}\right]_q + q \cdot U_{A_i} && \in \mathfrak{R}_Q\\
[B]_Q &= B^{(1)} \cdot B^{(2)} = \left[B^{(1)} \cdot B^{(2)}\right]_q + q \cdot U_B && \in \mathfrak{R}_Q
\end{aligned}$$

These operations are performed in large precision $Q = q^2$. The terms $U_{T_i}, U_{R_{i,j}}, U_{A_i}, U_B$ are in $\mathfrak{R}$. Then the polynomials are rescaled by Δ and rounded modulo q:

$$\begin{aligned}
[T'_i]_q &= \left[\left\lfloor \frac{[T_i]_Q}{\Delta} \right\rceil\right]_q = \left[\frac{[T_i]_Q}{\Delta} + \overline{T_i}\right]_q\\
[R'_{i,j}]_q &= \left[\left\lfloor \frac{[R_{i,j}]_Q}{\Delta} \right\rceil\right]_q = \left[\frac{[R_{i,j}]_Q}{\Delta} + \overline{R_{i,j}}\right]_q\\
[A'_i]_q &= \left[\left\lfloor \frac{[A_i]_Q}{\Delta} \right\rceil\right]_q = \left[\frac{[A_i]_Q}{\Delta} + \overline{A_i}\right]_q\\
[B']_q &= \left[\left\lfloor \frac{[B]_Q}{\Delta} \right\rceil\right]_q = \left[\frac{[B]_Q}{\Delta} + \overline{B}\right]_q
\end{aligned}$$

such that $\overline{T_i}, \overline{R_{i,j}}, \overline{A_i}, \overline{B}$ are the rounding errors in $\frac{\mathcal{U}(\llbracket -\Delta/2, \Delta/2 \rrbracket)}{\Delta}$. Let's compute the noise growth generated by these operations. In order to estimate it, we have to decrypt. Let $\vec{S}_R = (S_1 \cdot S_2, S_1 \cdot S_3, \cdots, S_{k-1} \cdot S_k) \in \mathfrak{R}_q^{\frac{(k-1)k}{2}}$ and $\vec{S}_T = (S_1^2, S_2^2, \cdots, S_k^2)$ such that $(\vec{S}_R || \vec{S}_T) = \vec{S} \otimes \vec{S}$.

$$
\begin{aligned}
&\mathsf{TensorDec}(\vec{T}', \vec{R}', \vec{A}', B') = \\
&\quad = B' - \vec{A}' \cdot \vec{S} + \vec{R}' \cdot \vec{S}_R + \vec{T}' \cdot \vec{S}_T && \in \mathfrak{R}_q \\
&\quad = \frac{[B]_Q}{\Delta} + \overline{B} - \left(\frac{\left[\vec{A}\right]_Q}{\Delta} + \overline{\vec{A}} \right) \cdot \vec{S} + \left(\frac{\left[\vec{R}\right]_Q}{\Delta} + \overline{\vec{R}} \right) \cdot \vec{S}_R + \left(\frac{\left[\vec{T}\right]_Q}{\Delta} + \overline{\vec{T}} \right) \cdot \vec{S}_T && \in \mathfrak{R}_q \\
&\quad = \frac{\left([B]_Q - \left[\vec{A}\right]_Q \cdot \vec{S} + \left[\vec{R}\right]_Q \cdot \vec{S}_R + \left[\vec{T}\right]_Q \cdot \vec{S}_T \right)}{\Delta} + \left(\overline{B} - \overline{\vec{A}} \cdot \vec{S} + \overline{\vec{R}} \cdot \vec{S}_R + \overline{\vec{T}} \cdot \vec{S}_T \right) && \in \mathfrak{R}_q
\end{aligned}
$$

So now, let's analyze the term:

$$
\begin{aligned}
&[B]_Q - \left[\vec{A}\right]_Q \cdot \vec{S} + \left[\vec{R}\right]_Q \cdot \vec{S}_R + \left[\vec{T}\right]_Q \cdot \vec{S}_T = \\
&\quad = B^{(1)} B^{(2)} - (\vec{A}^{(1)} B^{(2)} + \vec{A}^{(2)} B^{(1)}) \cdot \vec{S} + \\
&\qquad + \sum_{i=1}^{k} \sum_{j=1}^{i-1} (A_i^{(1)} \cdot A_j^{(2)} + A_j^{(1)} \cdot A_i^{(2)}) \cdot S_i S_j + \sum_{i=1}^{k} (A_i^{(1)} \cdot A_i^{(2)}) \cdot S_i^2 && \in \mathfrak{R}_q \\
&\quad = B^{(1)} B^{(2)} - (\vec{A}^{(1)} B^{(2)} + \vec{A}^{(2)} B^{(1)}) \cdot \vec{S} + \sum_{i=1}^{k} \sum_{j=1}^{k} (A_i^{(1)} \cdot A_j^{(2)}) \cdot S_i S_j && \in \mathfrak{R}_q \\
&\quad = B^{(1)} B^{(2)} - (\vec{A}^{(1)} B^{(2)} + \vec{A}^{(2)} B^{(1)}) \cdot \vec{S} + (\vec{A}^{(1)} \otimes \vec{A}^{(2)}) \cdot (\vec{S} \otimes \vec{S}) && \in \mathfrak{R}_q
\end{aligned}
\tag{A.1}
$$

Now, let's observe the following relations:

$$
\begin{aligned}
&(B^{(1)} - \vec{A}^{(1)} \cdot \vec{S})(B^{(2)} - \vec{A}^{(2)} \cdot \vec{S}) = \\
&\quad = B^{(1)} B^{(2)} - (\vec{A}^{(1)} B^{(2)} + \vec{A}^{(2)} B^{(1)}) \vec{S} + (\vec{A}^{(1)} \cdot \vec{S})(\vec{A}^{(2)} \cdot \vec{S}) \\
&\quad = B^{(1)} B^{(2)} - (\vec{A}^{(1)} B^{(2)} + \vec{A}^{(2)} B^{(1)}) \vec{S} + (\vec{A}^{(1)} \otimes \vec{A}^{(2)}) \cdot (\vec{S} \otimes \vec{S}) \quad \in \mathfrak{R}_Q
\end{aligned}
\tag{A.2}
$$

and

$$
\begin{aligned}
(B^{(1)} - \vec{A}^{(1)} \cdot \vec{S})(B^{(2)} - \vec{A}^{(2)} \cdot \vec{S}) &= (P_1 + E_1 + qU_1)(P_2 + E_2 + qU_2) && \in \mathfrak{R}_Q \\
&= P_1 P_2 + P_1 E_2 + P_2 E_1 + E_1 E_2 + \\
&\quad + q(P_1 U_2 + P_2 U_1 + E_1 U_2 + E_2 U_1) + q^2 U_1 U_2 && \in \mathfrak{R}_Q
\end{aligned}
\tag{A.3}
$$

Observe that the coefficients of S_i^2 and $S_i S_j$ are all $\leq N$, if the key is binary or ternary. Since $N < q$, we assume that they do not overlap modulo q. If the key is Gaussian, the coefficients will be a small factor of N in the worst case, and we still assume they do not overlap modulo q.

By putting together Equations A.1, A.2 and A.3, we obtain the following equality:

$$\begin{aligned}[B]_Q - \left[\vec{A}\right]_Q \cdot \vec{S} + \left[\vec{R}\right]_Q \cdot \vec{S}_R + \left[\vec{T}\right]_Q \cdot \vec{S}_T = {} & P_1P_2 + P_1E_2 + P_2E_1 + E_1E_2 + \\ & + q(P_1U_2 + P_2U_1 + E_1U_2 + E_2U_1) + \\ & + q^2U_1U_2 \qquad \in \mathfrak{R}_Q\end{aligned} \tag{A.4}$$

We then observe:

$$\begin{aligned}&\mathsf{TensorDec}(\vec{T'}, \vec{R'}, \vec{A'}, B') = \\ &= \frac{\left([B]_Q - \left[\vec{A}\right]_Q \cdot \vec{S} + \left[\vec{R}\right]_Q \cdot \vec{S}_R + \left[\vec{T}\right]_Q \cdot \vec{S}_T\right)}{\Delta} + \left(\overline{B} - \overline{\vec{A}} \cdot \vec{S} + \overline{\vec{R}} \cdot \vec{S}_R + \overline{\vec{T}} \cdot \vec{S}_T\right) \\ &= \frac{(P_1P_2 + P_1E_2 + P_2E_1 + E_1E_2)}{\Delta} + \frac{q\,(P_1U_2 + P_2U_1 + E_1U_2 + E_2U_1)}{\Delta} + \\ &\quad + \frac{q^2U_1U_2}{\Delta} + \left(\overline{B} - \overline{\vec{A}} \cdot \vec{S} + \overline{\vec{R}} \cdot \vec{S}_R + \overline{\vec{T}} \cdot \vec{S}_T\right) \\ &= \frac{(\Delta_1\Delta_2M_1M_2 + \Delta_1M_1E_2 + \Delta_2M_2E_1 + E_1E_2)}{\Delta} + \frac{q(\Delta_1M_1U_2 + \Delta_2M_2U_1 + E_1U_2 + E_2U_1)}{\Delta} + \\ &\quad + \frac{q^2U_1U_2}{\Delta} + \left(\overline{B} - \overline{\vec{A}} \cdot \vec{S} + \overline{\vec{R}} \cdot \vec{S}_R + \overline{\vec{T}} \cdot \vec{S}_T\right) \\ &= \Delta' M_1M_2 + \frac{\Delta_1}{\Delta}M_1E_2 + \frac{\Delta_2}{\Delta}M_2E_1 + \Delta^{-1}E_1E_2 + q\left(\frac{\Delta_1}{\Delta}M_1U_2 + \frac{\Delta_2}{\Delta}M_2U_1\right) + \\ &\quad + \frac{q}{\Delta}(E_1U_2 + E_2U_1) + q\frac{q}{\Delta}U_1U_2 + \left(\overline{B} - \overline{\vec{A}} \cdot \vec{S} + \overline{\vec{R}} \cdot \vec{S}_R + \overline{\vec{T}} \cdot \vec{S}_T\right) \quad \in \mathfrak{R}_q\end{aligned}$$

The value $\frac{q}{\Delta}$ is an integer smaller than q according to our parameter choices. So $\frac{q}{\Delta}$ will not overlap modulo q.

$$\begin{aligned}&\mathsf{TensorDec}(\vec{T'}, \vec{R'}, \vec{A'}, B') = \\ &= \Delta' M_1M_2 + \frac{\Delta_1}{\Delta}M_1E_2 + \frac{\Delta_2}{\Delta}M_2E_1 + \Delta^{-1}E_1E_2 + q\left(\frac{\Delta_1}{\Delta}M_1U_2 + \frac{\Delta_2}{\Delta}M_2U_1\right) + \\ &\quad + \frac{q}{\Delta}(E_1U_2 + E_2U_1) + q\frac{q}{\Delta}U_1U_2 + \left(\overline{B} - \overline{\vec{A}} \cdot \vec{S} + \overline{\vec{R}} \cdot \vec{S}_R + \overline{\vec{T}} \cdot \vec{S}_T\right) \\ &= \Delta' M_1M_2 + \frac{\Delta_1}{\Delta}M_1E_2 + \frac{\Delta_2}{\Delta}M_2E_1 + \Delta^{-1}E_1E_2 + \frac{q}{\Delta}(E_1U_2 + E_2U_1) + \\ &\quad + \left(\overline{B} - \overline{\vec{A}} \cdot \vec{S} + \overline{\vec{R}} \cdot \vec{S}_R + \overline{\vec{T}} \cdot \vec{S}_T\right) \quad \in \mathfrak{R}_q\end{aligned}$$

We extract the error:

$$\mathsf{Error}(\vec{T'}, \vec{R'}, \vec{A'}, B') = \underbrace{\frac{\Delta_1}{\Delta} M_1 E_2 + \frac{\Delta_2}{\Delta} M_2 E_1 + \Delta^{-1} E_1 E_2}_{(I)} + \\ + \underbrace{\frac{q}{\Delta}(E_1 U_2 + E_2 U_1)}_{(II)} + \underbrace{\left(\overline{B} - \overline{\vec{A}} \cdot \vec{S} + \overline{\vec{R}} \cdot \vec{S}_R + \overline{\vec{T}} \cdot \vec{S}_T\right)}_{(III)} \quad \in \mathfrak{R}_q$$

Let's analyze each term separately.

(I) We assume that $M_1 = \sum_{i=0}^{N-1} ||M_1||_\infty \cdot X^i$ and $M_2 = \sum_{i=0}^{N-1} ||M_2||_\infty \cdot X^i$.The variance of the first term is:

$$\begin{aligned} \mathsf{Var}(I) &= \mathsf{Var}\left(\frac{\Delta_1}{\Delta} M_1 E_2 + \frac{\Delta_2}{\Delta} M_2 E_1 + \Delta^{-1} E_1 E_2\right) \\ &= \mathsf{Var}\left(\frac{\Delta_1}{\Delta} M_1 E_2\right) + \mathsf{Var}\left(\frac{\Delta_2}{\Delta} M_2 E_1\right) + \mathsf{Var}(\Delta^{-1} E_1 E_2) \\ &= \frac{\Delta_1^2}{\Delta^2} N ||M_1||_\infty^2 \sigma_2^2 + \frac{\Delta_2^2}{\Delta^2} N ||M_2||_\infty^2 \sigma_1^2 + \frac{N}{\Delta^2} \sigma_1^2 \cdot \sigma_2^2 \\ &= \frac{N}{\Delta^2}\left(\Delta_1^2 ||M_1||_\infty^2 \sigma_2^2 + \Delta_2^2 ||M_2||_\infty^2 \sigma_1^2 + \sigma_1^2 \sigma_2^2\right) \end{aligned}$$

(II) Let's estimate the expectation and variance of U_1 and U_2. They come from the modulus switching adding two terms of error U_1 and U_2. They represent the part overlapping the modulo q.

Remember that, according to the RLWE assumptions, the $\vec{A}, B$ are insistinguishable from uniform in $\mathcal{U}(\llbracket -q/2, q/2 \rrbracket)$ with $\mathsf{CT} = (\vec{A}(X), B(X))$ an RLWE ciphertext.

Observe that:

$$\left[B - \sum_{i=1}^{k} A_i S_i\right]_Q = \left[B - \sum_{i=1}^{k} A_i S_i\right]_q + qU = [\Delta M + E + qU]_Q$$

We are looking for U. Then:

$$\frac{\left[B - \sum_{i=1}^{k} A_i S_i\right]_Q}{q} = \frac{\left[B - \sum_{i=1}^{k} A_i S_i\right]_q}{q} + U$$

We assume that $qU \approx \Delta M + E + qU$ since $\Delta M + E$ appears in the LSB. In practice it's like we did not cut the Gaussian's tails so we overestimate from here.

$$\frac{\left[B - \sum_{i=1}^{k} A_i S_i\right]_Q}{q} \approx U$$

In reality we study U as if we we were doing $\frac{B-\sum_{i=1}^{k} A_i S_i}{q}$ in $\mathbb{Z}$, it supposes in general that U is not bigger than Q/q so we can keep the modulo Q.

So we study the variance as follows:

$$\begin{aligned}
\mathsf{Var}\left(U\right) &= \mathsf{Var}\left(\frac{\left[B-\sum_{i=1}^{k} A_i S_i - \Delta M - E\right]_Q}{q}\right) \\
&\approx \mathsf{Var}\left(\frac{\left[B-\sum_{i=1}^{k} A_i S_i\right]_Q}{q}\right) \\
&= \frac{1}{q^2}\left(\mathsf{Var}(B) + \sum_{i=1}^{k} \mathsf{Var}(A_i S_i)\right) \\
&= \frac{1}{q^2}\left(\mathsf{Var}(B) + kN \cdot \mathsf{Var}(A_{i,j} \cdot S_{i,h})\right) \\
&= \frac{1}{q^2}\left(\mathsf{Var}(B_j) + kN \cdot \left(\mathsf{Var}(A_{i,j}) \cdot \mathsf{Var}(S_{i,h}) + \mathbb{E}^2(S_{i,h}) \cdot \mathsf{Var}(A_{i,j}) + \mathsf{Var}(S_{i,h}) \cdot \mathbb{E}^2(A_{i,j})\right)\right) \\
&= \frac{1}{q^2}\left(\frac{q^2-1}{12} + kN \cdot \left(\frac{q^2-1}{12} \cdot \mathsf{Var}(S_{i,h}) + \mathbb{E}^2(S_{i,h}) \cdot \frac{q^2-1}{12} + \mathsf{Var}(S_{i,h}) \cdot \frac{1}{4}\right)\right) \\
&= \frac{1}{q^2}\left(\frac{q^2-1}{12} \cdot \left(1 + kN \cdot \mathsf{Var}(S_{i,h}) + kN \cdot \mathbb{E}^2(S_{i,h})\right) + \frac{kN}{4} \cdot \mathsf{Var}(S_{i,h})\right)
\end{aligned}$$

And the expectation:

$$\begin{aligned}
\mathbb{E}\left(U\right) &= \mathbb{E}\left(\frac{\left[B-\sum_{i=1}^{k} A_i S_i - \Delta M - E\right]_Q}{q}\right) \\
&\approx \mathbb{E}\left(\frac{\left[B-\sum_{i=1}^{k} A_i S_i\right]_Q}{q}\right) \\
&= \frac{1}{q} \cdot \mathbb{E}\left(B - \sum_{i=1}^{k} A_i S_i\right) \\
&= \frac{1}{q} \cdot \left(\mathbb{E}(B) + \sum_{i=1}^{k} \mathbb{E}(A_i S_i)\right) \\
&= \frac{1}{q} \cdot \left(\mathbb{E}(B) + kN \cdot \mathbb{E}(A_{i,j}) \cdot \mathbb{E}(S_{i,h})\right) \\
&= -\frac{1}{2q} - \frac{kN}{2q} \cdot \mathbb{E}(S_{i,h}) \\
&= -\frac{1}{2q}\left(1 + kN \cdot \mathbb{E}(S_{i,h})\right)
\end{aligned}$$

By now, we note the coefficients $S_{i,h}$ of S_i, simply by S. Then:

$$\begin{aligned}
\mathsf{Var}(II) &= \mathsf{Var}\left(\frac{q}{\Delta}(E_1U_2 + E_2U_1)\right) \\
&= \frac{q^2}{\Delta^2} \cdot \mathsf{Var}\left(E_1U_2 + E_2U_1\right) \\
&= \frac{q^2}{\Delta^2} \cdot \left(\mathsf{Var}(E_1U_2) + \mathsf{Var}(E_2U_1)\right) \\
&= \frac{q^2}{\Delta^2} \cdot \left(N \cdot \mathsf{Var}(e_1u_2) + N \cdot \mathsf{Var}(e_2u_1)\right) \\
&= \frac{N \cdot q^2}{\Delta^2} \cdot \big(\mathsf{Var}(e_1)\mathsf{Var}(u_2) + \mathbb{E}^2(u_2)\mathsf{Var}(e_1) + \mathsf{Var}(e_2)\mathsf{Var}(u_1) + \mathbb{E}^2(u_1)\mathsf{Var}(e_2)\big) \\
&= \frac{N \cdot q^2}{\Delta^2} \cdot \big(\mathsf{Var}(e_1)\mathsf{Var}(u) + \mathbb{E}^2(u)\mathsf{Var}(e_1) + \mathsf{Var}(e_2)\mathsf{Var}(u) + \mathbb{E}^2(u)\mathsf{Var}(e_2)\big) \\
&= \frac{N \cdot q^2}{\Delta^2} \cdot \big((\mathsf{Var}(e_1) + \mathsf{Var}(e_2)) \cdot \mathsf{Var}(u) + \mathbb{E}^2(u) \cdot (\mathsf{Var}(e_1) + \mathsf{Var}(e_2))\big) \\
&= \frac{N \cdot q^2}{\Delta^2} \cdot \big(\mathsf{Var}(u) + \mathbb{E}^2(u)\big) \cdot \left(\mathsf{Var}(e_1) + \mathsf{Var}(e_2)\right) \\
&= \frac{N}{\Delta^2}\left(\frac{q^2-1}{12}\big(1 + kN\mathsf{Var}(S) + kN\mathbb{E}^2(S)\big) + \frac{kN}{4}\mathsf{Var}(S) + \frac{1}{4}\left(1 + kN\mathbb{E}(S)\right)^2\right)(\sigma_1^2 + \sigma_2^2)
\end{aligned}$$

Observe that e_i and u_j indicate the coefficients of E_i and U_j respectively. The factor N comes from the fact that they are polynomials.

(III) In this third part, we compute the variance of the error caused by the rounding. We consider that $\overline{B}, \overrightarrow{\overline{A}}, \overrightarrow{\overline{R}}, \overrightarrow{\overline{T}}$ are all sampled from $\frac{\mathcal{U}(\llbracket -\Delta/2, \Delta/2 \rrbracket)}{\Delta}$. We also consider that:

- $\vec{S}$ is composed by k elements of the form $S_i = S$, whose distribution are given in Table A.1.
- $\vec{S}_T$ is composed by k elements of the form $S_i^2 = S'$, whose distribution is studied in Section A.1.3. In particular, $S' = \sum_{\alpha=0}^{N-1} S'_\alpha X^\alpha = S'_{\mathsf{even}} + S'_{\mathsf{odd}}$ with $S'_{\mathsf{even}} = \sum_{i=0}^{N/2-1} S'_{2i}X^{2i}$ and $S'_{\mathsf{odd}} = \sum_{i=0}^{N/2-1} S'_{2i+1}X^{2i+1}$.
- $\vec{S}_R$ is composed by $\frac{k(k-1)}{2}$ elements of the form $S_i \cdot S_j = S''$, whose distribution is studied in Section A.1.3.

