TFHE: Fast Fully Homomorphic Encryption over the Torus

Ilaria Chillotti[1], Nicolas Gama[2,1], Mariya Georgieva[3], and Malika Izabachène[4]

[1] Laboratoire de Mathématiques de Versailles, UVSQ, CNRS, Université Paris-Saclay, 78035 Versailles, France
[2] Inpher, Lausanne, Switzerland
[3] Gemalto, 6 rue de la Verrerie 92190, Meudon, France
[4] CEA LIST, Point Courrier 172, 91191 Gif-sur-Yvette Cedex, France

Abstract. This work describes a fast fully homomorphic encryption scheme over the torus (TFHE), that revisits, generalizes and improves the fully homomorphic encryption (FHE) based on GSW and its ring variants. The simplest FHE schemes consist in bootstrapped binary gates. In this gate bootstrapping mode, we show that the scheme FHEW of [24] can be expressed only in terms of external product between a GSW and a LWE ciphertext. As a consequence of this result and of other optimizations, we decrease the running time of their bootstrapping from $690ms$ to $13ms$ single core, using 16MB bootstrapping key instead of 1GB, and preserving the security parameter. In leveled homomorphic mode, we propose two methods to manipulate packed data, in order to decrease the ciphertext expansion and to optimize the evaluation of look-up tables and arbitrary functions in RingGSW based homomorphic schemes. We also extend the automata logic, introduced in [26], to the efficient leveled evaluation of weighted automata, and present a new homomorphic counter called TBSR, that supports all the elementary operations that occur in a multiplication. These improvements speed-up the evaluation of most arithmetic functions in a packed leveled mode, with a noise overhead that remains additive. We finally present a new circuit bootstrapping that converts LWE ciphertexts into low-noise RingGSW ciphertexts in just $137ms$, which makes the leveled mode of TFHE composable, and which is fast enough to speed-up arithmetic functions, compared to the gate bootstrapping approach.
Finally, we provide an alternative