Then:

$$\begin{aligned}
\mathsf{Var}(III) &= \mathsf{Var}\left(\overline{B} - \overline{\vec{A}} \cdot \vec{S} + \overline{\vec{R}} \cdot \vec{S}_R + \overline{\vec{T}} \cdot \vec{S}_T\right) \\
&= \mathsf{Var}\left(\overline{B}\right) + \mathsf{Var}\left(\overline{\vec{A}} \cdot \vec{S}\right) + \mathsf{Var}\left(\overline{\vec{R}} \cdot \vec{S}_R\right) + \mathsf{Var}\left(\overline{\vec{T}} \cdot \vec{S}_T\right) \\
&= \frac{\Delta^2 - 1}{12\Delta^2} + k \cdot N \cdot \big(\mathsf{Var}(\overline{a}) \cdot \big(\mathsf{Var}(S) + \mathbb{E}^2(S)\big) + \mathbb{E}^2(\overline{a}) \cdot \mathsf{Var}(S)\big) + \\
&\quad + \frac{k(k-1)}{2} \cdot N \cdot \big(\mathsf{Var}(\overline{r}) \cdot \big(\mathsf{Var}(S'') + \mathbb{E}^2(S'')\big) + \mathbb{E}^2(\overline{r}) \cdot \mathsf{Var}(S'')\big) + \\
&\quad + k \cdot \left(\frac{N}{2} \cdot \mathsf{Var}(\overline{t}) \cdot \big(\mathsf{Var}(S'_{\mathsf{odd}}) + \mathsf{Var}(S'_{\mathsf{even}}) + 2 \cdot \mathbb{E}^2(S'_{\mathsf{mean}})\big) + \frac{N}{2} \cdot \mathbb{E}^2(\overline{t}) \cdot (\mathsf{Var}(S'_{\mathsf{odd}}) + \mathsf{Var}(S'_{\mathsf{even}}))\right) \\
&= \frac{\Delta^2 - 1}{12\Delta^2} + kN \cdot \left(\frac{\Delta^2 - 1}{12\Delta^2} \cdot \big(\mathsf{Var}(S) + \mathbb{E}^2(S)\big) + \frac{1}{4\Delta^2} \cdot \mathsf{Var}(S)\right) + \\
&\quad + \frac{k(k-1)N}{2} \cdot \left(\frac{\Delta^2 - 1}{12\Delta^2} \cdot \big(\mathsf{Var}(S'') + \mathbb{E}^2(S'')\big) + \frac{1}{4\Delta^2} \cdot \mathsf{Var}(S'')\right) + \\
&\quad + \frac{kN}{2} \cdot \left(\frac{\Delta^2 - 1}{12\Delta^2} \cdot \big(\mathsf{Var}(S'_{\mathsf{odd}}) + \mathsf{Var}(S'_{\mathsf{even}}) + 2 \cdot \mathbb{E}^2(S'_{\mathsf{mean}})\big) + \frac{1}{4\Delta^2} \cdot (\mathsf{Var}(S'_{\mathsf{odd}}) + \mathsf{Var}(S'_{\mathsf{even}}))\right) \\
&= \frac{\Delta^2 - 1}{12\Delta^2} + \frac{kN}{12\Delta^2} \cdot \big((\Delta^2 - 1) \cdot \big(\mathsf{Var}(S) + \mathbb{E}^2(S)\big) + 3 \cdot \mathsf{Var}(S)\big) + \\
&\quad + \frac{k(k-1)N}{24\Delta^2} \cdot \big((\Delta^2 - 1) \cdot \big(\mathsf{Var}(S'') + \mathbb{E}^2(S'')\big) + 3 \cdot \mathsf{Var}(S'')\big) + \\
&\quad + \frac{kN}{24\Delta^2} \cdot \big((\Delta^2 - 1) \cdot \big(\mathsf{Var}(S'_{\mathsf{odd}}) + \mathsf{Var}(S'_{\mathsf{even}}) + 2 \cdot \mathbb{E}^2(S'_{\mathsf{mean}})\big) + 3 \cdot (\mathsf{Var}(S'_{\mathsf{odd}}) + \mathsf{Var}(S'_{\mathsf{even}}))\big)
\end{aligned}$$

The formula is correct if $N \neq 1$. In fact, observe that the study for key S'' adapts for $N = 1$ but the key S' does not. In fact, S' is not a polynomial: there is just 1 term (not odd or even anymore). In case of S' with $N = 1$, we fix the formula as follows:

$$\text{If } N = 1 \begin{cases} \mathsf{Var}(S'_{\mathsf{odd}}) = 0 \\ \mathsf{Var}(S'_{\mathsf{even}}) = 2 \cdot \mathsf{Var}(s_i^2) \\ \mathbb{E}(S'_{\mathsf{mean}}) = \mathbb{E}(s_i^2) \end{cases}$$

The factor 2 in $\mathsf{Var}(S'_{\mathsf{even}})$ is given by the $N/2$ that took care of odd and even coefficients.

Finally:

$$\begin{aligned}
\mathsf{Var}(E) &= \mathsf{Var}(\mathsf{Error}(\vec{T}', \vec{R}', \vec{A}', B')) \\
&= \underbrace{\mathsf{Var}\left(\frac{\Delta_1}{\Delta} M_1 E_2 + \frac{\Delta_2}{\Delta} M_2 E_1 + \Delta^{-1} E_1 E_2\right)}_{(I)} + \\
&\quad + \underbrace{\mathsf{Var}\left(\frac{q}{\Delta}(E_1 U_2 + E_2 U_1)\right)}_{(II)} + \underbrace{\mathsf{Var}\left(\overline{B} - \overline{\vec{A}} \cdot \vec{S} + \overline{\vec{R}} \cdot \vec{S}_R + \overline{\vec{T}} \cdot \vec{S}_T\right)}_{(III)} \\
&= \underbrace{\tfrac{N}{\Delta^2}\left(\Delta_1^2 ||M_1||_\infty^2 \sigma_2^2 + \Delta_2^2 ||M_2||_\infty^2 \sigma_1^2 + \sigma_1^2 \sigma_2^2\right)}_{(I)} + \\
&\quad + \underbrace{\tfrac{N}{\Delta^2}\left(\tfrac{q^2-1}{12}\left(1 + kN\mathsf{Var}(S) + kN\mathbb{E}^2(S)\right) + \tfrac{kN}{4}\mathsf{Var}(S) + \tfrac{1}{4}(1 + kN\mathbb{E}(S))^2\right)(\sigma_1^2 + \sigma_2^2)}_{(II)} + \\
&\quad + \underbrace{\tfrac{\Delta^2-1}{12\cdot\Delta^2} + \tfrac{kN}{12\Delta^2}\cdot\left((\Delta^2-1)\cdot\left(\mathsf{Var}(S) + \mathbb{E}^2(S)\right) + 3\cdot\mathsf{Var}(S)\right) + \tfrac{k(k-1)N}{24\Delta^2}\cdot\left((\Delta^2-1)\cdot\left(\mathsf{Var}(S'') + \mathbb{E}^2(S'')\right) + 3\cdot\mathsf{Var}(S'')\right) +}_{(III)} \\
&\quad + \underbrace{\tfrac{kN}{24\Delta^2}\cdot\left((\Delta^2-1)\cdot\left(\mathsf{Var}(S'_{\mathsf{odd}}) + \mathsf{Var}(S'_{\mathsf{even}}) + 2\cdot\mathbb{E}^2(S'_{\mathsf{mean}})\right) + 3\cdot\left(\mathsf{Var}(S'_{\mathsf{odd}}) + \mathsf{Var}(S'_{\mathsf{even}})\right)\right)}_{(III)}
\end{aligned}$$

A.1.5 Bi-Distributed Error Polynomials

In this section, we analyse the case where the message is not a dense polynomial and the error polynomial has coefficients following two different distributions. In particular, we suppose that the message polynomial M contains $0 \leq \alpha \leq N$ *filled* coefficients, and their corresponding error terms following a Gaussian distribution $\mathcal{N}(0, \sigma_{\mathsf{fill}}^2)$, and there are $N - \alpha$ *empty* coefficients containing error from the distribution $\mathcal{N}(0, \sigma_{\mathsf{emp}}^2)$.

We consider two message polynomials M_1 and M_2. The first one contains the α_1 message coefficients $m_{1,1}, \cdots, m_{1,\alpha_1}$ and the second polynomial contains the α_2 message coefficients $m_{2,1}, \cdots, m_{2,\alpha_2}$.

We will make the noise analysis only for the coefficients in the resulting plaintext polynomial filled with single product of the form $m_{1,i} \cdot m_{2,j}$ for $1 \leq i \leq \alpha_1$ and $1 \leq j \leq \alpha_2$.

For instance, in a product like

$$\begin{aligned}
&(a_0 + a_1 X + a_3 X^3) \cdot (b_0 + b_1 X) = \\
&\quad = a_0 b_0 + (a_0 b_1 + a_1 b_0) X + a_1 b_1 X^2 + a_3 b_0 X^3 + a_3 b_1 X^4 \\
&\quad = c_0 + c_1 X + c_2 X^2 + c_3 X^3 + c_4 X^4
\end{aligned}$$

we are not going to analyse the noise in the c_1 coefficient for instance.

An example is the result of a LWE to GLWE key switching, where the constant term is the only one containing a message, and its error is larger than the error in the other coefficients.

In the error of the tensor product:

$$\mathsf{Error}(\vec{T}', \vec{R}', \vec{A}', B') = \underbrace{\frac{\Delta_1}{\Delta} M_1 E_2 + \frac{\Delta_2}{\Delta} M_2 E_1 + \Delta^{-1} E_1 E_2}_{(I)} + \\ + \underbrace{\frac{q}{\Delta}(E_1 U_2 + E_2 U_1)}_{(II)} + \underbrace{\left(\overline{B} - \overline{\vec{A}} \cdot \vec{S} + \overline{\vec{R}} \cdot \vec{S}_R + \overline{\vec{T}} \cdot \vec{S}_T\right)}_{(III)} \in \mathfrak{R}_q$$

the only difference happens in terms (I) and (II). Let us analyze them separately.

($\mathbf{I}_{\mathsf{fill}}$) The variance is:

$$\begin{aligned}
&\mathsf{Var}(\mathrm{I}_{\mathsf{fill}})\\
&= \mathsf{Var}\left(\frac{\Delta_1}{\Delta} M_1 E_2 + \frac{\Delta_2}{\Delta} M_2 E_1 + \Delta^{-1} E_1 E_2\right)\\
&= \mathsf{Var}\left(\frac{\Delta_1}{\Delta} M_1 E_2\right) + \mathsf{Var}\left(\frac{\Delta_2}{\Delta} M_2 E_1\right) + \mathsf{Var}(\Delta^{-1} E_1 E_2)\\
&= \frac{\Delta_1^2}{\Delta^2} ||M_1||_\infty^2 \left((\alpha_1 - 1) \cdot \sigma_{2,\mathsf{emp}}^2 + \sigma_{2,\mathsf{fill}}^2\right) + \frac{\Delta_2^2}{\Delta^2} ||M_2||_\infty^2 \left((\alpha_2 - 1) \cdot \sigma_{1,\mathsf{emp}}^2 + \sigma_{1,\mathsf{fill}}^2\right) +\\
&+ \frac{1}{\Delta^2}\left(\sigma_{1,\mathsf{fill}}^2 \sigma_{2,\mathsf{fill}}^2 + (\alpha_1 - 1)\sigma_{1,\mathsf{fill}}^2 \sigma_{2,\mathsf{emp}}^2 + (\alpha_2 - 1)\sigma_{1,\mathsf{emp}}^2 \sigma_{2,\mathsf{fill}}^2 + (N - \alpha_1 - \alpha_2 + 1)(\sigma_{1,\mathsf{emp}}^2 \sigma_{2,\mathsf{emp}}^2)\right)
\end{aligned}$$

($\mathbf{II}_{\mathsf{fill}}$) By now, we note the coefficients $S_{i,h}$ of S_i, simply by S. Then:

$$\begin{aligned}
&\mathsf{Var}(\mathrm{II}_{\mathsf{fill}}) =\\
&= \mathsf{Var}\left(\frac{q}{\Delta}(E_1 U_2 + E_2 U_1)\right)\\
&= \frac{q^2}{\Delta^2} \cdot \mathsf{Var}\left(E_1 U_2 + E_2 U_1\right)\\
&= \frac{q^2}{\Delta^2} \cdot (\mathsf{Var}(E_1 U_2) + \mathsf{Var}(E_2 U_1))\\
&= \frac{q^2}{\Delta^2} \cdot \left(\alpha_1 \mathsf{Var}(e_{1,\mathsf{fill}} u_2) + (N - \alpha_1)\mathsf{Var}(e_{1,\mathsf{emp}} u_2)\right) + \frac{q^2}{\Delta^2} \cdot \left(\alpha_2 \mathsf{Var}(e_{2,\mathsf{fill}} u_1) + (N - \alpha_2)\mathsf{Var}(e_{2,\mathsf{emp}} u_1)\right)\\
&= \frac{q^2}{\Delta^2} \cdot \left(\alpha_1 \mathsf{Var}(e_{1,\mathsf{fill}} u) + (N - \alpha_1)\mathsf{Var}(e_{1,\mathsf{emp}} u)\right) + \frac{q^2}{\Delta^2} \cdot \left(\alpha_2 \mathsf{Var}(e_{2,\mathsf{fill}} u) + (N - \alpha_2)\mathsf{Var}(e_{2,\mathsf{emp}} u)\right)\\
&= \frac{q^2}{\Delta^2} \cdot \left(\mathsf{Var}(u) + \mathbb{E}^2(u)\right) \cdot \left(\alpha_1 \mathsf{Var}(e_{1,\mathsf{fill}}) + (N - \alpha_1)\mathsf{Var}(e_{1,\mathsf{emp}}) + \alpha_2 \mathsf{Var}(e_{2,\mathsf{fill}}) + (N - \alpha_2)\mathsf{Var}(e_{2,\mathsf{emp}})\right)\\
&= \frac{N}{\Delta^2} \cdot \left(\frac{q^2 - 1}{12}\left(1 + kN\mathsf{Var}(S) + kN\mathbb{E}^2(S)\right) + \frac{kN}{4}\mathsf{Var}(S) + \frac{1}{4}\left(1 + kN\mathbb{E}(S)\right)^2\right) \cdot\\
&\quad \cdot \left(\alpha_1 \sigma_{1,\mathsf{fill}} + (N - \alpha_1)\sigma_{1,\mathsf{emp}} + \alpha_2 \sigma_{2,\mathsf{fill}} + (N - \alpha_2)\sigma_{2,\mathsf{emp}}\right)
\end{aligned}$$

Observe that e_i and u_j indicate the coefficients of E_i and U_j respectively. The factor N comes from the fact that they are polynomials.

A.1.6 Relinearization

The last step (relinearization) is to compute:

$$\mathsf{Relin}(\vec{T'}, \vec{R'}, \vec{A'}, B') = (A'_1, \cdots, A'_k, B') + \sum_{i=1}^{k} \left\langle \overline{\mathsf{CT}}_{i,i}, \mathsf{dec}^{(\beta,\ell)}\left(T'_i\right) \right\rangle + \sum_{1\leq i\leq k}^{1\leq j<i} \left\langle \overline{\mathsf{CT}}_{i,j} \cdot \mathsf{dec}^{(\beta,\ell)}\left(R'_{i,j}\right) \right\rangle$$

where:

$$\mathsf{RLK} = \left\{ \overline{\mathsf{CT}}_{i,j} = \mathsf{GLev}_{\vec{S}}^{(\beta,\ell)}\left(S_i \cdot S_j\right) = \left\{ \mathsf{RLK}_h^{(i,j)} \right\}_{1\leq h\leq \ell} \right\}_{1\leq i\leq k}^{1\leq j\leq i}$$

is the **realinearization key**, so that each component $\mathsf{RLK}_h^{(i,j)}$, with $i \in [1,k], j \in [1,i], h \in [1,\ell]$ is defined as:

$$\mathsf{RLK}_h^{(i,j)} = \mathsf{RLWE}_{\vec{S}}\left(S_i \cdot S_j \cdot \frac{q}{\beta^h} \right) = \left(\vec{A}_{\mathsf{RLK}_h^{(i,j)}}, B_{\mathsf{RLK}_h^{(i,j)}} \right)$$

with

$$\begin{cases} B_{\mathsf{RLK}_h^{(i,j)}} = \sum_{\alpha=1}^{k} A_{\alpha,\mathsf{RLK}_h^{(i,j)}} \cdot S_\alpha + E_{\mathsf{RLK}_h^{(i,j)}} - S_i \cdot S_j \cdot \frac{q}{\beta^j} \mod q \\ A_{\alpha,\mathsf{RLK}_h^{(i,j)}} \text{ coefficients in } \mathcal{U}(\llbracket -\frac{q}{2}, \frac{q}{2} \rrbracket) \\ E_{\mathsf{RLK}_h^{(i,j)}} \text{ coefficients in } \mathcal{N}(0, \sigma^2_{\mathsf{RLK}}) \end{cases}$$

To ease the notations in this section, we will note $\vec{T'}, \vec{R'}, \vec{A'}, B'$ as $\vec{T}, \vec{R}, \vec{A}, B$ respectively.

Decomposition We start by decomposing $\vec{T}$ and $\vec{R}$ w.r.t the basis β and the number of level ℓ starting from the MSB. By using the previous notation, we have:

- $\vec{T} = \{T_i\}_{i\in[k]} = \{T'_i + \overline{T_i}\}_{i\in[k]}$, with $T'_i = \sum_{p=0}^{N-1} T'_{i,p} X^p$, and each $T'_{i,p}$ is the closest multiple of $\frac{q}{\beta^\ell}$ in $\mathbb{Z}_q$. Then, we write each $T'_{i,p}$ as $\sum_{h=1}^{\ell} T'_{i,p,h} \frac{q}{\beta^h}$, where each $T'_{i,p,h} \in \llbracket -\frac{\beta}{2}, \frac{\beta}{2} \llbracket$.
  The $\overline{T_i} = \sum_{p=0}^{N-1} \overline{T_{i,p}} \cdot X^p$ term represents the rounding error, so that each coefficient of $\overline{T_{i,p}} \sim \mathcal{U}(\llbracket -\frac{q}{2\beta^\ell}, \frac{q}{2\beta^\ell} \rrbracket)$.

- $\vec{R} = \{R_{i,j}\}_{i\in[k],j\in[i-1]} = \{R'_{i,j} + \overline{R_{i,j}}\}_{i\in[k],j\in[i-1]}$, with $R'_{i,j} = \sum_{p=0}^{N-1} R'_{i,j,p} X^p$, and each $R'_{i,j,p}$ is the closest multiple of $\frac{q}{\beta^\ell}$ in $\mathbb{Z}_q$. Then, we write each $R'_{i,j,p}$ as $\sum_{h=1}^{\ell} R'_{i,j,p,h} \frac{q}{\beta^h}$, where each $R'_{i,j,p,h} \in \llbracket -\frac{\beta}{2}, \frac{\beta}{2} \llbracket$.
  The $\overline{R_{i,j}} = \sum_{p=0}^{N-1} \overline{R_{i,j,p}} \cdot X^p$ term represents the rounding error, so that each coefficient of $\overline{R_{i,j,p}} \sim \mathcal{U}(\llbracket -\frac{q}{2\beta^\ell}, \frac{q}{2\beta^\ell} \rrbracket)$.

We denote by $T'_{i,h}$ the polynomial $\sum_{p=0}^{N-1} T'_{i,p,h} X^p$ and by $R'_{i,j,h}$ the polynomial $\sum_{p=0}^{N-1} R'_{i,j,p,h} X^p$.

Relinearization

$$\begin{aligned}
\mathsf{CT} &= \mathsf{Relin}(\vec{T}, \vec{R}, \vec{A}, B) \\
&= \left(\vec{A}, B\right) + \sum_{i=1}^{k} \left\langle \overline{\mathsf{CT}}_{i,i}, \mathsf{dec}^{(\beta,\ell)}\left(T_i\right)\right\rangle + \sum_{1\le i\le k}^{1\le j<i} \left\langle \overline{\mathsf{CT}}_{i,j} \cdot \mathsf{dec}^{(\beta,\ell)}\left(R_{i,j}\right)\right\rangle \\
&= (\vec{A}, B) + \sum_{i=1}^{k}\sum_{h=1}^{\ell} \mathsf{RLK}_h^{(i,i)} \cdot T'_{i,h} + \sum_{i=1}^{k}\sum_{j=1}^{i-1}\sum_{h=1}^{\ell} \mathsf{RLK}_h^{(i,j)} \cdot R'_{i,j,h} \\
&= (\vec{A}, B) + \sum_{i=1}^{k}\sum_{h=1}^{\ell} \left(\vec{A}_{\mathsf{RLK}_h^{(i,i)}}, B_{\mathsf{RLK}_h^{(i,i)}}\right) \cdot T'_{i,h} + \sum_{i=1}^{k}\sum_{j=1}^{i-1}\sum_{h=1}^{\ell} \left(\vec{A}_{\mathsf{RLK}_h^{(i,j)}}, B_{\mathsf{RLK}_h^{(i,j)}}\right) \cdot R'_{i,j,h} \\
&= \left(\vec{A} + \textstyle\sum_{i=1}^{k}\sum_{h=1}^{\ell}\left(\vec{A}_{\mathsf{RLK}_h^{(i,i)}} \cdot T'_{i,h} + \sum_{j=1}^{i-1} \vec{A}_{\mathsf{RLK}_h^{(i,j)}} \cdot R'_{i,j,h}\right), B + \sum_{i=1}^{k}\sum_{h=1}^{\ell}\left(B_{\mathsf{RLK}_h^{(i,i)}} \cdot T'_{i,h} + \sum_{j=1}^{i-1} B_{\mathsf{RLK}_h^{(i,j)}} \cdot R'_{i,j,h}\right)\right) \\
&= (\vec{A}_{res}, B_{res}) \quad \in \mathfrak{R}_q^2
\end{aligned}$$

The computation of the phase gives:

$$\begin{aligned}
\mathsf{CT} \cdot (-\vec{S}, 1) &= B_{res} - \vec{A}_{res} \cdot \vec{S} \\
&= B + \textstyle\sum_{i=1}^{k}\sum_{h=1}^{\ell}\left(B_{\mathsf{RLK}_h^{(i,i)}} \cdot T'_{i,h} + \sum_{j=1}^{i-1} B_{\mathsf{RLK}_h^{(i,j)}} \cdot R'_{i,j,h}\right) - \vec{A}\cdot\vec{S} - \sum_{i=1}^{k}\sum_{h=1}^{\ell}\left(\vec{A}_{\mathsf{RLK}_h^{(i,i)}} \cdot T'_{i,h} + \sum_{j=1}^{i-1} \vec{A}_{\mathsf{RLK}_h^{(i,j)}} \cdot R'_{i,j,h}\right)\cdot\vec{S} \\
&= B - \vec{A}\cdot\vec{S} + \textstyle\sum_{i=1}^{k}\sum_{h=1}^{\ell}\left(\vec{A}_{\mathsf{RLK}_h^{(i,i)}} \cdot\vec{S} + E_{\mathsf{RLK}_h^{(i,i)}} + S_i^2 \cdot \frac{q}{\beta^h}\right)\cdot T'_{i,h} + \\
&\quad + \textstyle\sum_{i=1}^{k}\sum_{h=1}^{\ell}\sum_{j=1}^{i-1}\left(\vec{A}_{\mathsf{RLK}_h^{(i,j)}} \cdot\vec{S} + E_{\mathsf{RLK}_h^{(i,j)}} + S_i \cdot S_j \cdot \frac{q}{\beta^h}\right)\cdot R'_{i,j,h} - \sum_{i=1}^{k}\sum_{h=1}^{\ell}\left(\vec{A}_{\mathsf{RLK}_h^{(i,i)}} \cdot T'_{i,h} + \sum_{j=1}^{i-1} \vec{A}_{\mathsf{RLK}_h^{(i,j)}} \cdot R'_{i,j,h}\right)\cdot\vec{S} \\
&= B - \vec{A}\cdot\vec{S} + \textstyle\sum_{i=1}^{k}\sum_{h=1}^{\ell}\left(E_{\mathsf{RLK}_h^{(i,i)}} + S_i^2 \cdot \frac{q}{\beta^h}\right)\cdot T'_{i,h} + \sum_{i=1}^{k}\sum_{h=1}^{\ell}\sum_{j=1}^{i-1}\left(E_{\mathsf{RLK}_h^{(i,j)}} + S_i \cdot S_j \cdot \frac{q}{\beta^h}\right)\cdot R'_{i,j,h} \\
&= B - \vec{A}\cdot\vec{S} + \textstyle\sum_{i=1}^{k}\sum_{h=1}^{\ell} E_{\mathsf{RLK}_h^{(i,i)}} \cdot T'_{i,h} + \sum_{i=1}^{k}\sum_{h=1}^{\ell} S_i^2 \cdot \frac{q}{\beta^h} \cdot T'_{i,h} + \\
&\quad + \textstyle\sum_{i=1}^{k}\sum_{h=1}^{\ell}\sum_{j=1}^{i-1} E_{\mathsf{RLK}_h^{(i,j)}} \cdot R'_{i,j,h} + \sum_{i=1}^{k}\sum_{h=1}^{\ell}\sum_{j=1}^{i-1} S_i \cdot S_j \cdot \frac{q}{\beta^h} \cdot R'_{i,j,h} \\
&= B - \vec{A}\cdot\vec{S} + \textstyle\sum_{i=1}^{k}\sum_{h=1}^{\ell} E_{\mathsf{RLK}_h^{(i,i)}} \cdot T'_{i,h} + \sum_{i=1}^{k} S_i^2 \cdot T'_i + \sum_{i=1}^{k}\sum_{h=1}^{\ell}\sum_{j=1}^{i-1} E_{\mathsf{RLK}_h^{(i,j)}} \cdot R'_{i,j,h} + \sum_{i=1}^{k}\sum_{j=1}^{i-1} S_i \cdot S_j \cdot R'_{i,j} \\
&= B - \vec{A}\cdot\vec{S} + \textstyle\sum_{i=1}^{k}\sum_{h=1}^{\ell}\left(E_{\mathsf{RLK}_h^{(i,i)}} \cdot T'_{i,h} + \sum_{j=1}^{i-1} E_{\mathsf{RLK}_h^{(i,j)}} \cdot R'_{i,j,h}\right) + \sum_{i=1}^{k}\left(S_i^2 \cdot (T_i - \overline{T_i}) + \sum_{j=1}^{i-1} S_i \cdot S_j \cdot (R_{i,j} - \overline{R_{i,j}})\right) \\
&= B - \vec{A}\cdot\vec{S} + \vec{R}\cdot\vec{S}_R + \vec{T}\cdot\vec{S}_T + \textstyle\sum_{i=1}^{k}\sum_{h=1}^{\ell}\left(E_{\mathsf{RLK}_h^{(i,i)}} \cdot T'_{i,h} + \sum_{j=1}^{i-1} E_{\mathsf{RLK}_h^{(i,j)}} \cdot R'_{i,j,h}\right) - \overline{\vec{R}}\cdot\vec{S}_R - \overline{\vec{T}}\cdot\vec{S}_T \\
&= P_{res} + \underbrace{E + \sum_{i=1}^{k}\sum_{h=1}^{\ell}\left(E_{\mathsf{RLK}_h^{(i,i)}} \cdot T'_{i,h} + \sum_{j=1}^{i-1} E_{\mathsf{RLK}_h^{(i,j)}} \cdot R'_{i,j,h}\right) - \overline{\vec{R}} \cdot \vec{S}_R - \overline{\vec{T}} \cdot \vec{S}_T}_{Error}
\end{aligned}$$

The error term is then:

$$Error = \underbrace{E}_{(I)} + \underbrace{\sum_{i=1}^{k}\sum_{h=1}^{\ell}\left(E_{\mathsf{RLK}_h^{(i,i)}} \cdot T'_{i,h} + \sum_{j=1}^{i-1} E_{\mathsf{RLK}_h^{(i,j)}} \cdot R'_{i,j,h}\right)}_{(II)} - \underbrace{\overline{\vec{R}} \cdot \vec{S}_R - \overline{\vec{T}} \cdot \vec{S}_T}_{(III)}$$

For each term, we compute their variance.

The variance of (I) is the error obtained from the tensor product computation.

The variance of the second term (II) is

$$\begin{aligned}
\mathsf{Var}(II) &= \mathsf{Var}\left(\sum_{i=1}^{k}\sum_{h=1}^{\ell}\left(E_{\mathsf{RLK}_h^{(i,i)}} \cdot T'_{i,h} + \sum_{j=1}^{i-1} E_{\mathsf{RLK}_h^{(i,j)}} \cdot R'_{i,j,h}\right)\right) \\
&= \mathsf{Var}\left(\sum_{i=1}^{k}\sum_{h=1}^{\ell} E_{\mathsf{RLK}_h^{(i,i)}} \cdot T'_{i,h}\right) + \mathsf{Var}\left(\sum_{i=1}^{k}\sum_{h=1}^{\ell}\sum_{j=1}^{i-1} E_{\mathsf{RLK}_h^{(i,j)}} \cdot R'_{i,j,h}\right) \\
&= k\ell \mathsf{Var}\left(E_{\mathsf{RLK}_h^{(i,i)}} \cdot T'_{i,h}\right) + \tfrac{k(k-1)\ell}{2}\mathsf{Var}\left(E_{\mathsf{RLK}_h^{(i,j)}} \cdot R'_{i,j,h}\right) \\
&= k\ell N \mathsf{Var}\left(e_{\mathsf{RLK}_h^{(i,i)}} \cdot t'_{i,h}\right) + \tfrac{k(k-1)\ell N}{2}\mathsf{Var}\left(e_{\mathsf{RLK}_h^{(i,j)}} \cdot r'_{i,j,h}\right) \\
&= k\ell N\left(\mathsf{Var}(e_{\mathsf{RLK}_h^{(i,i)}}) \cdot \mathsf{Var}(t'_{i,h}) + \mathsf{Var}(e_{\mathsf{RLK}_h^{(i,i)}}) \cdot \mathbb{E}^2(t'_{i,h}) + \mathbb{E}^2(e_{\mathsf{RLK}_h^{(i,i)}}) \cdot \mathsf{Var}(t'_{i,h})\right) + \\
&\quad + \tfrac{k(k-1)\ell N}{2}\left(\mathsf{Var}(e_{\mathsf{RLK}_h^{(i,j)}}) \cdot \mathsf{Var}(r'_{i,j,h}) + \mathsf{Var}(e_{\mathsf{RLK}_h^{(i,j)}}) \cdot \mathbb{E}^2(r'_{i,j,h}) + \mathbb{E}^2(e_{\mathsf{RLK}_h^{(i,j)}}) \cdot \mathsf{Var}(r'_{i,j,h})\right) \\
&= k\ell N\left(\sigma^2_{\mathsf{RLK}} \cdot \mathsf{Var}(t'_{i,h}) + \sigma^2_{\mathsf{RLK}} \cdot \mathbb{E}^2(t'_{i,h})\right) + \tfrac{k(k-1)\ell N}{2}\left(\sigma^2_{\mathsf{RLK}} \cdot \mathsf{Var}(r'_{i,j,h}) + \sigma^2_{\mathsf{RLK}} \cdot \mathbb{E}^2(r'_{i,j,h})\right) \\
&= k\ell N \sigma^2_{\mathsf{RLK}} \cdot \left(\tfrac{\beta^2-1}{12} + \tfrac{1}{4}\right) + \tfrac{k(k-1)\ell N}{2}\sigma^2_{\mathsf{RLK}} \cdot \left(\tfrac{\beta^2-1}{12} + \tfrac{1}{4}\right) \\
&= k\ell N \sigma^2_{\mathsf{RLK}} \cdot \frac{(k+1)}{2} \cdot \frac{\beta^2+2}{12}\ .
\end{aligned}$$

The variance of the third term (III) is

$$\begin{aligned}
&\mathsf{Var}(\overline{\vec{R}} \cdot \vec{S}_R - \overline{\vec{T}} \cdot \vec{S}_T) \\
&\quad = \mathsf{Var}(\overline{\vec{R}} \cdot \vec{S}_R) + \mathsf{Var}(\overline{\vec{T}} \cdot \vec{S}_T) \\
&\quad = \textstyle\sum_{i=1}^{k}\sum_{j=1}^{i-1} \mathsf{Var}(\overline{R_{i,j}} \cdot S_i S_j) + \sum_{i=1}^{k} \mathsf{Var}(\overline{T_i} \cdot S_i^2) \\
&\quad = \tfrac{k(k-1)}{2} \cdot \mathsf{Var}(\overline{R_{i,j}} \cdot S_i S_j) + k \cdot \mathsf{Var}(\overline{T_i} \cdot S_i^2) \\
&\quad = \tfrac{k(k-1)N}{2} \cdot \left(\mathsf{Var}(\overline{r_{i,j}}) \cdot \mathsf{Var}(S'') + \mathbb{E}^2(\overline{r_{i,j}}) \cdot \mathsf{Var}(S'') + \mathsf{Var}(\overline{r_{i,j}}) \cdot \mathbb{E}^2(S''_{\mathsf{mean}})\right) + \\
&\quad\quad + \tfrac{kN}{2} \cdot \left(\mathsf{Var}(S'_{\mathsf{odd}}) + \mathsf{Var}(S'_{\mathsf{even}})\right) \cdot \left(\mathsf{Var}(\overline{t_i}) + \mathbb{E}^2(\overline{t_i})\right) + kN \cdot \mathbb{E}^2(S'_{\mathsf{mean}}) \cdot \mathsf{Var}(\overline{t_i}) \\
&\quad = \tfrac{k(k-1)N}{2} \cdot \left(\left(\tfrac{q^2}{12\beta^{2\ell}} - \tfrac{1}{12}\right) \cdot \mathsf{Var}(S'') + \tfrac{1}{4} \cdot \mathsf{Var}(S'') + \left(\tfrac{q^2}{12\beta^{2\ell}} - \tfrac{1}{12}\right) \cdot \mathbb{E}^2(S''_{\mathsf{mean}})\right) +
\end{aligned}$$

$$
\begin{aligned}
&+\tfrac{kN}{2}\cdot\big(\mathsf{Var}(S'_{\mathsf{odd}})+\mathsf{Var}(S'_{\mathsf{even}})\big)\cdot\left(\left(\tfrac{q^2}{12\beta^{2\ell}}-\tfrac{1}{12}\right)+\tfrac{1}{4}\right)+kN\cdot\mathbb{E}^2(S'_{\mathsf{mean}})\cdot\left(\tfrac{q^2}{12\beta^{2\ell}}-\tfrac{1}{12}\right)\\
=\;&\tfrac{kN}{2}\cdot(k-1)\left(\tfrac{q^2}{12\beta^{2\ell}}-\tfrac{1}{12}\right)\cdot(\mathsf{Var}(S'')+\mathbb{E}^2(S''_{\mathsf{mean}}))+\tfrac{kN}{8}\cdot(k-1)\cdot\mathsf{Var}(S'')+\\
&+\tfrac{kN}{2}\cdot\big(\mathsf{Var}(S'_{\mathsf{odd}})+\mathsf{Var}(S'_{\mathsf{even}})+2\mathbb{E}^2(S'_{\mathsf{mean}})\big)\cdot\left(\tfrac{q^2}{12\beta^{2\ell}}-\tfrac{1}{12}\right)+\tfrac{kN}{8}\cdot\big(\mathsf{Var}(S'_{\mathsf{odd}})+\mathsf{Var}(S'_{\mathsf{even}})\big)\\
=\;&\tfrac{kN}{2}\left(\tfrac{q^2}{12\beta^{2\ell}}-\tfrac{1}{12}\right)\big((k-1)\cdot(\mathsf{Var}(S'')+\mathbb{E}^2(S''_{\mathsf{mean}}))+\mathsf{Var}(S'_{\mathsf{odd}})+\mathsf{Var}(S'_{\mathsf{even}})+2\mathbb{E}^2(S'_{\mathsf{mean}})\big)+\\
&+\tfrac{kN}{8}\cdot\big((k-1)\cdot\mathsf{Var}(S'')+\mathsf{Var}(S'_{\mathsf{odd}})+\mathsf{Var}(S'_{\mathsf{even}})\big)\;.
\end{aligned}
$$

Finally,

$$
\begin{aligned}
&\mathsf{Var}(\mathsf{Mult}) = \mathsf{Var}\,(E) + k\ell N\sigma^2_{\mathsf{RLK}}\cdot\frac{(k+1)}{2}\cdot\frac{\beta^2+2}{12}+\\
&+\frac{kN}{2}\left(\frac{q^2}{12\beta^{2\ell}}-\frac{1}{12}\right)\big((k-1)\cdot(\mathsf{Var}(S'')+\mathbb{E}^2(S''_{\mathsf{mean}}))+\mathsf{Var}(S'_{\mathsf{odd}})+\mathsf{Var}(S'_{\mathsf{even}})+2\mathbb{E}^2(S'_{\mathsf{mean}})\big)+\\
&+\frac{kN}{8}\cdot((k-1)\cdot\mathsf{Var}(S'')+\mathsf{Var}(S'_{\mathsf{odd}})+\mathsf{Var}(S'_{\mathsf{even}}))\;.
\end{aligned}
$$

A.2 Noise Analysis of the Generalized PBS

In this section, we provide a detailed proof of Theorem 22.

Proof 44 *We consider the input LWE ciphertext $(a_i, \cdots, a_n, b)$ encrypted under the secret key $\vec{s} = (s_1, \cdots, s_n)$ so we have $b = \sum_{i=1}^{n} a_i \cdot s_i + m + e$ with a message m and an error $e \in \chi_\sigma$. We want to modulus switch this ciphertext and compute $a'_i \leftarrow \left[\left\lfloor \frac{a_i \cdot 2N\cdot 2^{\varkappa-\vartheta}}{q} \right\rceil \cdot 2^{\vartheta}\right]_{2N}$, for $1 \leq i \leq n+1$.*

Let $w = 2N \cdot 2^{-\vartheta}$ and $q' = q \cdot 2^{-\varkappa}$. We note $a''_i = \left\lfloor \frac{w}{q'} a_i \right\rceil = \frac{w}{q'}a_i + \overline{a_i}$, then we have $a''_i \in \mathcal{U}(\llbracket \frac{-w}{2}, \frac{w}{2} \rrbracket)$ and $\overline{a_i} \in \frac{w}{q'}\mathcal{U}(\llbracket \frac{-q'}{2w}, \frac{q'}{2w} \rrbracket)$.

It means that $\mathsf{Var}(a''_i) = \frac{w^2-1}{12}$ and $\mathbb{E}(a''_i) = \frac{-1}{2}$, and that $\mathsf{Var}(\overline{a_i}) = \frac{1}{12} - \frac{w^2}{12q'^2}$ and $\mathbb{E}(\overline{a_i}) = \frac{-w}{2q'}$.

We decrypt:

$$
\begin{aligned}
&\mathsf{Decrypt}\left((a''_1, \cdots, a''_n, b'' = a''_{n+1}), \mathsf{SK}\right) =\\
&= b'' - \sum_{i=1}^{n} a''_i \cdot s_i = \frac{w}{q'}b + \bar{b} - \sum_{i=1}^{n}(\frac{w}{q'}a_i + \overline{a_i}) \cdot s_i\\
&= \frac{w}{q'}\left(b - \sum_{i=1}^{n} a_i \cdot s_i\right) + \bar{b} - \sum_{i=1}^{n} \overline{a_i} \cdot s_i\\
&= \frac{w}{q'}m + \frac{w}{q'}e + \bar{b} - \sum_{i=1}^{n} \overline{a_i} \cdot s_i
\end{aligned}
$$

We can now study the error:

$$
\begin{aligned}
&\mathsf{Var}(E_{\mathsf{res}}) = \\
&= \mathsf{Var}\left(\frac{w}{q'}e + \bar{b} - \sum_{i=1}^{n} \overline{a_i} \cdot s_i\right) \\
&= \frac{w^2\sigma_{\mathsf{in}}^2}{q'^2} + \mathsf{Var}(\bar{b}) + n \cdot \mathsf{Var}(\overline{a_i}) \cdot (\mathsf{Var}(s_i) + \mathbb{E}^2(s_i)) + n \cdot \mathbb{E}^2(\overline{a_i}) \cdot \mathsf{Var}(s_i) \\
&= \frac{w^2\sigma_{\mathsf{in}}^2}{q'^2} + \frac{1}{12} - \frac{w^2}{12q'^2} + n \cdot \left(\frac{1}{12} - \frac{w^2}{12q'^2}\right) \cdot \frac{1}{2} + n \cdot \frac{w'^2}{4q'^2} \cdot \frac{1}{4} \\
&= \frac{w^2\sigma_{\mathsf{in}}^2}{q'^2} + \frac{1}{12} - \frac{w^2}{12q'^2} + \frac{n}{24} + \frac{nw^2}{48q'^2}
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{E}(E_{\mathsf{res}}) &= \mathbb{E}\left(\frac{w}{q'}e + \bar{b} - \sum_{i=1}^{n} \overline{a_i} \cdot s_i\right) = \cancel{\mathbb{E}\left(\frac{w}{q'}e\right)} + \mathbb{E}(\bar{b}) - \sum_{i=1}^{n} \mathbb{E}(\overline{a_i} \cdot s_i) \\
&= \frac{-w}{2q'} - \sum_{i=1}^{n} \frac{-w}{2q'} \cdot \mathbb{E}(s_i) = \frac{w}{2q'} \cdot \left(\frac{n}{2} - 1\right)
\end{aligned}
$$

In order to have correctness of the modulus switching with probability $P = \mathsf{erf}\left(\frac{\Gamma}{\sqrt{2}}\right)$, *the following condition must be satisfied:*

$$
\Gamma \cdot \sqrt{\mathsf{Var}(E_{\mathsf{res}})} = \Gamma \cdot \sqrt{\frac{w^2\sigma_{\mathsf{in}}^2}{q'^2} + \frac{1}{12} - \frac{w^2}{12q'^2} + \frac{n}{24} + \frac{nw^2}{48q'^2}} < \frac{w\Delta_{\mathsf{in}}}{2q'}
$$

which implies that:

$$
\sigma_{\mathsf{in}}^2 < \frac{\Delta_{\mathsf{in}}^2}{4\Gamma^2} - \frac{q'^2}{12w^2} + \frac{1}{12} - \frac{nq'^2}{24w^2} - \frac{n}{48}.
$$

The steps of blind rotation and sample extraction are the same as in TFHE [Chi+20a] (see Theorem 14 and Algorithm 10). We report the proof that estimates the noise after the blind rotation (our analysis is slightly different from the analysis done in TFHE [Chi+20a]): the sample extraction step does not add any noise. In order to do the noise analysis for the blind rotation, we need to analyze the noise produced by the ***external product****.*

Let the GLWE *secret key be* $\vec{S} = (S_1, \ldots, S_k) \in \mathfrak{R}^k$ *(in the algorithm it is notes* $\vec{S'}$ *but we use the* $\vec{S}$ *notation to make the proof more readable), such that each polynomial key* S_i *has coefficients sampled from a uniform binary, uniform ternary or Gaussian distribution with* $\sigma = 3.2$*. The external product* $\boxdot$: GLWE $\times$ GGSW $\to$ GLWE *takes in* ***input****:*

- *A* GLWE *ciphertext*

$$\vec{c} = \mathsf{GLWE}_{\vec{S}}(\mu) = (A_1, \ldots, A_k, B = A_{k+1}) \in \mathfrak{R}_q^{(k+1)}$$

such that the coefficients of the polynomials A_α are sampled from $\mathcal{U}(\llbracket -\frac{q}{2}, \frac{q}{2} \rrbracket)$ and $B = \sum_{\alpha=1}^{k} A_\alpha \cdot S_\alpha + M_1 + E_1$, where $E_1 \in \mathfrak{R}_q$ is such that each coefficient is sampled from $\mathcal{N}(0, \sigma_1^2)$.

- *A* GGSW *ciphertext*

$$\vec{C} = \mathsf{GGSW}_{\vec{S}}(m) = \vec{Z} + M_2 \cdot \vec{G} \in \mathfrak{R}_q^{(k+1)\ell \times (k+1)}$$

where $\vec{G} = \mathrm{Id} \otimes \vec{g}$ is the gadget matrix, with $\vec{g}^\top = (\frac{q}{\mathfrak{B}}, \ldots, \frac{q}{\mathfrak{B}^\ell})$. Each line of the GGSW *ciphertext is of the form*

$$\vec{C}^{(i,j)} = (A_1^{(i,j)}, \ldots, A_k^{(i,j)}, B^{(i,j)}) = (\vec{A}^{(i,j)}, B^{(i,j)}) \in \mathfrak{R}_q^{(k+1)}$$

with $i \in \{1, \ldots, k+1\}$ and $j \in \{1, \ldots, \ell\}$, such that the coefficients of the polynomials $A_\alpha^{(i,j)}$ are sampled from $\mathcal{U}(\llbracket -\frac{q}{2}, \frac{q}{2} \rrbracket)$ and $B^{(i,j)} = \sum_{\alpha=1}^{k} A_\alpha^{(i,j)} \cdot S_\alpha + M_2^{(i,j)} + E_2^{(i,j)}$, where $E_2^{(i,j)} \in \mathbb{Z}_q[X]/(X^N+1)$ is such that each coefficient is sampled from $\mathcal{N}(0, \sigma_2^2)$ and $M_2^{(i,j)} = M_2 \frac{q}{\mathfrak{B}^j}(-S_i)$ (with $S_{k+1} = -1$).

*The **output** of the external product is:*

- *A* GLWE *ciphertext*

$$\vec{c}^{out} = \mathsf{GLWE}_{\vec{S}}(M_1 \cdot M_2) = (A_1^{out}, \ldots, A_k^{out}, B^{out}) \in \mathfrak{R}_q^{(k+1)}$$

*with error $E \in \mathfrak{R}_q$ such that each coefficient is sampled from $\mathcal{N}(0, \sigma^2)$. **We want to estimate** σ.*

Observe that the output is computed as:

$$\vec{c}^{out} = \underbrace{\vec{G}^{-1}(\vec{c})}_{\mathfrak{R}^{(k+1)\ell}} \cdot \vec{C} = \vec{u} \cdot \vec{C}$$

where the operation $\vec{G}^{-1}$ is the decomposition with respect to the gadget matrix.
The product consists in the following steps:

1. *Start by rounding each A_α ($\alpha = 1, \ldots, k+1$) at the $\ell \log_2(\mathfrak{B})$ bit by A'_α, such that the coefficients of $\overline{A}_\alpha = A_\alpha - A'_\alpha$ come from $\mathcal{U}(\llbracket -\frac{q}{2\mathfrak{B}^\ell}, \frac{q}{2\mathfrak{B}^\ell} \rrbracket)$.*

2. *Decompose each* $A'_\alpha = \sum_{j=1}^{\ell} A'_{\alpha,j} \cdot \frac{q}{\mathfrak{B}^{-l}}$ *with* $A'_{\alpha,j} \in \mathfrak{R}$ *with coefficients from* $\mathcal{U}(\llbracket -\frac{\mathfrak{B}}{2}, \frac{\mathfrak{B}}{2} \rrbracket)$.

3. *Return* $\vec{u} = (A'_{1,1}, \ldots, A'_{1,\ell}, \ldots, A'_{k,1}, \ldots, A'_{k,\ell}, B'_1 = A'_{k+1,1}, \ldots, B'_\ell = A'_{k+1,\ell})$.

Observe that:

$$\begin{aligned}\vec{c}^{out} &= \vec{G}^{-1}(\vec{c}) \cdot \vec{C} = \vec{u} \cdot \vec{C} \\ &= \sum_{i=1}^{k}\sum_{j=1}^{\ell} A'_{i,j} \cdot \vec{C}^{(i,j)} + \sum_{j=1}^{\ell} B'_j \cdot \vec{C}^{(k+1,j)}\end{aligned}$$

Let us write down the phase:

$$\begin{aligned}
&\left(\sum_{i=1}^{k}\sum_{j=1}^{\ell} A'_{i,j} \cdot \vec{C}^{(i,j)} + \sum_{j=1}^{\ell} B'_j \cdot \vec{C}^{(k+1,j)}\right) \cdot \left(-\vec{S}, 1\right) \\
&= \left(\sum_{i=1}^{k}\sum_{j=1}^{\ell} A'_{i,j} \cdot (\vec{A}^{(i,j)}, B^{(i,j)}) + \sum_{j=1}^{\ell} B'_j \cdot (\vec{A}^{(k+1,j)}, B^{(k+1,j)})\right) \cdot \left(-\vec{S}, 1\right) \\
&= \sum_{i=1}^{k}\sum_{j=1}^{\ell} A'_{i,j} \cdot (B^{(i,j)} - \vec{A}^{(i,j)} \cdot \vec{S}) + \sum_{j=1}^{\ell} B'_j \cdot (B^{(k+1,j)} - \vec{A}^{(k+1,j)} \cdot \vec{S}) \\
&= \sum_{i=1}^{k}\sum_{j=1}^{\ell} A'_{i,j} \cdot (-M_2 \frac{q}{\mathfrak{B}^j} S_i + E_2^{(i,j)}) + \sum_{j=1}^{\ell} B'_j \cdot (M_2 \frac{q}{\mathfrak{B}^j} + E_2^{(k+1,j)}) \\
&= \sum_{i=1}^{k}\sum_{j=1}^{\ell} A'_{i,j} \cdot E_2^{(i,j)} + \sum_{j=1}^{\ell} B'_j \cdot E_2^{(k+1,j)} - \sum_{i=1}^{k}\sum_{j=1}^{\ell} A'_{i,j} \cdot M_2 \frac{q}{\mathfrak{B}^j} S_i + \sum_{j=1}^{\ell} B'_j \cdot M_2 \frac{q}{\mathfrak{B}^j} \\
&= \sum_{j=1}^{\ell}\left(\sum_{i=1}^{k} (A'_{i,j} \cdot E_2^{(i,j)}) + B'_j \cdot E_2^{(k+1,j)}\right) + M_2 \left(-\sum_{i=1}^{k}\sum_{j=1}^{\ell} (A'_{i,j} \cdot \frac{q}{\mathfrak{B}^j}) S_i + \sum_{j=1}^{\ell} (B'_j \cdot \frac{q}{\mathfrak{B}^j})\right) \\
&= \sum_{j=1}^{\ell}\sum_{i=1}^{k+1} (A'_{i,j} \cdot E_2^{(i,j)}) + M_2 \left(-\sum_{i=1}^{k} A'_i \cdot S_i + B'\right) \\
&= \sum_{j=1}^{\ell}\sum_{i=1}^{k+1} (A'_{i,j} \cdot E_2^{(i,j)}) + M_2 \left(-\sum_{i=1}^{k} (A_i - \overline{A}_i) \cdot S_i + (B - \overline{B})\right) \\
&= \sum_{j=1}^{\ell}\sum_{i=1}^{k+1} (A'_{i,j} \cdot E_2^{(i,j)}) + M_2 \left(B - \sum_{i=1}^{k} A_i \cdot S_i - \overline{B} + \sum_{i=1}^{k} \overline{A}_i \cdot S_i\right) \\
&= \sum_{j=1}^{\ell}\sum_{i=1}^{k+1} (A'_{i,j} \cdot E_2^{(i,j)}) + M_2 \left(M_1 + E_1 - \overline{B} + \sum_{i=1}^{k} \overline{A}_i \cdot S_i\right) \\
&= \underbrace{\sum_{j=1}^{\ell}\sum_{i=1}^{k+1} (A'_{i,j} \cdot E_2^{(i,j)}) + M_2 \cdot \left(E_1 - \overline{B} + \sum_{i=1}^{k} \overline{A}_i \cdot S_i\right)}_{E} + M_1 \cdot M_2
\end{aligned}$$

So:

$$E = \sum_{j=1}^{\ell}\sum_{i=1}^{k+1}(A'_{i,j} \cdot E_2^{(i,j)}) + M_2 \cdot \left(E_1 - \overline{B} + \sum_{i=1}^{k} \overline{A}_i \cdot S_i\right)$$

Let us establish the variance of E:

$$\mathsf{Var}(E) = \underbrace{\mathsf{Var}\left(\sum_{j=1}^{\ell}\sum_{i=1}^{k+1}(A'_{i,j} \cdot E_2^{(i,j)})\right)}_{(1)} + \underbrace{\mathsf{Var}\left(M_2 \cdot \left(E_1 - \overline{B} + \sum_{i=1}^{k} \overline{A}_i \cdot S_i\right)\right)}_{(2)}$$

We give more details on steps (1) *and* (2).

Step (1). *Observe that $A'_{i,j} \in \mathfrak{R}$ with coefficients from $\mathcal{U}(\llbracket -\frac{\mathfrak{B}}{2}, \frac{\mathfrak{B}}{2} \rrbracket)$, so:*

- $\mathbb{E}(A'_{i,j}) = -\frac{1}{2}$,
- $\mathsf{Var}(A'_{i,j}) = \frac{\mathfrak{B}^2-1}{12}$.

$E_2^{(i,j)} \in \mathfrak{R}_q$ *is such that each coefficient is sampled from $\mathcal{N}(0, \sigma_2^2)$. Then,*

$$\begin{aligned}
&\mathsf{Var}\left(\sum_{j=1}^{\ell}\sum_{i=1}^{k+1}(A'_{i,j} \cdot E_2^{(i,j)})\right)\\
&= \ell \cdot (k+1) \cdot \mathsf{Var}\left(A'_{i,j} \cdot E_2^{(i,j)}\right) = \ell \cdot (k+1) \cdot N \cdot \mathsf{Var}\left(a'_{i,j} \cdot e_2^{(i,j)}\right)\\
&= \ell \cdot (k+1) \cdot N \cdot \left(\mathsf{Var}\left(a'_{i,j}\right) \cdot \mathsf{Var}\left(e_2^{(i,j)}\right) + \mathbb{E}^2\left(a'_{i,j}\right) \cdot \mathsf{Var}\left(e_2^{(i,j)}\right) + \mathsf{Var}\left(a'_{i,j}\right) \cdot \mathbb{E}^2\left(e_2^{(i,j)}\right)\right)\\
&= \ell \cdot (k+1) \cdot N \cdot \left(\frac{\mathfrak{B}^2-1}{12} \cdot \sigma_2^2 + \frac{1}{4} \cdot \sigma_2^2\right)\\
&= \ell \cdot (k+1) \cdot N \cdot \frac{\mathfrak{B}^2+2}{12} \cdot \sigma_2^2 .
\end{aligned}$$

Step (2). *Observe that $M_2 \in \mathfrak{R}$ and we have no information about the distribution. Observe that $E_1 \in \mathfrak{R}_q$ is such that each coefficient is sampled from $\mathcal{N}(0, \sigma_1^2)$. Observe that $\overline{B}, \overline{A}_i$ come from $\mathcal{U}(\llbracket -\frac{q}{2\mathfrak{B}^\ell}, \frac{q}{2\mathfrak{B}^\ell} \rrbracket)$. So:*

- $\mathbb{E}(\overline{A}_i) = \mathbb{E}(\overline{B}) = -\frac{1}{2}$,
- $\mathsf{Var}(\overline{A}_i) = \mathsf{Var}(\overline{B}) = \frac{q^2}{12\mathfrak{B}^{2\ell}} - \frac{1}{12} = \frac{q^2 - \mathfrak{B}^{2\ell}}{12\mathfrak{B}^{2\ell}}$.

Then,

$$\mathsf{Var}\left(M_2 \cdot \left(E_1 - \overline{B} + \sum_{i=1}^{k} \overline{A}_i \cdot S_i\right)\right) = ||M_2||_2^2 \cdot \mathsf{Var}\left(E_1 - \overline{B} + \sum_{i=1}^{k} \overline{A}_i \cdot S_i\right)$$

where

$$\begin{aligned}
&\mathsf{Var}\left(E_1 - \overline{B} + \sum_{i=1}^{k} \overline{A}_i \cdot S_i\right) \\
&\quad = \mathsf{Var}\left(E_1\right) + \mathsf{Var}\left(\overline{B}\right) + \mathsf{Var}\left(\sum_{i=1}^{k} \overline{A}_i \cdot S_i\right) \\
&\quad = \mathsf{Var}\left(E_1\right) + \mathsf{Var}\left(\overline{B}\right) + kN \cdot \mathsf{Var}\left(\overline{a}_i \cdot s_i\right) \\
&\quad = \sigma_1^2 + \frac{q^2 - \mathfrak{B}^{2\ell}}{12\mathfrak{B}^{2\ell}} + kN \cdot \big(\mathsf{Var}\left(\overline{a}_i\right) \cdot \mathsf{Var}\left(s_i\right) + \mathbb{E}^2\left(\overline{a}_i\right) \cdot \mathsf{Var}\left(s_i\right) + \mathsf{Var}\left(\overline{a}_i\right) \cdot \mathbb{E}^2\left(s_i\right)\big) \\
&\quad = \sigma_1^2 + \frac{q^2 - \mathfrak{B}^{2\ell}}{12\mathfrak{B}^{2\ell}} + kN \cdot \left(\frac{q^2 - \mathfrak{B}^{2\ell}}{12\mathfrak{B}^{2\ell}} \cdot \big(\mathsf{Var}\left(s_i\right) + \mathbb{E}^2\left(s_i\right)\big) + \frac{1}{4} \cdot \mathsf{Var}\left(s_i\right)\right) \\
&\quad = \sigma_1^2 + \frac{q^2 - \mathfrak{B}^{2\ell}}{12\mathfrak{B}^{2\ell}} \cdot \big(1 + kN \cdot \big(\mathsf{Var}(s_i) + \mathbb{E}^2(s_i)\big)\big) + \frac{kN}{4} \cdot \mathsf{Var}(s_i)\ .
\end{aligned}$$

If $M_2 \in \{0, 1\}$ is a bit of the binary key as in the TFHE bootstrapping ($\mathbb{E}(M_2) = \frac{1}{2}$ and $\mathsf{Var}(M_2) = \frac{1}{4}$ and M_2 is a constant polynomial which we note m_2), then we have:

$$\begin{aligned}
&\mathsf{Var}\left(m_2 \cdot \left(E_1 - \overline{B} + \sum_{i=1}^{k} \overline{A}_i \cdot S_i\right)\right) \\
&= \big(\mathsf{Var}\left(m_2\right) + \mathbb{E}^2\left(m_2\right)\big) \cdot \mathsf{Var}\left(E_1 - \overline{B} + \sum_{i=1}^{k} \overline{A}_i \cdot S_i\right) + \mathsf{Var}\left(m_2\right) \cdot \mathbb{E}^2\left(E_1 - \overline{B} + \sum_{i=1}^{k} \overline{A}_i \cdot S_i\right) \\
&= \frac{1}{2} \cdot \mathsf{Var}\left(E_1 - \overline{B} + \sum_{i=1}^{k} \overline{A}_i \cdot S_i\right) + \frac{1}{4} \cdot \mathbb{E}^2\left(E_1 - \overline{B} + \sum_{i=1}^{k} \overline{A}_i \cdot S_i\right)
\end{aligned}$$

Observe that

$$\begin{aligned}
\mathbb{E}\left(E_1 - \overline{B} + \sum_{i=1}^{k} \overline{A}_i \cdot S_i\right) &= \mathbb{E}\left(E_1\right) - \mathbb{E}\left(\overline{B}\right) + \mathbb{E}\left(\sum_{i=1}^{k} \overline{A}_i \cdot S_i\right) \\
&= 0 + \frac{1}{2} + kN \cdot \mathbb{E}\left(\overline{a}_i\right) \cdot \mathbb{E}\left(s_i\right) = \frac{1}{2} - \frac{kN}{2} \cdot \mathbb{E}\left(s_i\right) \\
&= \frac{1}{2} \cdot \left(1 - kN \cdot \mathbb{E}(s_i)\right)
\end{aligned}$$

and

$$\mathbb{E}^2\left(E_1 - \overline{B} + \sum_{i=1}^{k} \overline{A}_i \cdot S_i\right) = \frac{1}{4} \cdot \left(1 - kN \cdot \mathbb{E}(s_i)\right)^2\ .$$

Hence

$$\begin{aligned}
&\mathsf{Var}\left(m_2 \cdot \left(E_1 - \overline{B} + \sum_{i=1}^{k} \overline{A}_i \cdot S_i\right)\right) \\
&= \frac{1}{2} \cdot \mathsf{Var}\left(E_1 - \overline{B} + \sum_{i=1}^{k} \overline{A}_i \cdot S_i\right) + \frac{1}{4} \cdot \mathbb{E}^2\left(E_1 - \overline{B} + \sum_{i=1}^{k} \overline{A}_i \cdot S_i\right) \\
&= \frac{\sigma_1^2}{2} + \frac{q^2 - \mathfrak{B}^{2\ell}}{24\mathfrak{B}^{2\ell}} + \frac{kN}{2} \cdot \frac{q^2 - \mathfrak{B}^{2\ell}}{12\mathfrak{B}^{2\ell}} \cdot \left(\mathsf{Var}\left(s_i\right) + \mathbb{E}^2\left(s_i\right)\right) + \frac{kN}{8} \cdot \mathsf{Var}(s_i) + \frac{1}{16} \cdot \left(1 - kN \cdot \mathbb{E}(s_i)\right)^2 \\
&= \frac{\sigma_1^2}{2} + \frac{q^2 - \mathfrak{B}^{2\ell}}{24\mathfrak{B}^{2\ell}} \cdot \left(1 + kN \cdot \left(\mathsf{Var}\left(s_i\right) + \mathbb{E}^2\left(s_i\right)\right)\right) + \frac{kN}{8} \cdot \mathsf{Var}(s_i) + \frac{1}{16} \cdot \left(1 - kN \cdot \mathbb{E}(s_i)\right)^2 \ .
\end{aligned}$$

Finally, if $M_2 \in \{0,1\}$ *is a bit of the binary key as in the TFHE's PBS* ($\mathbb{E} = \frac{1}{2}$ *and* $\mathsf{Var} = \frac{1}{4}$*), then we have:*

$$\begin{aligned}
\mathsf{Var}(E) &= \underbrace{\mathsf{Var}\left(\sum_{j=1}^{\ell}\sum_{i=1}^{k+1}\left(A'_{i,j} \cdot E_2^{(i,j)}\right)\right)}_{(1)} + \underbrace{\mathsf{Var}\left(M_2 \cdot \left(E_1 - \overline{B} + \sum_{i=1}^{k} \overline{A}_i \cdot S_i\right)\right)}_{(2)} \\
&= \underbrace{\ell \cdot (k+1) \cdot N \cdot \frac{\mathfrak{B}^2 + 2}{12} \cdot \sigma_2^2}_{(1)} + \\
&\quad + \underbrace{\frac{\sigma_1^2}{2} + \frac{q^2 - \mathfrak{B}^{2\ell}}{24\mathfrak{B}^{2\ell}} \cdot \left(1 + kN \cdot \left(\mathsf{Var}\left(s_i\right) + \mathbb{E}^2\left(s_i\right)\right)\right) + \frac{kN}{8} \cdot \mathsf{Var}(s_i) + \frac{1}{16} \cdot \left(1 - kN \cdot \mathbb{E}(s_i)\right)^2}_{(2)}
\end{aligned}$$

The external product is used to compute the ***CMux****, which is used in the programmable bootstrapping (PBS). In the PBS, we have*

- *The message* m_2 *in the* GGSW *is one of the bits of the* LWE *secret key, so it comes from a uniform distribution* $\mathcal{U}(\{0,1\})$*;*
- *The CMux computes the following*

  $$(ACC \cdot X^{-a_i} - ACC) \boxdot \mathsf{BSK}_i + ACC$$

  and the noise added by a CMux is the same as the noise produced by one external product;
- *The CMux in the PBS is repeated* n *times;*
- *The initial* σ_{RLWE} *in the PBS is equal to* 0.

Then, the noise in the PBS can be estimated by

$$\begin{aligned}\mathsf{Var}(\mathsf{PBS}) &= n \cdot \ell \cdot (k+1) \cdot N \cdot \frac{\mathfrak{B}^2+2}{12} \cdot \mathsf{Var}(\mathsf{BSK}) + \\ &+ n \cdot \frac{q^2 - \mathfrak{B}^{2\ell}}{24\mathfrak{B}^{2\ell}} \cdot \left(1 + kN \cdot \left(\mathsf{Var}\left(s_i\right) + \mathbb{E}^2\left(s_i\right)\right)\right) + \frac{nkN}{8} \cdot \mathsf{Var}(s_i) + \frac{n}{16} \cdot \left(1 - kN \cdot \mathbb{E}(s_i)\right)^2 .\end{aligned}$$

If the RLWE *secret key is binary*

$$\begin{aligned}\mathsf{Var}(\mathsf{PBS}) &= n\ell(k+1)N \cdot \frac{\mathfrak{B}^2+2}{12} \cdot \mathsf{Var}(\mathsf{BSK}) + \\ &+ n \cdot \frac{q^2 - \mathfrak{B}^{2\ell}}{24\mathfrak{B}^{2\ell}} \cdot \left(1 + \frac{kN}{2}\right) + \frac{nkN}{32} + \frac{n}{16} \cdot \left(1 - \frac{kN}{2}\right)^2 .\end{aligned}$$

□

A.3 Noise Analysis of the Packing Key Switch

We consider the LWE secret key $\vec{s} = (s_1, \cdots, s_n) \in \mathbb{Z}_q^n$ and an input LWE ciphertext $\mathsf{ct} = (a_1, \cdots, a_n, b) \in \mathbb{Z}_q^{n+1}$ such that $b = \sum_{i=1}^n a_i s_i + m + e$ with e from χ_σ.

We consider the GLWE secret key $\vec{S}{=}(S_1,\cdots,S_k){\in}\mathfrak{R}_q^k$ and a key switching key composed of the following GLWE ciphertexts $\{C^{(i,j)}{=}(A_1^{(i,j)},\cdots,A_k^{(i,j)},B^{(i,j)}){\in}\mathfrak{R}_q^{k+1}\}$ with $1 \leq i \leq n$ and $1 \leq j \leq \ell$ such that $B^{(i,j)} = \sum_{\psi=1}^k A_\psi^{(i,j)} \cdot S_\psi + s_i \frac{q}{\mathfrak{B}^j} + E^{(i,j)}$ with coefficients of $E^{(i,j)}$ from $\chi_{\sigma_{\mathsf{KSK}}}$.

During the algorithm we will round the $\{a_i\}$ to the closest multiple of $\frac{q}{\mathfrak{B}^\ell}$ and then decompose them such that for $1 \leq i \leq n$ we have $a'_i = a_i + \bar{a}_i$ and for $1 \leq j \leq \ell$ we have $a'_i = \sum_{j=1}^\ell a'_{i,j} \frac{q}{\mathfrak{B}^j}$ with $\bar{a}_i$ uniform in $\llbracket \frac{-q}{2\mathfrak{B}^\ell}, \frac{q}{2\mathfrak{B}^\ell} \llbracket$ and $a'_{i,j}$ uniform in $\llbracket \frac{-\mathfrak{B}}{2}, \frac{\mathfrak{B}}{2} \llbracket$.

We have $\mathsf{Var}(\bar{a}_i) = \frac{q^2}{12\mathfrak{B}^{2\ell}} - \frac{1}{12}$, $\mathsf{Var}(a'_{i,j}) = \frac{\mathfrak{B}^2-1}{12}$ and $\mathbb{E}(\bar{a}_i) = \mathbb{E}(a'_{i,j}) = -\frac{1}{2}$.

The output is:

$$\begin{aligned}C_{\mathsf{res}} &= (0, \cdots, 0, b) - \sum_{i=1}^n \sum_{j=1}^\ell a'_{i,j} \cdot C^{(i,j)} \\ &= \left(- \sum_{i=1}^n \sum_{j=1}^\ell a'_{i,j} \cdot A_1^{(i,j)}, \cdots, - \sum_{i=1}^n \sum_{j=1}^\ell a'_{i,j} \cdot A_k^{(i,j)}, b - \sum_{i=1}^n \sum_{j=1}^\ell a'_{i,j} \cdot B^{(i,j)} \right)\end{aligned}$$

Let's decrypt the output:

$$
\begin{aligned}
&\mathsf{Decrypt}(C_{\mathsf{res}}, \vec{S}) = \\
&= b - \sum_{i=1}^{n}\sum_{j=1}^{\ell} a'_{i,j} \cdot B^{(i,j)} - \left(\sum_{\psi=1}^{k} \left(- \sum_{i=1}^{n}\sum_{j=1}^{\ell} a'_{i,j} \cdot A_1^{(i,j)} \right) S_\psi \right) \\
&= b - \sum_{i=1}^{n}\sum_{j=1}^{\ell} a'_{i,j} \cdot B^{(i,j)} + \sum_{\psi=1}^{k}\sum_{i=1}^{n}\sum_{j=1}^{\ell} a'_{i,j} \cdot A_1^{(i,j)} \cdot S_\psi \\
&= b - \sum_{i=1}^{n}\sum_{j=1}^{\ell} a'_{i,j} \cdot \left(\cancel{\sum_{\psi=1}^{k} A_\psi^{(i,j)} \cdot S_\psi} + s_i \frac{q}{\mathfrak{B}^j} + E^{(i,j)} \right) + \cancel{\sum_{\psi=1}^{k}\sum_{i=1}^{n}\sum_{j=1}^{\ell} a'_{i,j} \cdot A_1^{(i,j)} \cdot S_\psi} \\
&= b - \sum_{i=1}^{n}\sum_{j=1}^{\ell} a'_{i,j} \cdot s_i \frac{q}{\mathfrak{B}^j} - \sum_{i=1}^{n}\sum_{j=1}^{\ell} a'_{i,j} \cdot E^{(i,j)} = b - \sum_{i=1}^{n} a'_i \cdot s_i - \sum_{i=1}^{n}\sum_{j=1}^{\ell} a'_{i,j} \cdot E^{(i,j)} \\
&= b - \sum_{i=1}^{n} (a_i + \bar{a}_i) \cdot s_i - \sum_{i=1}^{n}\sum_{j=1}^{\ell} a'_{i,j} \cdot E^{(i,j)} = b - \sum_{i=1}^{n} a_i \cdot s_i - \sum_{i=1}^{n} \bar{a}_i \cdot s_i - \sum_{i=1}^{n}\sum_{j=1}^{\ell} a'_{i,j} \cdot E^{(i,j)} \\
&= m + e - \sum_{i=1}^{n} \bar{a}_i \cdot s_i - \sum_{i=1}^{n}\sum_{j=1}^{\ell} a'_{i,j} \cdot E^{(i,j)}
\end{aligned}
$$

Let's now study the error term in the filled coefficient:

$$
\begin{aligned}
\mathsf{Var}_{\mathsf{fill}} &= \mathsf{Var}\left(e - \sum_{i=1}^{n} \bar{a}_i \cdot s_i - \sum_{i=1}^{n}\sum_{j=1}^{\ell} a'_{i,j} \cdot E^{(i,j)} \right) \\
&= \mathsf{Var}(e) + \mathsf{Var}\left(\sum_{i=1}^{n} \bar{a}_i \cdot s_i \right) + \mathsf{Var}\left(\sum_{i=1}^{n}\sum_{j=1}^{\ell} a'_{i,j} \cdot E^{(i,j)} \right) \\
&= \sigma^2 + n \cdot (\mathsf{Var}(\bar{a}_i)\mathsf{Var}(s_i) + \mathbb{E}^2(\bar{a}_i)\mathsf{Var}(s_i) + \mathbb{E}^2(s_i)\mathsf{Var}(\bar{a}_i)) + n \cdot \ell \cdot \sigma^2_{\mathsf{KSK}} \cdot (\mathsf{Var}(a'_{i,j}) + \mathbb{E}^2(a'_{i,j})) \\
&= \sigma^2 + n \cdot \left(\frac{q^2}{12\mathfrak{B}^{2\ell}} - \frac{1}{12} \right) \cdot \left(\mathsf{Var}(s_i) + \mathbb{E}^2(s_i) \right) + \frac{n}{4} \cdot \mathsf{Var}(s_i) + n \cdot \ell \cdot \sigma^2_{\mathsf{KSK}} \cdot \frac{\mathfrak{B}^2 + 2}{12}
\end{aligned}
$$

Let's finally study the error term in the other coefficients:

$$
\mathsf{Var}_{\mathsf{emp}} = \mathsf{Var}\left(\sum_{i=1}^{n}\sum_{j=1}^{\ell} a'_{i,j} \cdot E^{(i,j)} \right) = n \cdot \ell \cdot \sigma^2_{\mathsf{KSK}} \cdot \frac{\mathfrak{B}^2 + 2}{12}.
$$

A.4 Noise Analysis of the Sample Extract

Proof 45 (Correctness for Constant Sample Extraction) *As in algorithm 35, we consider a GLWE ciphertext* $\mathsf{CT}_{\mathsf{in}} := (A_0, \cdots, A_{k-1}, B) \in \mathsf{GLWE}_{\vec{S}^{[\phi]}}(P) \subseteq \mathfrak{R}_{q,N}^{k+1}$ *where* $P = \sum_{i=0}^{N-1} p_i X^i \in \mathfrak{R}_{q,N}$ *and for all* $0 \leq i \leq k-1$ *we have* $A_i = \sum_{j=0}^{N-1} a_{i,j} X^j$ *and* $B = \sum_{j=0}^{N-1} b_j X^j$*. The GLWE secret key is noted* $\vec{S}^{[\phi]} = (S_0, \cdots, S_{k-1}) \in \mathfrak{R}_{q,N}^k$ *and follows Definition 35. By definition of GLWE ciphertexts, it means that it exists an error polynomial* $E = \sum_{i=0}^{N-1} e_i X^i \in \mathfrak{R}_{q,N}$ *such that* $B - \sum_{i=0}^{k-1} A_i \cdot S_i = P + E$*.*

Following Algorithm 35, the constant sample extraction outputs the following LWE ciphertext: $\mathsf{ct}_{\mathsf{out}} = (a_{\mathsf{out},0}, \cdots, a_{\mathsf{out},\phi-1}, b_{\mathsf{out}}) \in \mathsf{LWE}_{\vec{\bar{s}}}(p_0) \subseteq \mathbb{Z}_q^{\phi+1}$ *encrypted under the LWE secret key* $\vec{\bar{s}} = (\bar{s}_0, \cdots, \bar{s}_{\phi-1}) \in \mathbb{Z}_q^{\phi}$ *obtained as defined in Definition 36.*

First we define two index functions, the first one is $\iota : i \mapsto \left(\left\lfloor \frac{i}{N} \right\rfloor, i \bmod N\right)$ and the second one is $\tilde{\iota} : i \mapsto \left(\left\lfloor \frac{i}{N} \right\rfloor, (N-i) \bmod N\right)$. We also need to define a last function $\gamma : i \mapsto 1 - ((i \bmod N) == 0)$

$$\begin{aligned}
b_{\mathsf{out}} - \sum_{i=0}^{\phi-1} a_{\mathsf{out},i} \cdot \bar{s}_i &= b_0 - \sum_{i=0}^{\phi-1} a_{\mathsf{out},i} \cdot \bar{s}_i - \underbrace{\sum_{i=\phi}^{kN-1} (-1)^{\gamma(i)} \cdot a_{\tilde{\iota}(i)} \cdot s_{\iota(i)}}_{\text{null since all } s_{\iota(i)}=0 \text{ because it is a partial key)}} \\
&= b_0 - \sum_{i=0}^{\phi-1} \underbrace{(-1)^{\gamma(i)} \cdot a_{\tilde{\iota}(i)} \cdot s_{\iota(i)}}_{\text{lines 2 and 3 in Algorithm 35}} - \sum_{i=\phi}^{kN-1} (-1)^{\gamma(i)} \cdot a_{\tilde{\iota}(i)} \cdot s_{\iota(i)} \\
&= b_0 - \sum_{i=0}^{kN-1} (-1)^{\gamma(i)} \cdot a_{\tilde{\iota}(i)} \cdot s_{\iota(i)} = b_0 - \sum_{i=0}^{k-1} \sum_{j=0}^{N-1} a_{i,N-j} \cdot s_{i,j} \\
&= b_0 - \sum_{i=0}^{k-1} \sum_{j=0}^{N-1} a_{i,(N-j) \bmod N} X^{(N-j) \bmod N} \cdot s_{i,j} X^j
\end{aligned} \tag{A.5}$$

This quantity is what we have on the constant term of the polynomial resulting from the decryption of $\mathsf{CT}_{\mathsf{in}}$:

$$\begin{aligned}
B - \sum_{i=0}^{k-1} A_i \cdot S_i &= B - \sum_{i=0}^{k-1} \sum_{j=0}^{N-1} \sum_{j'=0}^{N-1} a_{i,j} X^j \cdot s_{i,j'} X^{j'} \\
&= \underbrace{X^0 \cdot \left(b_0 - \sum_{i=0}^{k-1} \sum_{j=0}^{N-1} a_{i,(N-j) \bmod N} X^{(N-j) \bmod N} \cdot s_{i,j} X^j \right)}_{\text{constant coefficient with the same quantity}} \\
&\quad \underbrace{+ X^1 \cdot (b_1 - \ldots) \ + \ \cdots \ + X^{N-1} \cdot (b_{N-1} - \ldots)}_{\text{non-constant coefficients}}
\end{aligned} \tag{A.6}$$

$\square$

Proof 46 (Correctness for Sample Extraction) *We follow the context and inputs of Algorithm 36. It is trivial to show that the α-th coefficient of the decryption of $\mathsf{CT}_{\mathsf{in}}$ is equal to what is in the constant coefficient of $X^{-\alpha} \cdot \mathsf{CT}_{\mathsf{in}}$.*

$\square$

A.5 Noise Analysis of the GLWE-to-GLWE Key Switch

Proof 47 (Theorem 31) *The inputs of a* GLWE-*to*-GLWE *key switching (Algorithm 37) are:*

- *The input GLWE ciphertext:* $\mathsf{CT}_{\mathsf{in}} = \left(\vec{A}_{\mathsf{in}}, B_{\mathsf{in}}\right) \in \mathsf{GLWE}_{\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}}(\Delta \cdot M) \subseteq \mathfrak{R}_{q,N}^{k_{\mathsf{in}}+1}$, *where* $B_{\mathsf{in}} = \sum_{i=0}^{k_{\mathsf{in}}-1} A_{\mathsf{in},i} \cdot S_{\mathsf{in},i} + \Delta \cdot M + E_{\mathsf{in}}$, $A_{\mathsf{in},i} = \sum_{j=0}^{N-1} a_{i,j} \cdot X^j \hookleftarrow \mathcal{U}(\mathfrak{R}_{q,N})$ *for all* $i \in [\![0, k[\![$ *and* $E_{\mathsf{in}} = \sum_{j=0}^{N-1} e_j \cdot X^j$, *and* $e_j \hookleftarrow \mathcal{N}_{\sigma_{\mathsf{in}}^2}$ *for all* $j \in [\![0, N-1[\![$.

- *The key switch key:* $\mathsf{KSK} = (\mathsf{KSK}_0, \ldots, \mathsf{KSK}_{k_{\mathsf{in}}-1})$, *where* $\mathsf{KSK}_i \in \mathsf{GLev}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}(S_{\mathsf{in},i}) = \left(\mathsf{GLWE}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}\left(\frac{q}{\beta} S_{\mathsf{in},i}\right), \cdots, \mathsf{GLWE}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}\left(\frac{q}{\beta^\ell} S_{\mathsf{in},i}\right)\right)$ *for all* $0 \le i < k_{\mathsf{in}}$. *We note by* $\mathsf{KSK}_{i,j} = (\vec{A_{i,j}}, B_{i,j}) \in \mathsf{GLWE}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}\left(\frac{q}{\beta^{j+1}} S_{\mathsf{in},i}\right)$, *for all* $0 \le i < k_{\mathsf{in}}$ *and for all* $0 \le j < \ell$, *where* $B_{i,j} = \sum_{\tau=0}^{k_{\mathsf{out}}-1} A_{i,j,\tau} \cdot S_{\mathsf{out},\tau}^{[\phi_{\mathsf{out}}]} + \frac{q}{\beta^{j+1}} S_{\mathsf{in},i} + E_{\mathsf{ksk},i,j}$, *and* $E_{\mathsf{ksk},i,j} = \sum_{\tau=0}^{N-1} e_{\mathsf{ksk},i,j,\tau} \cdot X^\tau$ *and* $e_{\mathsf{ksk},i,j,\tau} \hookleftarrow \mathcal{N}_{\sigma_{\mathsf{ksk}}^2}$.

The output of this algorithm is $\mathsf{CT}_{\mathsf{out}} = \left(\vec{A}_{\mathsf{out}}, B_{\mathsf{out}}\right) \in \mathsf{GLWE}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}(\Delta \cdot M) \subseteq \mathfrak{R}_{q,N}^{k_{\mathsf{out}}+1}$. *By definition, in the decomposition algorithm, we have that* $\mathsf{dec}^{(\mathfrak{B},\ell)}(A_{\mathsf{in},i}) = \left(\widetilde{A}_{\mathsf{in},i,0}, \cdots, \widetilde{A}_{\mathsf{in},i,\ell-1}\right)$ *such that* $\widetilde{A}_{\mathsf{in},i} = \sum_{j=0}^{\ell-1} \frac{q}{\beta^{j+1}} \widetilde{A}_{\mathsf{in},i,j}$, *for all* $0 \le i < k_{\mathsf{in}}$.
Let define $\bar{A}_{\mathsf{in},i} = A_{\mathsf{in},i} - \widetilde{A}_{\mathsf{in},i}$, $|\bar{a}_{i,\tau}| = |a_{i,\tau} - \widetilde{a}_{i,\tau}| < \frac{q}{2\beta^\ell}$, $\bar{a}_{i,\tau} \in \left[\!\!\left[\frac{-q}{2\beta^\ell}, \frac{q}{2\beta^\ell}\right[\!\!\right[$ *for all* $0 \le \tau < N$. *So we have that their expectations and variances are respectively* $\mathbb{E}(\bar{a}_{i,\tau}) = -\frac{1}{2}$, $\mathsf{Var}(\bar{a}_{i,\tau}) = \frac{q^2}{12\mathfrak{B}^{2\ell}} - \frac{1}{12}$, $\mathbb{E}(\widetilde{a}_{i,\tau}) = -\frac{1}{2}$ *and* $\mathsf{Var}(\widetilde{a}_{i,\tau}) = \frac{\beta^2-1}{12}$.
Now, we can decrypt:

$$\begin{aligned}
B_{\mathsf{out}} - \left\langle \vec{A}_{\mathsf{out}}, \vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]} \right\rangle &= \left\langle \left(\vec{A}_{\mathsf{out}}, B_{\mathsf{out}}\right), \left(-\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}, 1\right) \right\rangle \\
&= \left\langle (\vec{0}, B_{\mathsf{in}}) - \sum_{i=0}^{k_{\mathsf{in}}-1} \mathsf{dec}^{(\beta,\ell)}(A_{\mathsf{in},i}) \cdot \mathsf{KSK}_i, \left(-\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}, 1\right) \right\rangle \\
&= B_{\mathsf{in}} - \sum_{i=0}^{k_{\mathsf{in}}-1} \sum_{j=0}^{\ell-1} \widetilde{A}_{\mathsf{in},i,j} \left\langle \mathsf{KSK}_{i,j}, \left(-\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}, 1\right) \right\rangle \\
&= B_{\mathsf{in}} - \sum_{i=0}^{k_{\mathsf{in}}-1} \sum_{j=0}^{\ell-1} \widetilde{A}_{\mathsf{in},i,j} \left(\frac{q}{\beta^{j+1}} S_{\mathsf{in},i} + E_{\mathsf{ksk},i,j} \right) \\
&= \underbrace{B_{\mathsf{in}} - \sum_{i=0}^{k_{\mathsf{in}}-1} \widetilde{A}_{\mathsf{in},i} S_{\mathsf{in},i}}_{(I)} - \underbrace{\sum_{i=0}^{k_{\mathsf{in}}-1} \sum_{j=0}^{\ell-1} \widetilde{A}_{\mathsf{in},i,j} \cdot E_{\mathsf{ksk},i,j}}_{(II)}
\end{aligned}$$

Now let us focus on the w^{th} *coefficient of part (I):*

$$\begin{aligned}
& b_{\mathsf{in},w} - \sum_{i=0}^{k_{\mathsf{in}}-1} \left(\sum_{\tau=0}^{w} \widetilde{a}_{\mathsf{in},i,w-\tau} \cdot s_{\mathsf{in},i,\tau} - \sum_{\tau=w+1}^{N-1} \widetilde{a}_{\mathsf{in},i,N+w-\tau} \cdot s_{\mathsf{in},i,\tau} \right) \\
&= b_{\mathsf{in},w} - \sum_{i=0}^{k_{\mathsf{in}}-1} \left(\sum_{\tau=0}^{w} (a_{\mathsf{in},i,w-\tau} - \bar{a}_{\mathsf{in},i,w-\tau}) \cdot s_{\mathsf{in},i,\tau} - \sum_{\tau=w+1}^{N-1} (a_{\mathsf{in},i,N+w-\tau} - \bar{a}_{\mathsf{in},i,N+w-\tau}) \cdot s_{\mathsf{in},i,\tau} \right)
\end{aligned}$$

$$= \Delta m_w + e_w + \sum_{i=0}^{k_{\mathsf{in}}-1}\left(\sum_{\tau=0}^{w}\bar{a}_{\mathsf{in},i,w-\tau}\cdot s_{\mathsf{in},i,\tau} - \sum_{\tau=w+1}^{N-1}\bar{a}_{\mathsf{in},i,N+w-\tau}\cdot s_{\mathsf{in},i,\tau}\right)$$

Now let us focus on the w^{th} coefficient of part (II):

$$\sum_{i=0}^{k_{\mathsf{in}}-1}\sum_{j=0}^{\ell-1}\left(\sum_{\tau=0}^{w}\widetilde{a}_{\mathsf{in},i,j,w-\tau}\cdot e_{\mathsf{ksk},i,j,\tau} - \sum_{\tau=w+1}^{N-1}\widetilde{a}_{\mathsf{in},i,j,N+w-\tau}\cdot e_{\mathsf{ksk},i,j,\tau}\right)$$

We can now isolate the output error for the w^{th} coefficient and remove the message coefficient. We obtain the output error

$$\begin{aligned}e'_w = e_w &+ \underbrace{\sum_{i=0}^{k_{\mathsf{in}}-1}\left(\sum_{\tau=0}^{w}\bar{a}_{\mathsf{in},i,w-\tau}\cdot s_{\mathsf{in},i,\tau} - \sum_{\tau=w+1}^{N-1}\bar{a}_{\mathsf{in},i,N+w-\tau}\cdot s_{\mathsf{in},i,\tau}\right)}_{(*)}\\ &+ \sum_{i=0}^{k_{\mathsf{in}}-1}\sum_{j=0}^{\ell-1}\left(\sum_{\tau=0}^{w}\widetilde{a}_{\mathsf{in},i,j,w-\tau}\cdot e_{\mathsf{ksk},i,j,\tau} - \sum_{\tau=w+1}^{N-1}\widetilde{a}_{\mathsf{in},i,j,N+w-\tau}\cdot e_{\mathsf{ksk},i,j,\tau}\right)\ .\end{aligned}$$

Observe that in the term $(*)$ there are $k_{\mathsf{in}}N - \phi_{\mathsf{in}}$ terms of type $\bar{a}_{\mathsf{in},i,\cdot}\cdot s_{\mathsf{in},i,\cdot}$ that are equal to 0. So we have:

$$\begin{aligned}\mathsf{Var}(e'_w) &= \mathsf{Var}(e_w) + \phi_{\mathsf{in}}\cdot\mathsf{Var}\left(\bar{a}_{\mathsf{in},i,\cdot}\cdot s_{\mathsf{in},i,\cdot}\right) + k_{\mathsf{in}}\cdot\ell\cdot N\cdot\mathsf{Var}\left(\widetilde{a}_{\mathsf{in},i,j,\cdot}\cdot e_{\mathsf{ksk},i,j,\cdot}\right)\\ &= \sigma^2_{\mathsf{in}} + \phi_{\mathsf{in}}\left(\mathsf{Var}\left(\bar{a}_{\mathsf{in},i,\cdot}\right)\mathsf{Var}\left(s_{\mathsf{in},i,\cdot}\right) + \mathsf{Var}\left(\bar{a}_{\mathsf{in},i,\cdot}\right)\mathbb{E}^2\left(s_{\mathsf{in},i,\cdot}\right) + \mathbb{E}^2(\bar{a}_{\mathsf{in},i,\cdot})\mathsf{Var}\left(s_{\mathsf{in},i,\cdot}\right)\right)\\ &\quad + \ell k_{\mathsf{in}}N\left(\mathsf{Var}\left(\widetilde{a}_{\mathsf{in},i,j,\cdot}\right)\mathsf{Var}\left(e_{\mathsf{ksk},i,j,\cdot}\right) + \mathbb{E}^2\left(\widetilde{a}_{\mathsf{in},i,j,\cdot}\right)\mathsf{Var}\left(e_{\mathsf{ksk},i,j,\cdot}\right)\right.\\ &\quad \left.+ \mathsf{Var}\left(\widetilde{a}_{\mathsf{in},i,j,\cdot}\right)\mathbb{E}^2\left(e_{\mathsf{ksk},i,j,\cdot}\right)\right)\\ &= \sigma^2_{\mathsf{in}} + \phi_{\mathsf{in}}\left(\frac{q^2-\beta^{2\ell}}{12\beta^{2\ell}}\right)\left(\mathsf{Var}\left(\vec{S}^{[\phi_{\mathsf{in}}]}_{\mathsf{in}}\right) + \mathbb{E}^2\left(\vec{S}^{[\phi_{\mathsf{in}}]}_{\mathsf{in}}\right)\right)\\ &\quad + \frac{\phi_{\mathsf{in}}}{4}\mathsf{Var}\left(\vec{S}^{[\phi_{\mathsf{in}}]}_{\mathsf{in}}\right) + \ell k_{\mathsf{in}}N\frac{\beta^2+2}{12}\sigma^2_{ksk}.\end{aligned}$$

□

A.6 Noise Analysis of the Secret Product GLWE-to-GLWE Key Switch

Proof 48 (Theorem 32) *This proof is similar to the proof proposed for the key switch with partial key (Proof 47).*
The inputs of the secret-product GLWE key switch (Algorithm 38) are:

- *The input GLWE ciphertext:* $\mathsf{CT}_{\mathsf{in}} = \left(\vec{A}_{\mathsf{in}}, B_{\mathsf{in}}\right) \in \mathsf{GLWE}_{\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}}(\Delta \cdot M_1) \subseteq \mathfrak{R}_{q,N}^{k_{\mathsf{in}}+1}$, *where* $B_{\mathsf{in}} = \sum_{i=0}^{k_{\mathsf{in}}-1} A_{\mathsf{in},i} \cdot S_{\mathsf{in},i}^{[\phi_{\mathsf{in}}]} + \Delta \cdot M_1 + E_{\mathsf{in}}$, $A_{\mathsf{in},i} = \sum_{j=0}^{k-1} a_{i,j} \cdot X^j \hookleftarrow \mathcal{U}(\mathfrak{R}_{q,N})$ *for all* $i \in [\![0, k[\![$ *and* $E_{\mathsf{in}} = \sum_{j=0}^{k-1} e_j \cdot X^j$, *and* $e_j \hookleftarrow \mathcal{N}_{\sigma_{\mathsf{in}}^2}$ *for all* $j \in [\![0, N-1[\![$.

- *The secret product key switch key :* $\mathsf{KSK} = (\mathsf{KSK}_0, \ldots, \mathsf{KSK}_{k_{\mathsf{in}}})$, *where* $\mathsf{KSK}_i \in \mathsf{GLev}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}\left(-M_2 S_{\mathsf{in},i}^{[\phi_{\mathsf{in}}]}\right) = \left(\mathsf{GLWE}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}\left(-\frac{qM_2}{\mathfrak{B}} S_{\mathsf{in},i}^{[\phi_{\mathsf{in}}]}\right), \cdots, \mathsf{GLWE}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}\left(-\frac{qM_2}{\mathfrak{B}^\ell} S_{\mathsf{in},i}^{[\phi_{\mathsf{in}}]}\right)\right)$ *for all* $0 \leq i \leq k_{\mathsf{in}}$ *(for this proof, we define* $S_{k_{\mathsf{in}}} = -1$*). We note by* $\mathsf{KSK}_{i,j} = (\vec{A_{i,j}}, B_{i,j}) \in \mathsf{GLWE}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}\left(-\frac{qM_2}{\mathfrak{B}^{j+1}} S_{\mathsf{in},i}^{[\phi_{\mathsf{in}}]}\right)$, *for all* $0 \leq i < k_{\mathsf{in}}$ *and for all* $0 \leq j < \ell$, *where* $B_{i,j} = \sum_{\tau=0}^{k_{\mathsf{out}}-1} A_{i,j,\tau} \cdot S_{\mathsf{out},\tau}^{[\phi_{\mathsf{out}}]} + \frac{qM_2}{\mathfrak{B}^{j+1}} S_{\mathsf{in},i}^{[\phi_{\mathsf{in}}]} + E_{\mathsf{ksk},i,j}$, *and* $E_{\mathsf{ksk},i,j} = \sum_{\tau=0}^{N-1} e_{\mathsf{ksk},i,j,\tau} \cdot X^\tau$ *and* $e_{\mathsf{ksk},i,j,m} \hookleftarrow \mathcal{N}_{\sigma_{\mathsf{ksk}}^2}$.

As output we obtain $\mathsf{CT}_{\mathsf{out}} = \left(\vec{A}_{\mathsf{out}}, B_{\mathsf{out}}\right) \in \mathsf{GLWE}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}(\Delta \cdot M_1 \cdot M_2) \subseteq \mathfrak{R}_{q,N}^{k_{\mathsf{out}}+1}$.
By definition, for any random polynomial A_i, *we have* $A_i = \sum_{j=0}^{N-1} a_{i,j} \cdot X^j$ *where* $a_{i,j} \sim \mathbb{U}(\mathbb{Z}_q)$,*(* $a_{i,j} \in [\![\frac{-q}{2}, \frac{q}{2}[\![$*).*
By definition, for the decomposition (described in Section 2.3.2), we have $\mathsf{dec}^{(\mathfrak{B},\ell)}(A_i) = \left(\tilde{A}_{i,0}, \cdots, \tilde{A}_{i,\ell-1}\right)$ *such that* $\tilde{A}_i = \sum_{j=0}^{\ell-1} \frac{q}{\mathfrak{B}^{j+1}} \tilde{A}_{i,j}$.
Let define $\bar{A}_i = |A_i - \tilde{A}_i|$, $\bar{a}_{i,j} = |a_{i,j} - \tilde{a_{i,j}}| < \frac{q}{2\mathfrak{B}^\ell}$*;* $\bar{a}_{i,j} \in [\![\frac{-q}{2\mathfrak{B}^\ell}, \frac{q}{2\mathfrak{B}^\ell}[\![$. *Finally we obtain* $\mathbb{E}(\bar{a}_i) = -\frac{1}{2}$*;* $\mathsf{Var}(\bar{a}_i) = \frac{q^2}{12\mathfrak{B}^{2\ell}} - \frac{1}{12}$*;* $\mathbb{E}(\tilde{a}_{i,j}) = -\frac{1}{2}$*;* $\mathsf{Var}(\tilde{a}_i) = \frac{\mathfrak{B}^2-1}{12}$.
As B_i *is seen as an uniform polynomial, we obtain the same results for the variance and the expectation for* $\tilde{B}_i$ *(resp.* $\bar{B}_i$*) than* $\tilde{A}_i$ *(Resp.* $\bar{A}_i$*). In the next calculations,* $\tilde{B}_{\mathsf{in},j} \cdot E_j$ *will be writen as* $-\tilde{A}_{\mathsf{in},k_{\mathsf{in}},j} \cdot E_{k_{\mathsf{in}},j}$
Now, we can compute the decryption:

$$\begin{aligned}
B_{\mathsf{out}} - \left\langle \vec{A}_{\mathsf{out}}, \vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]} \right\rangle =& \left\langle \left(\vec{A}_{\mathsf{out}}, B_{\mathsf{out}}\right); \left(-\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}, 1\right) \right\rangle \\
=& \left\langle \mathsf{dec}^{(\mathfrak{B},\ell)}(B_{\mathsf{in}}) \cdot \mathsf{KSK}_{k_{\mathsf{in}}} + \sum_{i=0}^{k_{\mathsf{in}}-1} \mathsf{dec}^{(\mathfrak{B},\ell)}(A_{\mathsf{in},i}) \cdot \mathsf{KSK}_i; \left(-\vec{S}^{[\phi_{\mathsf{out}}]}, 1\right) \right\rangle \\
=& M_2 \left(\tilde{B}_{\mathsf{in},i} - \sum_{i=0}^{k_{\mathsf{in}}-1} \tilde{A}_{\mathsf{in},i} \cdot S_{\mathsf{in},i} \right) - \sum_{i=0}^{k_{\mathsf{in}}} \sum_{j=0}^{\ell-1} \tilde{A}_{\mathsf{in},i,j} \cdot E_{i,j} \\
=& M_2 \left(\Delta M_1 + E_{\mathsf{in}} + \bar{B}_{\mathsf{in},i} - \sum_{i=0}^{k_{\mathsf{in}}-1} \bar{A}_{\mathsf{in},i} \cdot S_{\mathsf{in},i} \right) - \sum_{i=0}^{k_{\mathsf{in}}} \sum_{j=0}^{\ell-1} \tilde{A}_{\mathsf{in},i,j} \cdot E_{i,j} \\
=& \Delta M_2 \cdot M_1 + M_2 \left(E_{\mathsf{in}} + \bar{B}_{\mathsf{in},i} - \sum_{i=0}^{k_{\mathsf{in}}-1} \bar{A}_{\mathsf{in},i} \cdot S_{\mathsf{in},i} \right) - \sum_{i=0}^{k_{\mathsf{in}}} \sum_{j=0}^{\ell-1} \tilde{A}_{\mathsf{in},i,j} \cdot E_{i,j}
\end{aligned}$$

By following the same idea as the proof of the key switch (proof 47), we can isolate the noise and compute his variance. We obtain:

$$\begin{aligned}
&\mathsf{Var}\left(M_2\left(E+\bar{B}_{\mathsf{in},i}-\sum_{i=0}^{k_{\mathsf{in}}-1}\bar{A}_{\mathsf{in},i}\cdot S_{\mathsf{in},i}\right)-\sum_{i=0}^{k_{\mathsf{in}}}\sum_{j=0}^{\ell-1}\tilde{A}_{\mathsf{in},i,j}\cdot E_{i,j}\right)\\
&=||M_2||_2^2\cdot\mathsf{Var}\left(E+\bar{B}_{\mathsf{in},i}-\sum_{i=0}^{k_{\mathsf{in}}-1}\bar{A}_{\mathsf{in},i}\cdot S_{\mathsf{in},i}\right)+\mathsf{Var}\left(\sum_{i=0}^{k_{\mathsf{in}}}\sum_{j=0}^{\ell-1}\tilde{A}_{\mathsf{in},i,j}\cdot E_{i,j}\right)
\end{aligned}$$

where

$$\begin{aligned}
&\mathsf{Var}\left(E+\bar{B}_{\mathsf{in},i}-\sum_{i=0}^{k_{\mathsf{in}}-1}\bar{A}_{\mathsf{in},i}\cdot S_{\mathsf{in},i}\right)\\
&=\sigma_{\mathsf{in}}^2+\left(\frac{q^2}{12\mathfrak{B}^{2\ell}}-\frac{1}{12}\right)+\phi_{\mathsf{in}}\left(\frac{q^2}{12\mathfrak{B}^{2\ell}}-\frac{1}{12}\right)\left(\mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)+\mathbb{E}^2\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)\right)+\frac{\phi_{\mathsf{in}}}{4}\mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)\\
&=\sigma_{\mathsf{in}}^2+\left(\frac{q^2-\mathfrak{B}^{2\ell}}{12\mathfrak{B}^{2\ell}}\right)\left(1+\phi_{\mathsf{in}}\left(\mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)+\mathbb{E}^2\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)\right)\right)+\frac{\phi_{\mathsf{in}}}{4}\mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)
\end{aligned}$$

and

$$\mathsf{Var}\left(\sum_{i=0}^{k_{\mathsf{in}}}\sum_{j=0}^{\ell-1}\tilde{A}_{\mathsf{in},i,j}\cdot E_{i,j}\right)=\ell(k_{\mathsf{in}}+1)N\sigma_{\mathsf{KSK}}^2\frac{\mathfrak{B}^2+2}{12}$$

□

A.7 Noise Analysis of the Partial Key External Product

Proof 49 (Noise Eternal Product in Bootstrapping) *Theorem 33 gave us the following noise for an external product:*

$$\begin{aligned}
&\mathsf{Var}\left(M_2\left(E+\bar{B}_{\mathsf{in},i}-\sum_{i=0}^{k_{\mathsf{in}}-1}\bar{A}_{\mathsf{in},i}\cdot S_{\mathsf{in},i}\right)-\sum_{i=0}^{k_{\mathsf{in}}}\sum_{j=0}^{\ell-1}\tilde{A}_{\mathsf{in},i,j}\cdot E_{i,j}\right)\\
&=||M_2||_2^2\cdot\mathsf{Var}\left(E+\bar{B}_{\mathsf{in},i}-\sum_{i=0}^{k_{\mathsf{in}}-1}\bar{A}_{\mathsf{in},i}\cdot S_{\mathsf{in},i}\right)+\mathsf{Var}\left(\sum_{i=0}^{k_{\mathsf{in}}}\sum_{j=0}^{\ell-1}\tilde{A}_{\mathsf{in},i,j}\cdot E_{i,j}\right)
\end{aligned}$$

where

$$\begin{aligned}
&\mathsf{Var}\left(E + \bar{B}_{\mathsf{in},i} - \sum_{i=0}^{k_{\mathsf{in}}-1} \bar{A}_{\mathsf{in},i} \cdot S_{\mathsf{in},i}\right) \\
&= \sigma_{\mathsf{in}}^2 + \left(\frac{q^2}{12\beta^{2\ell}} - \frac{1}{12}\right) + \phi_{\mathsf{in}}\left(\frac{q^2}{12\beta^{2\ell}} - \frac{1}{12}\right)\left(\mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right) + \mathbb{E}^2\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)\right) + \frac{\phi_{\mathsf{in}}}{4}\mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right) \\
&= \sigma_{\mathsf{in}}^2 + \left(\frac{q^2 - \beta^{2\ell}}{12\beta^{2\ell}}\right)\left(1 + \phi_{\mathsf{in}}\left(\mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right) + \mathbb{E}^2\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)\right)\right) + \frac{\phi_{\mathsf{in}}}{4}\mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)
\end{aligned}$$

and

$$\mathsf{Var}\left(\sum_{i=0}^{k_{\mathsf{in}}}\sum_{j=0}^{\ell-1} \tilde{A}_{\mathsf{in},i,j} \cdot E_{i,j}\right) = \ell(k_{\mathsf{in}}+1)N\sigma_{\mathsf{in}}^2\frac{\beta^2+2}{12} .$$

In TFHE, we use binary key to perform the bootstrap. So we have $M \in \{0,1\}$ *which represent a bit of the binary key.* $\mathsf{Var}(M_2) = \frac{1}{4}$ *and* $\mathbb{E}(M_2) = \frac{1}{2}$*. Let focus on the part with the message:*

$$\begin{aligned}
&\mathsf{Var}\left(M_2\left(E + \bar{B}_{\mathsf{in},i} - \sum_{i=0}^{k_{\mathsf{in}}-1} \bar{A}_{\mathsf{in},i} \cdot S_{\mathsf{in},i}\right)\right) \\
&\quad= \Big(\mathsf{Var}(M_2) + \mathbb{E}^2(M_2)\Big)\,\mathsf{Var}\left(E + \bar{B}_{\mathsf{in},i} - \sum_{i=0}^{k_{\mathsf{in}}-1} \bar{A}_{\mathsf{in},i} \cdot S_{\mathsf{in},i}\right) \\
&\qquad+ \mathsf{Var}\left(M_2\right)\mathbb{E}^2\left(E + \bar{B}_{\mathsf{in},i} - \sum_{i=0}^{k_{\mathsf{in}}-1} \bar{A}_{\mathsf{in},i} \cdot S_{\mathsf{in},i}\right) \\
&\quad= \frac{1}{2}\mathsf{Var}\left(E + \bar{B}_{\mathsf{in},i} - \sum_{i=0}^{k_{\mathsf{in}}-1} \bar{A}_{\mathsf{in},i} \cdot S_{\mathsf{in},i}\right) + \frac{1}{4}\mathbb{E}^2\left(E + \bar{B}_{\mathsf{in},i} - \sum_{i=0}^{k_{\mathsf{in}}-1} \bar{A}_{\mathsf{in},i} \cdot S_{\mathsf{in},i}\right) .
\end{aligned}$$

We have

$$\begin{aligned}
\mathbb{E}^2\left(E + \bar{B}_{\mathsf{in},i} - \sum_{i=0}^{k_{\mathsf{in}}-1} \bar{A}_{\mathsf{in},i} \cdot S_{\mathsf{in},i}\right) &= \left(\mathbb{E}(E) + \mathbb{E}(\bar{B}_{\mathsf{in},i}) - \mathbb{E}\left(\sum_{i=0}^{k_{\mathsf{in}}-1} \bar{A}_{\mathsf{in},i} \cdot S_{\mathsf{in},i}\right)\right)^2 \\
&= \left(0 - \frac{1}{2} - \phi_{\mathsf{in}}\mathbb{E}\left(\bar{a}_{\mathsf{in},i}\right)\mathbb{E}\left(s_{\mathsf{in},i}\right)\right)^2 \\
&= \frac{1}{4}\left(-1 + \phi_{\mathsf{in}}\mathbb{E}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)\right)^2 .
\end{aligned}$$

Finally, for each coefficient after one external product in the bootstrapping, we obtain the *following formula for the noise variance*

$$\frac{\sigma_{\mathsf{in}}^2}{2} + \left(\frac{q^2-\beta^{2\ell}}{24\beta^{2\ell}}\right)\left(1+\phi_{\mathsf{in}}\left(\mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)+\mathbb{E}^2\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)\right)\right)+\frac{\phi_{\mathsf{in}}}{8}\mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)$$
$$+\frac{1}{16}\left(-1+\phi_{\mathsf{in}}\mathbb{E}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)^2\right)+\ell(k_{\mathsf{in}}+1)N\sigma_{\mathsf{in}}^2\frac{\beta^2+2}{12}\ .$$

$\square$

A.8 Noise Analysis of the Shrinking Key Switch

Proof 50 (Theorem 36) *The proof of this theorem follows the same footprint as the other key switching proofs presented in this thesis (e.g., Theorem 31). We generalize the proof of this theorem to the GLWE case: the LWE result presented in the theorem follows by taking $k_0 = n_0$, $k_1 = n_1$ and $N = 1$.*

We consider two GLWE secret keys with shared randomness $\vec{S}^{(0)} \prec \vec{S}^{(1)}$ with $\vec{S}^{(0)} = \left(S_0^{(0)},\dots,S_{k_0-1}^{(0)}\right) \in \mathfrak{R}_{q,N}^{k_0}$, $\vec{S}^{(1)} = \left(S_0^{(1)},\dots,S_{k_1-1}^{(1)}\right) \in \mathfrak{R}_{q,N}^{k_1}$, $1 < k_0 < k_1$ and $S_i^{(1)} = S_i^{(0)}$ for all $0 \le i < k_0$. The inputs are

- *A GLWE ciphertext $\mathsf{CT}_{\mathsf{in}} = \left(\vec{A}_{\mathsf{in}}, B_{\mathsf{in}}\right) \in \mathsf{GLWE}_{\vec{S}^{(1)}}(\Delta M) \subseteq \mathfrak{R}_{q,N}^{k_1+1}$, where $B_{\mathsf{in}} = \sum_{i=0}^{k_1-1} A_{\mathsf{in},i}\cdot S_i^{(1)} + \Delta M + E_{\mathsf{in}}$, $A_{\mathsf{in},i} = \sum_{j=0}^{N-1} a_{i,j}\cdot X^j \hookleftarrow \mathcal{U}(\mathfrak{R}_{q,N})$ for all $i \in [\![0,k[\![$ and $E_{\mathsf{in}} = \sum_{j=0}^{N-1} e_j\cdot X^j$, and $e_j \hookleftarrow \mathcal{N}_{\sigma_1^2}$ for all $j \in [\![0,N-1[\![$,*

- *The key switching key $\mathsf{KSK} = (\mathsf{KSK}_0,\dots,\mathsf{KSK}_{k_1-k_0-1})$, where $\mathsf{KSK}_i \in \mathsf{GLev}_{\vec{S}^{(0)}}\left(S_{k_0+i}^{(1)}\right) = \left(\mathsf{GLWE}_{\vec{S}^{(0)}}\left(\frac{q}{\beta}S_{k_0+i}^{(1)}\right),\cdots,\mathsf{GLWE}_{\vec{S}^{(0)}}\left(\frac{q}{\beta^\ell}S_{k_0+i}^{(1)}\right)\right)$ for all $0 \le i < k_1$. We note by $\mathsf{KSK}_{i,j} = (\vec{A}_{i,j}, B_{i,j}) \in \mathsf{GLWE}_{\vec{S}^{(0)}}\left(\frac{q}{\beta^{j+1}}S_{k_0+i}^{(1)}\right)$, for all $0 \le i < k_1$ and for all $0 \le j < \ell$, where $B_{i,j} = \sum_{\tau=0}^{k_0-1} A_{i,j,\tau}\cdot \vec{S}_\tau^{(0)} + \frac{q}{\beta^{j+1}}S_{k_0+i}^{(1)} + E_{\mathsf{ksk},i,j}$, and $E_{\mathsf{ksk},i,j} = \sum_{\tau=0}^{N-1} e_{\mathsf{ksk},i,j,\tau}\cdot X^\tau$ and $e_{\mathsf{ksk},i,j,\tau} \hookleftarrow \mathcal{N}_{\sigma_{\mathsf{ksk}}^2}$.*

The output of this algorithm is

$$(A_{\mathsf{in},0},\dots,A_{\mathsf{in},k_0-1},B_{\mathsf{in}}) - \sum_{i=0}^{k_1-k_0-1}\mathsf{dec}^{(\beta,\ell)}(A_{\mathsf{in},k_0+i})\cdot\mathsf{KSK}_i \in \mathsf{GLWE}_{\vec{S}^{(0)}}(\Delta M) \subseteq \mathfrak{R}_{q,N}^{k_0+1}$$

By definition, in the decomposition algorithm, we have that $\mathsf{dec}^{(\beta,\ell)}(A_{\mathsf{in},i}) = \left(\widetilde{A}_{\mathsf{in},i,0},\cdots,\widetilde{A}_{\mathsf{in},i,\ell-1}\right)$ such that $\widetilde{A}_{\mathsf{in},i} = \sum_{j=0}^{\ell-1}\frac{q}{\beta^{j+1}}\widetilde{A}_{\mathsf{in},i,j}$, for all $k_0 \le i < k_1$.
Let define $\bar{A}_{\mathsf{in},i} = A_{\mathsf{in},i} - \widetilde{A}_{\mathsf{in},i}$, $|\bar{a}_{i,\tau}| = |a_{i,\tau} - \tilde{a}_{i,\tau}| < \frac{q}{2\beta^\ell}$, $\bar{a}_{i,\tau} \in \left[\!\left[\frac{-q}{2\beta^\ell},\frac{q}{2\beta^\ell}\right[\!\right[$ for all $0 \le \tau < N$. So we have that their expectation and variance are respectively $\mathbb{E}(\bar{a}_{i,\tau}) = -\frac{1}{2}$,

$\mathsf{Var}\left(\bar{a}_{i,\tau}\right) = \frac{q^2}{12\beta^{2\ell}} - \frac{1}{12}$, $\mathbb{E}\left(\widetilde{a}_{i,\tau}\right) = -\frac{1}{2}$ *and* $\mathsf{Var}\left(\widetilde{a}_{i,\tau}\right) = \frac{\beta^2-1}{12}$.

Now, by decrypting:

$$\begin{aligned}
&\left\langle (A_{\mathsf{in},0}, \ldots, A_{\mathsf{in},k_0-1}, B_{\mathsf{in}}) - \sum_{i=0}^{k_1-k_0-1} \mathsf{dec}^{(\beta,\ell)}\left(A_{\mathsf{in},k_0+i}\right) \cdot \mathsf{KSK}_i, \left(-\vec{S}^{(0)}, 1\right) \right\rangle \\
&= B_{\mathsf{in}} - \sum_{i=0}^{k_0-1} A_{\mathsf{in},i} \cdot S_i^{(0)} - \sum_{i=0}^{k_1-k_0-1} \sum_{j=0}^{\ell-1} \widetilde{A}_{\mathsf{in},k_0+i,j} \cdot \left\langle \mathsf{KSK}_{i,j}, \left(-\vec{S}^{(0)}, 1\right) \right\rangle \\
&= B_{\mathsf{in}} - \sum_{i=0}^{k_0-1} A_{\mathsf{in},i} \cdot S_i^{(0)} - \sum_{i=0}^{k_1-k_0-1} \sum_{j=0}^{\ell-1} \widetilde{A}_{\mathsf{in},k_0+i,j} \cdot \left(\frac{q}{\beta^{j+1}} S_{k_0+i}^{(1)} + E_{\mathsf{ksk},i,j} \right) \\
&= B_{\mathsf{in}} - \sum_{i=0}^{k_0-1} A_{\mathsf{in},i} \cdot S_i^{(0)} - \sum_{i=0}^{k_1-k_0-1} \widetilde{A}_{\mathsf{in},k_0+i} \cdot S_{k_0+i}^{(1)} - \sum_{i=0}^{k_1-k_0-1} \sum_{j=0}^{\ell-1} \widetilde{A}_{\mathsf{in},k_0+i,j} \cdot E_{\mathsf{ksk},i,j} \\
&= B_{\mathsf{in}} - \sum_{i=0}^{k_0-1} A_{\mathsf{in},i} \cdot S_i^{(0)} - \sum_{i=0}^{k_1-k_0-1} \left(A_{\mathsf{in},k_0+i} - \bar{A}_{\mathsf{in},k_0+i}\right) \cdot S_{k_0+i}^{(1)} - \sum_{i=0}^{k_1-k_0-1} \sum_{j=0}^{\ell-1} \widetilde{A}_{\mathsf{in},k_0+i,j} \cdot E_{\mathsf{ksk},i,j} .
\end{aligned}$$

Since $S_i^{(0)} = S_i^{(1)}$ *for all* $0 \le i < k_0$*, the equation becomes:*

$$\begin{aligned}
&= B_{\mathsf{in}} - \sum_{i=0}^{k_0-1} A_{\mathsf{in},i} \cdot S_i^{(1)} - \sum_{i=0}^{k_1-k_0-1} \left(A_{\mathsf{in},k_0+i} - \bar{A}_{\mathsf{in},k_0+i}\right) \cdot S_{k_0+i}^{(1)} - \sum_{i=0}^{k_1-k_0-1} \sum_{j=0}^{\ell-1} \widetilde{A}_{\mathsf{in},k_0+i,j} \cdot E_{\mathsf{ksk},i,j} \\
&= \Delta M + E_{\mathsf{in}} + \underbrace{\sum_{i=0}^{k_1-k_0-1} \bar{A}_{\mathsf{in},k_0+i} \cdot S_{k_0+i}^{(1)}}_{(I)} \underbrace{- \sum_{i=0}^{k_1-k_0-1} \sum_{j=0}^{\ell-1} \widetilde{A}_{\mathsf{in},k_0+i,j} \cdot E_{\mathsf{ksk},i,j}}_{(II)}
\end{aligned}$$

The w^{th} *coefficient of part* (I) *is equal to*

$$\sum_{i=k_0}^{k_1-1} \left(\sum_{\tau=0}^{w} \bar{a}_{\mathsf{in},i,w-\tau} \cdot s_{i,\tau}^{(1)} - \sum_{\tau=w+1}^{N-1} \bar{a}_{\mathsf{in},i,N+w-\tau} \cdot s_{i,\tau}^{(1)} \right) .$$

The w^{th} *coefficient of part* (II) *is equal to*

$$\sum_{i=0}^{k_1-k_0-1} \sum_{j=0}^{\ell-1} \left(\sum_{\tau=0}^{w} \widetilde{a}_{\mathsf{in},k_0+i,j,w-\tau} \cdot e_{\mathsf{ksk},i,j,\tau} - \sum_{\tau=w+1}^{N-1} \widetilde{a}_{\mathsf{in},k_0+i,j,N+w-\tau} \cdot e_{\mathsf{ksk},i,j,\tau} \right) .$$

We can now isolate the output error for the w^{th} *coefficient and remove the message coef-*

ficient. We obtain that the output error is

$$e'_w = e_{\mathsf{in},w} + \sum_{i=k_0}^{k_1-1}\left(\sum_{\tau=0}^{w} \bar{a}_{\mathsf{in},i,w-\tau}\cdot s^{(1)}_{i,\tau} - \sum_{\tau=w+1}^{N-1} \bar{a}_{\mathsf{in},i,N+w-\tau}\cdot s^{(1)}_{i,\tau}\right)$$
$$- \sum_{i=0}^{k_1-k_0-1}\sum_{j=0}^{\ell-1}\left(\sum_{\tau=0}^{w} \widetilde{a}_{\mathsf{in},k_0+i,j,w-\tau}\cdot e_{\mathsf{ksk},i,j,\tau} - \sum_{\tau=w+1}^{N-1} \widetilde{a}_{\mathsf{in},k_0+i,j,N+w-\tau}\cdot e_{\mathsf{ksk},i,j,\tau}\right) .$$

So the variance is

$$\begin{aligned}\mathsf{Var}(e'_w) &= \mathsf{Var}(e_{\mathsf{in},w}) + (k_1-k_0)N\mathsf{Var}\left(\bar{a}_{\mathsf{in},i,\cdot}.s^{(1)}_{i,\cdot}\right) + (k_1-k_0)\ell N\mathsf{Var}\left(\widetilde{a}_{\mathsf{in},i,j,\cdot}\cdot e_{\mathsf{ksk},i,j,\cdot}\right)\\ &= \sigma^2_{\mathsf{in}} + (k_1-k_0)N\left(\frac{q^2-\beta^{2\ell}}{12\beta^{2\ell}}\right)\left(\mathsf{Var}\left(\vec{S}^{(1)}\right) + \mathbb{E}^2\left(\vec{S}^{(1)}\right)\right)\\ &+ \frac{(k_1-k_0)N}{4}\mathsf{Var}\left(\vec{S}^{(1)}\right) + (k_1-k_0)\ell N\frac{\beta^2+2}{12}\sigma^2_{\mathsf{KSK}} .\end{aligned}$$

□

A.9 Noise Analysis of the GLWE Key Switch with Partial Keys with Shared Randomness

Proof 51 (Theorem 39) *Consider two partial secret keys with shared randomness such that $\vec{S}^{[\phi_{\mathsf{out}}]}_{\mathsf{out}} \prec \vec{S}^{[\phi_{\mathsf{in}}]}_{\mathsf{in}}$. We have $\vec{S}^{[\phi_{\mathsf{out}}]}_{\mathsf{out}} = (S_{\mathsf{out},0},\cdots,S_{\mathsf{out},k_{\mathsf{out}}-1})$, where $S_{\mathsf{out},k_{\mathsf{out}}-1} = \sum_{i=0}^{\phi_{\mathsf{out}}-(k_{\mathsf{out}}-1)N-1} s_{\mathsf{out},k_{\mathsf{out}}-1,i}X^i$ we call $S_{\mathsf{out},k_{\mathsf{out}}-1}$: $\underline{S}$.*

We have $\vec{S}^{[\phi_{\mathsf{in}}]}_{\mathsf{in}} = (S_{\mathsf{in},0},\cdots,S_{\mathsf{in},k_{\mathsf{in}}-1})$ such that for all $j\in[\![0,k_{\mathsf{out}}-1[\![$, $S_{\mathsf{out},j} = S_{\mathsf{in},j}$ and $S_{\mathsf{in},k_{\mathsf{out}}-1} = \underline{S} + \bar{S}$ where $\bar{S} = \sum_{j=\phi_{\mathsf{out}}-(k_{\mathsf{out}}-1)N}^{N-1} s_{\mathsf{in},k_{\mathsf{out}}-1,j}X^j$.

The inputs of a GLWE key switching with partial & shared randomness keys (Algorithm 45) are

- *The input GLWE ciphertext $\mathsf{CT}_{\mathsf{in}} = \left(\vec{A}_{\mathsf{in}}, B_{\mathsf{in}}\right) \in \mathsf{GLWE}_{\vec{S}^{[\phi_{\mathsf{in}}]}_{\mathsf{in}}}(\Delta\cdot M) \subseteq \mathfrak{R}^{k_{\mathsf{in}}+1}_{q,N}$, where $B_{\mathsf{in}} = \sum_{i=0}^{k_{\mathsf{in}}-1} A_{\mathsf{in},i}\cdot S_{\mathsf{in},i} + \Delta\cdot M + E_{\mathsf{in}}$, $A_{\mathsf{in},i} = \sum_{j=0}^{k-1} a_{i,j}\cdot X^j \hookleftarrow \mathcal{U}(\mathfrak{R}_{q,N})$ for all $i\in[\![0,k[\![$ and $E_{\mathsf{in}} = \sum_{j=0}^{k-1} e_j\cdot X^j$, and $e_j \hookleftarrow \mathcal{N}_{\sigma^2_{\mathsf{in}}}$ for all $j\in[\![0,N-1[\![$,*
- *The key switch key $\mathsf{KSK} = (\mathsf{KSK}_{k_{\mathsf{out}-1}}, \mathsf{KSK}_{k_{\mathsf{out}}}\cdots,\mathsf{KSK}_{k_{\mathsf{in}}-1})$, where $\mathsf{KSK}_i \in \mathsf{GLev}_{\vec{S}^{[\phi_{\mathsf{out}}]}_{\mathsf{out}}}(S_{\mathsf{in},i}) = \left(\mathsf{GLWE}_{\vec{S}^{[\phi_{\mathsf{out}}]}_{\mathsf{out}}}\left(\frac{q}{\beta}S_{\mathsf{in},i}\right),\cdots,\mathsf{GLWE}_{\vec{S}^{[\phi_{\mathsf{out}}]}_{\mathsf{out}}}\left(\frac{q}{\beta^\ell}S_{\mathsf{in},i}\right)\right)$ for all $k_{\mathsf{out}}\le i < k_{\mathsf{in}}$, and $\mathsf{KSK}_{k_{\mathsf{out}}-1} \in \mathsf{GLev}_{\vec{S}^{[\phi_{\mathsf{out}}]}_{\mathsf{out}}}\left(\bar{S}\right) = \left(\mathsf{GLWE}_{\vec{S}^{[\phi_{\mathsf{out}}]}_{\mathsf{out}}}\left(\frac{q}{\beta}\bar{S}\right),\cdots,\mathsf{GLWE}_{\vec{S}^{[\phi_{\mathsf{out}}]}_{\mathsf{out}}}\left(\frac{q}{\beta^\ell}\bar{S}\right)\right)$*

We note by $\mathsf{KSK}_{i,j} = (\vec{A}_{i,j}, B_{i,j}) \in \mathsf{GLWE}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}\left(\frac{q}{\beta^{j+1}} S_{\mathsf{in},i}\right)$*, for all* $k_{\mathsf{out}} \leq i < k_{\mathsf{in}}$ *for all* $0 \leq j < \ell$*, where* $B_{i,j} = \sum_{\tau=0}^{k_{\mathsf{out}}-1} A_{i,j,\tau} \cdot S_{\mathsf{out},\tau} + \frac{q}{\beta^{j+1}} S_{\mathsf{in},i} + E_{\mathsf{ksk},i,j}$*, and* $E_{\mathsf{ksk},i,j} = \sum_{\tau=0}^{N-1} e_{\mathsf{ksk},i,j,\tau} \cdot X^{\tau}$ *and* $e_{\mathsf{ksk},i,j,m} \hookleftarrow \mathcal{N}_{\sigma^2_{\mathsf{ksk}}}$.

We note $\mathsf{KSK}_{k_{\mathsf{out}}-1,j} = (\vec{A}_{k_{\mathsf{out}}-1,j}, B_{k_{\mathsf{out}}-1,j}) \in \mathsf{GLWE}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}\left(\frac{q}{\beta^{j+1}} \bar{S}\right)$ *for all* $0 \leq j < \ell$*, where* $B_{k_{\mathsf{out}}-1,j} = \sum_{\tau=0}^{k_{\mathsf{out}}-1} A_{k_{\mathsf{out}}-1,j,\tau} \cdot S_{\mathsf{out},\tau} + \frac{q}{\beta^{j+1}} \bar{S} + E_{\mathsf{ksk},k_{\mathsf{out}}-1,j}$*, and* $E_{\mathsf{ksk},k_{\mathsf{out}}-1,j} = \sum_{\tau=0}^{N-1} e_{\mathsf{ksk},k_{\mathsf{out}}-1,j,\tau} \cdot X^{\tau}$ *and* $e_{\mathsf{ksk},k_{\mathsf{out}}-1,j,m} \hookleftarrow \mathcal{N}_{\sigma^2_{\mathsf{ksk}}}$.

The output of this algorithm is $\mathsf{CT}_{\mathsf{out}} = \left(\vec{A}_{\mathsf{out}}, B_{\mathsf{out}}\right) \in \mathsf{GLWE}_{\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}}(\Delta \cdot M) \subseteq \mathfrak{R}_{q,N}^{k_{\mathsf{out}}+1}$.
By definition, for any polynomial $A_{\mathsf{in},i}$*, we have the decomposition (described in Section 2.3.2),* $\mathsf{dec}^{(\mathfrak{B},\ell)}(A_{\mathsf{in},i}) = \left(\tilde{A}_{\mathsf{in},i,1}, \cdots, \tilde{A}_{\mathsf{in},i,\ell}\right)$ *such that* $\tilde{A}_{\mathsf{in},i} = \sum_{j=0}^{\ell-1} \frac{q}{\mathfrak{B}^{j+1}} \tilde{A}_{\mathsf{in},i,j}$*. Now, we can decrypt:*

$$\begin{aligned}
B_{\mathsf{out}} - \left\langle \vec{A}_{\mathsf{out}}, \vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]} \right\rangle &= \left\langle \left(\vec{A}_{\mathsf{out}}, B_{\mathsf{out}}\right), \left(-\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}, 1\right) \right\rangle \\
&= \Big\langle (A_{\mathsf{in},0}, \cdots, A_{\mathsf{in},k_{\mathsf{out}}-1}, 0 \cdots, 0, B_{\mathsf{in}}) - \mathsf{dec}^{(\mathfrak{B},\ell)}(A_{\mathsf{in},k_{\mathsf{out}}-1}) \mathsf{KSK}_{k_{\mathsf{out}}-1} \\
&\quad - \sum_{i=k_{\mathsf{out}}}^{k_{\mathsf{in}}-1} \mathsf{dec}^{(\mathfrak{B},\ell)}(A_{\mathsf{in},i}) \mathsf{KSK}_i, \left(-\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}, 1\right) \Big\rangle \\
&= B_{\mathsf{in}} - \sum_{i=0}^{k_{\mathsf{out}}-1} A_{\mathsf{in},i} S_{\mathsf{out},i} - \sum_{j=0}^{\ell-1} \tilde{A}_{\mathsf{in},k_{\mathsf{out}}-1,j} \left\langle \mathsf{KSK}_{k_{\mathsf{out}}-1,j}, \left(-\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}, 1\right) \right\rangle \\
&\quad - \sum_{i=k_{\mathsf{out}}}^{k_{\mathsf{in}}-1} \sum_{j=0}^{\ell-1} \tilde{A}_{\mathsf{in},i,j} \left\langle \mathsf{KSK}_{i,j}, \left(-\vec{S}_{\mathsf{out}}^{[\phi_{\mathsf{out}}]}, 1\right) \right\rangle \\
&= B_{\mathsf{in}} - \sum_{i=0}^{k_{\mathsf{out}}-2} A_{\mathsf{in},i} S_{\mathsf{in},i} - A_{\mathsf{in},k_{\mathsf{out}}-1} \underline{S} - \sum_{j=0}^{\ell-1} \tilde{A}_{\mathsf{in},k_{\mathsf{out}}-1,j} \left(\frac{q}{\mathfrak{B}^{j+1}} \bar{S} + E_{\mathsf{ksk},k_{\mathsf{out}}-1,j}\right) \\
&\quad - \sum_{i=k_{\mathsf{out}}}^{k_{\mathsf{in}}-1} \sum_{j=0}^{\ell-1} \tilde{A}_{\mathsf{in},i,j} \left(\frac{q}{\mathfrak{B}^{j+1}} S_{\mathsf{in},i} + E_{\mathsf{ksk},i,j}\right) \\
&= \underbrace{B_{\mathsf{in}} - \sum_{i=0}^{k_{\mathsf{out}}-1} A_{\mathsf{in},i} S_{\mathsf{in},i} - A_{\mathsf{in},k_{\mathsf{out}}-1} \underline{S}}_{(I)} \underbrace{- \tilde{A}_{\mathsf{in},k_{\mathsf{out}}-1} \bar{S} - \sum_{j=0}^{\ell-1} \tilde{A}_{\mathsf{in},k_{\mathsf{out}}-1,j} \cdot E_{\mathsf{ksk},k_{\mathsf{out}}-1,j}}_{(II)} \\
&\quad \underbrace{- \sum_{i=k_{\mathsf{out}}}^{k_{\mathsf{in}}-1} \tilde{A}_{\mathsf{in},i} S_{\mathsf{in},i} - \sum_{i=k_{\mathsf{out}}}^{k_{\mathsf{in}}-1} \sum_{j=0}^{\ell-1} \tilde{A}_{\mathsf{in},i,j} \cdot E_{\mathsf{ksk},i,j}}_{(III)}
\end{aligned}$$

After decrypting, we can split the previous result in three distinct part and analyze the noise provide by each of them. The first part of the result (term (I)) is only composed of the noise present in the B_{in}.
The second part of the result (term (II)) can be seen as a key switching with partial key

(Algorithm 37) from $\bar{S}$ to S_{out}. The proof of noise add by this part follows the proof of Theorem 31.
As for the second part of the result, the third part of the result (term (III)) can be seen as a key switching with partial key (Algorithm 37) from $(S_{\mathsf{in},k_{\mathsf{out}}}, \cdots, S_{\mathsf{in},k_{\mathsf{in}}-1})$ to S_{out}. The proof of noise add by this part follows as well the proof of Theorem 31.
By adding this different noises, we will obtain $\mathsf{Var}(e_{\mathsf{out}}) = \mathsf{Var}(I) + \mathsf{Var}(II) + \mathsf{Var}(III)$ where

$$\begin{aligned}
\mathsf{Var}(I) &= \sigma_{\mathsf{in}}^2 \\
\mathsf{Var}(II) &= (Nk_{\mathsf{out}} - \phi_{\mathsf{out}}) \left(\frac{q^2 - \mathfrak{B}^{2\ell}}{12\mathfrak{B}^{2\ell}}\right) \left(\mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right) + \mathbb{E}^2\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)\right) \\
&\quad + \frac{Nk_{\mathsf{out}} - \phi_{\mathsf{out}}}{4} \mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right) + \ell N \sigma_{\mathsf{ksk}}^2 \frac{\mathfrak{B}^2 + 2}{12} \\
\mathsf{Var}(III) &= (\phi_{\mathsf{in}} - Nk_{\mathsf{out}}) \left(\frac{q^2 - \mathfrak{B}^{2\ell}}{12\mathfrak{B}^{2\ell}}\right) \left(\mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right) + \mathbb{E}^2\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)\right) \\
&\quad + \frac{\phi_{\mathsf{in}} - Nk_{\mathsf{out}}}{4} \mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right) + \ell (k_{\mathsf{in}} - k_{\mathsf{out}} - 1) N \sigma_{\mathsf{ksk}}^2 \frac{\mathfrak{B}^2 + 2}{12} \ .
\end{aligned}$$

To conclude we have

$$\begin{aligned}
\mathsf{Var}(e_{\mathsf{out}}) = \sigma_{\mathsf{in}}^2 &+ (\phi_{\mathsf{in}} - \phi_{\mathsf{out}}) \left(\frac{q^2 - \mathfrak{B}^{2\ell}}{12\mathfrak{B}^{2\ell}}\right) \left(\mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right) + \mathbb{E}^2\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right)\right) \\
&+ \frac{\phi_{\mathsf{in}} - \phi_{\mathsf{out}}}{4} \mathsf{Var}\left(\vec{S}_{\mathsf{in}}^{[\phi_{\mathsf{in}}]}\right) + \ell (k_{\mathsf{in}} - k_{\mathsf{out}}) N \sigma_{\mathsf{ksk}}^2 \frac{\mathfrak{B}^2 + 2}{12} \ .
\end{aligned}$$

□

A.10 Parameters for the Partial Keys with Shared Randomness

In this section, we display the parameters used for the different figures in Chapter 8.

p	Partial Shared Keys	LWE-KS Algorithm	GLWE Parameters		PBS Parameters		LWE-KS Parameters		Metrics	
			Parameter	Value	Parameter	Value	Parameter	Value	Name	Value
1	✗	traditional LWE-to-LWE	n	588	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$	15	$\log_2(\mathfrak{B}_{\mathsf{KS}})$	3	time	6.6480
			$\log_2(\sigma_n)$	-12.66	ℓ_{PBS}	1	ℓ_{KS}	3	size	58.6
			k	5						
			$\log_2(N)$	8						
			$\log_2(\sigma_{k\cdot N})$	-31.07						
1	✓	2 steps *(Alg. 43)*	n	532	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$	15	n_{KS}	782	time	4.9808
			$\log_2(\sigma_n)$	-11.17	ℓ_{PBS}	1	$\log_2\left(\sigma_{n_{\mathsf{KS}}}\right)$	-17.82	size	44.45
			k	5			$\log_2\left(\mathfrak{B}_{\mathsf{KS}_1}\right)$	9		
			$\log_2(N)$	8			ℓ_{KS_1}	1		
			ϕ	1280			$\log_2\left(\mathfrak{B}_{\mathsf{KS}_2}\right)$	2		
			$\log_2\left(\sigma_\phi\right)$	-31.07			ℓ_{KS_2}	4		
1	✓	FFT-based *(Alg. 44)*	n	534	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$	15	k_{in}	3	time	3.8792
			$\log_2(\sigma_n)$	-11.22	ℓ_{PBS}	1	k_{out}	3	size	37.76
			k	5			$\log_2(N_{\mathsf{KS}})$	8		
			$\log_2(N)$	8			$\log_2(\mathfrak{B}_{\mathsf{KS}})$	1		
			ϕ	1280			ℓ_{KS}	9		
			$\log_2\left(\sigma_\phi\right)$	-31.07						
2	✗	traditional LWE-to-LWE	n	668	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$	18	$\log_2(\mathfrak{B}_{\mathsf{KS}})$	4	time	10.185
			$\log_2(\sigma_n)$	-14.79	ℓ_{PBS}	1	ℓ_{KS}	3	size	87.45
			k	6						
			$\log_2(N)$	8						
			$\log_2(\sigma_{k\cdot N})$	-37.88						
2	✓	2 steps *(Alg. 43)*	n	576	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$	18	n_{KS}	896	time	7.7625
			$\log_2(\sigma_n)$	-12.34	ℓ_{PBS}	1	$\log_2\left(\sigma_{n_{\mathsf{KS}}}\right)$	-20.85	size	66.55
			k	6			$\log_2\left(\mathfrak{B}_{\mathsf{KS}_1}\right)$	10		
			$\log_2(N)$	8			ℓ_{KS_1}	1		
			ϕ	1536			$\log_2\left(\mathfrak{B}_{\mathsf{KS}_2}\right)$	2		
			$\log_2\left(\sigma_\phi\right)$	-37.88			ℓ_{KS_2}	5		
2	✓	FFT-based *(Alg. 44)*	n	590	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$	18	k_{in}	1	time	5.9151
			$\log_2(\sigma_n)$	-12.71	ℓ_{PBS}	1	k_{out}	1	size	56.64
			k	6			$\log_2(N_{\mathsf{KS}})$	10		
			$\log_2(N)$	8			$\log_2(\mathfrak{B}_{\mathsf{KS}})$	1		
			ϕ	1536			ℓ_{KS}	11		
			$\log_2\left(\sigma_\phi\right)$	-37.88						
3	✗	traditional LWE-to-LWE	n	720	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$	21	$\log_2(\mathfrak{B}_{\mathsf{KS}})$	4	time	14.704
			$\log_2(\sigma_n)$	-16.17	ℓ_{PBS}	1	ℓ_{KS}	3	size	104.1
			k	4						
			$\log_2(N)$	9						
			$\log_2(\sigma_{k\cdot N})$	-51.49						
3	✓	2 steps *(Alg. 43)*	n	648	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$	18	n_{KS}	944	time	8.4892
			$\log_2(\sigma_n)$	-14.25	ℓ_{PBS}	1	$\log_2\left(\sigma_{n_{\mathsf{KS}}}\right)$	-22.13	size	57.83
			k	3			$\log_2\left(\mathfrak{B}_{\mathsf{KS}_1}\right)$	7		
			$\log_2(N)$	9			ℓ_{KS_1}	2		
			ϕ	1536			$\log_2\left(\mathfrak{B}_{\mathsf{KS}_2}\right)$	2		
			$\log_2\left(\sigma_\phi\right)$	-37.88			ℓ_{KS_2}	6		
3	✓	FFT-based *(Alg. 44)*	n	686	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$	18	k_{in}	1	time	6.0204
			$\log_2(\sigma_n)$	-15.27	ℓ_{PBS}	1	k_{out}	1	size	43.08
			k	3			$\log_2(N_{\mathsf{KS}})$	10		
			$\log_2(N)$	9			$\log_2(\mathfrak{B}_{\mathsf{KS}})$	1		
			ϕ	1536			ℓ_{KS}	13		
			$\log_2\left(\sigma_\phi\right)$	-37.88						
4	✗	traditional LWE-to-LWE	n	788	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$	23	$\log_2(\mathfrak{B}_{\mathsf{KS}})$	4	time	16.265
			$\log_2(\sigma_n)$	-17.98	ℓ_{PBS}	1	ℓ_{KS}	3	size	92.39
			k	2						
			$\log_2(N)$	10						
			$\log_2(\sigma_{k\cdot N})$	-51.49						
4	✓	2 steps *(Alg. 43)*	n	664	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$	22	n_{KS}	1126	time	12.658
			$\log_2(\sigma_n)$	-14.68	ℓ_{PBS}	1	$\log_2\left(\sigma_{n_{\mathsf{KS}}}\right)$	-26.97	size	68.68
			k	2			$\log_2\left(\mathfrak{B}_{\mathsf{KS}_1}\right)$	13		
			$\log_2(N)$	10			ℓ_{KS_1}	1		
			ϕ	2048			$\log_2\left(\mathfrak{B}_{\mathsf{KS}_2}\right)$	2		
			$\log_2\left(\sigma_\phi\right)$	-51.49			ℓ_{KS_2}	6		
4	✓	FFT-based *(Alg. 44)*	n	682	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$	23	k_{in}	3	time	8.7397
			$\log_2(\sigma_n)$	-15.16	ℓ_{PBS}	1	k_{out}	3	size	48.61
			k	2			$\log_2(N_{\mathsf{KS}})$	9		
			$\log_2(N)$	10			$\log_2(\mathfrak{B}_{\mathsf{KS}})$	1		
			ϕ	2048			ℓ_{KS}	14		
			$\log_2\left(\sigma_\phi\right)$	-51.49						

Table A.4: Parameter sets, benchmarks for PBS+LWE-KS and sizes of public material for CJP and two variants based on both partial and shared randomness secret keys. Note that we use $\log_2(\nu) = p$. Sizes are given in MB and times in milliseconds.

p	Partial Shared Keys	LWE-KS Algorithm	GLWE Parameters		PBS Parameters		LWE-KS Parameters		Metrics	
			Parameter	Value	Parameter	Value	Parameter	Value	Name	Value
5	✗	traditional LWE-to-LWE	n $\log_2(\sigma_n)$ k $\log_2(N)$ $\log_2(\sigma_{k\cdot N})$	840 -19.36 1 11 -51.49	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	23 1	$\log_2(\mathfrak{B}_{\mathsf{KS}})$ ℓ_{KS}	3 6	time size	24.964 131.3
5	✓	2 steps *(Alg. 43)*	n $\log_2(\sigma_n)$ k $\log_2(N)$ ϕ $\log_2(\sigma_\phi)$	732 -16.49 1 11 2048 -51.49	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	23 1	n_{KS} $\log_2(\sigma_{n_{\mathsf{KS}}})$ $\log_2(\mathfrak{B}_{\mathsf{KS}_1})$ ℓ_{KS_1} $\log_2(\mathfrak{B}_{\mathsf{KS}_2})$ ℓ_{KS_2}	1171 -28.17 9 2 2 7	time size	18.638 78.62
5	✓	FFT-based *(Alg. 44)*	n $\log_2(\sigma_n)$ k $\log_2(N)$ ϕ $\log_2(\sigma_\phi)$	766 -17.39 1 11 2048 -51.49	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	23 1	k_{in} k_{out} $\log_2(N_{\mathsf{KS}})$ $\log_2(\mathfrak{B}_{\mathsf{KS}})$ ℓ_{KS}	3 3 9 1 15	time size	13.089 48.58
6	✗	traditional LWE-to-LWE	n $\log_2(\sigma_n)$ k $\log_2(N)$ $\log_2(\sigma_{k\cdot N})$	840 -19.36 1 12 -62.00	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	14 2	$\log_2(\mathfrak{B}_{\mathsf{KS}})$ ℓ_{KS}	3 5	time size	67.688 341.4
6	✓	2 steps *(Alg. 43)*	n $\log_2(\sigma_n)$ k $\log_2(N)$ ϕ $\log_2(\sigma_\phi)$	748 -16.91 1 12 2443 -62.00	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	14 2	n_{KS} $\log_2(\sigma_{n_{\mathsf{KS}}})$ $\log_2(\mathfrak{B}_{\mathsf{KS}_1})$ ℓ_{KS_1} $\log_2(\mathfrak{B}_{\mathsf{KS}_2})$ ℓ_{KS_2}	1313 -31.94 16 1 2 8	time size	53.320 224.2
6	✓	FFT-based *(Alg. 44)*	n $\log_2(\sigma_n)$ k $\log_2(N)$ ϕ $\log_2(\sigma_\phi)$	774 -17.61 1 12 2443 -62.00	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	14 2	k_{in} k_{out} $\log_2(N_{\mathsf{KS}})$ $\log_2(\mathfrak{B}_{\mathsf{KS}})$ ℓ_{KS}	1 1 11 1 15	time size	45.647 194.0
7	✗	traditional LWE-to-LWE	n $\log_2(\sigma_n)$ k $\log_2(N)$ $\log_2(\sigma_{k\cdot N})$	896 -20.85 1 13 -62.00	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	15 2	$\log_2(\mathfrak{B}_{\mathsf{KS}})$ ℓ_{KS}	3 6	time size	147.05 784.4
7	✓	2 steps *(Alg. 43)*	n $\log_2(\sigma_n)$ k $\log_2(N)$ ϕ $\log_2(\sigma_\phi)$	776 -17.66 1 13 2443 -62.00	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	15 2	n_{KS} $\log_2(\sigma_{n_{\mathsf{KS}}})$ $\log_2(\mathfrak{B}_{\mathsf{KS}_1})$ ℓ_{KS_1} $\log_2(\mathfrak{B}_{\mathsf{KS}_2})$ ℓ_{KS_2}	1332 -32.45 10 2 1 16	time size	111.14 463.3
7	✓	FFT-based *(Alg. 44)*	n $\log_2(\sigma_n)$ k $\log_2(N)$ ϕ $\log_2(\sigma_\phi)$	818 -18.78 1 13 2443 -62.00	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	14 2	k_{in} k_{out} $\log_2(N_{\mathsf{KS}})$ $\log_2(\mathfrak{B}_{\mathsf{KS}})$ ℓ_{KS}	1 1 11 1 16	time size	98.870 409.5
8	✗	traditional LWE-to-LWE	n $\log_2(\sigma_n)$ k $\log_2(N)$ $\log_2(\sigma_{k\cdot N})$	968 -22.77 1 14 -62.00	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	11 3	$\log_2(\mathfrak{B}_{\mathsf{KS}})$ ℓ_{KS}	3 6	time size	467.25 2179
8	✓	2 steps *(Alg. 43)*	n $\log_2(\sigma_n)$ k $\log_2(N)$ ϕ $\log_2(\sigma_\phi)$	816 -18.72 1 14 2443 -62.00	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	11 3	n_{KS} $\log_2(\sigma_{n_{\mathsf{KS}}})$ $\log_2(\mathfrak{B}_{\mathsf{KS}_1})$ ℓ_{KS_1} $\log_2(\mathfrak{B}_{\mathsf{KS}_2})$ ℓ_{KS_2}	1359 -33.17 9 2 1 17	time size	351.64 1304
8	✓	FFT-based *(Alg. 44)*	n $\log_2(\sigma_n)$ k $\log_2(N)$ ϕ $\log_2(\sigma_\phi)$	854 -19.73 1 14 2443 -62.00	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	11 3	k_{in} k_{out} $\log_2(N_{\mathsf{KS}})$ $\log_2(\mathfrak{B}_{\mathsf{KS}})$ ℓ_{KS}	1 1 11 1 18	time size	326.44 1282

Table A.5: Parameter sets, benchmarks for PBS+LWE-KS and sizes of public material for CJP and two variants based on both partial and shared randomness secret keys. Note that we use $\log_2(\nu) = p$. Sizes are given in MB and times in milliseconds.

p	Partial Shared Keys	LWE-KS Algorithm	GLWE Parameters		PBS Parameters		LWE-KS Parameters		Metrics	
			Parameter	Value	Parameter	Value	Parameter	Value	Name	Value
9	✗	traditional LWE-to-LWE	n $\log_2(\sigma_n)$ k $\log_2(N)$ $\log_2(\sigma_{k \cdot N})$	1024 -24.26 1 15 -62.00	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	9 4	$\log_2(\mathfrak{B}_{\mathsf{KS}})$ ℓ_{KS}	3 7	time size	1383.8 5890
9	✓	2 steps *(Alg. 43)*	n $\log_2(\sigma_n)$ k $\log_2(N)$ ϕ $\log_2(\sigma_\phi)$	860 -19.89 1 15 2443 -62.00	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	8 4	n_{KS} $\log_2(\sigma_{n_{\mathsf{KS}}})$ $\log_2(\mathfrak{B}_{\mathsf{KS}_1})$ ℓ_{KS_1} $\log_2(\mathfrak{B}_{\mathsf{KS}_2})$ ℓ_{KS_2}	1388 -33.94 10 2 1 18	time size	1148.1 3525
9	✓	FFT-based *(Alg. 44)*	n $\log_2(\sigma_n)$ k $\log_2(N)$ ϕ $\log_2(\sigma_\phi)$	902 -21.01 1 15 2443 -62.00	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	8 4	k_{in} k_{out} $\log_2(N_{\mathsf{KS}})$ $\log_2(\mathfrak{B}_{\mathsf{KS}})$ ℓ_{KS}	1 1 11 1 18	time size	1058.1 3609
10	✗	traditional LWE-to-LWE	n $\log_2(\sigma_n)$ k $\log_2(N)$ $\log_2(\sigma_{k \cdot N})$	1096 -26.17 1 16 -62.00	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	6 6	$\log_2(\mathfrak{B}_{\mathsf{KS}})$ ℓ_{KS}	2 12	time size	4794.4 19730
10	✓	2 steps *(Alg. 43)*	n $\log_2(\sigma_n)$ k $\log_2(N)$ ϕ $\log_2(\sigma_\phi)$	904 -21.06 1 16 2443 -62.00	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	6 6	n_{KS} $\log_2(\sigma_{n_{\mathsf{KS}}})$ $\log_2(\mathfrak{B}_{\mathsf{KS}_1})$ ℓ_{KS_1} $\log_2(\mathfrak{B}_{\mathsf{KS}_2})$ ℓ_{KS_2}	1417 -34.71 11 2 1 19	time size	3721.0 10940
10	✓	FFT-based *(Alg. 44)*	n $\log_2(\sigma_n)$ k $\log_2(N)$ ϕ $\log_2(\sigma_\phi)$	938 -21.97 1 16 2443 -62.00	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	6 6	k_{in} k_{out} $\log_2(N_{\mathsf{KS}})$ $\log_2(\mathfrak{B}_{\mathsf{KS}})$ ℓ_{KS}	3 3 9 1 20	time size	3628.1 11260
11	✗	traditional LWE-to-LWE	n $\log_2(\sigma_n)$ k $\log_2(N)$ $\log_2(\sigma_{k \cdot N})$	1132 -27.13 1 17 -62.00	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	2 20	$\log_2(\mathfrak{B}_{\mathsf{KS}})$ ℓ_{KS}	2 13	time size	37795 105300
11	✓	2 steps *(Alg. 43)*	n $\log_2(\sigma_n)$ k $\log_2(N)$ ϕ $\log_2(\sigma_\phi)$	984 -23.19 1 17 2443 -62.00	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	3 12	n_{KS} $\log_2(\sigma_{n_{\mathsf{KS}}})$ $\log_2(\mathfrak{B}_{\mathsf{KS}_1})$ ℓ_{KS_1} $\log_2(\mathfrak{B}_{\mathsf{KS}_2})$ ℓ_{KS_2}	1471 -36.15 11 2 1 21	time size	18237 47330
11	✓	FFT-based *(Alg. 44)*	n $\log_2(\sigma_n)$ k $\log_2(N)$ ϕ $\log_2(\sigma_\phi)$	1018 -24.10 1 17 2443 -62.00	$\log_2(\mathfrak{B}_{\mathsf{PBS}})$ ℓ_{PBS}	3 13	k_{in} k_{out} $\log_2(N_{\mathsf{KS}})$ $\log_2(\mathfrak{B}_{\mathsf{KS}})$ ℓ_{KS}	3 3 9 1 22	time size	19224 52940

Table A.6: Parameter sets, benchmarks for PBS+LWE-KS and sizes of public material for CJP and two variants based on both partial and shared randomness secret keys. Note that we use $\log_2(\nu) = p$. Sizes are given in MB and times in milliseconds.

Titre : Construction de nouveaux outils pour un chiffrement homomorphe efficace

Mot clés : chiffrement homomorphe, cloud computing, learning with errors, cryptologie, optimisation, TFHE

Résumé : Dans notre vie de tous les jours, nous produisons une multitude de données à chaque fois que nous accédons à un service en ligne. Certaines sont partagées volontairement et d'autres à contrecœur. Ces données sont collectées et analysées en clair, ce qui menace la vie privée de l'utilisateur et empêche la collaboration entre entités travaillant sur des données sensibles. Le chiffrement complètement homomorphe (*Fully Homomorphic Encryption*) apporte une lueur d'espoir en permettant d'effectuer des calculs sur des données chiffrées ce qui permet de les analyser et de les exploiter sans jamais y accéder en clair. Cette thèse se focalise sur TFHE, un récent schéma complètement homomorphe capable de réaliser un *bootstrapping* en un temps record. Dans celle-ci, nous introduisons une méthode d'optimisation pour sélectionner les degrés de liberté inhérents aux calculs homomorphiques permettant aux profanes d'utiliser TFHE. Nous détaillons une multitude de nouveaux algorithmes homomorphes qui améliorent l'efficacité de TFHE et réduisent voire éliminent les restrictions d'algorithmes connus. Une implémentation efficace de ceux-ci est d'ores et déjà en accès libre.

Title: Constructing new tools for efficient homomorphic encryption

Keywords: homomorphic encryption, cloud computing, learning with errors, cryptology, optimization, TFHE

Abstract: In our everyday life, we leave a trail of data whenever we access online services. Some are given voluntarily and others reluctantly. Those data are collected and analyzed in the clear which leads to major threats on the user's privacy and prevents collaborations between entities working on sensitive data. In this context, *Fully Homomorphic Encryption* brings a new hope by enabling computation over encrypted data, which removes the need to access data in the clear to analyze and exploit it. This thesis focuses on TFHE, a recent fully homomorphic encryption scheme able to compute a *bootstrapping* in record time. We introduce an optimization framework to set the degrees of freedom inherent to homomorphic computations which gives non-experts the ability to use it (more) easily. We describe a plethora of new FHE algorithms which improve significantly the state of the art and limit, (if not remove) existing restrictions. Efficient open source implementations are already accessible.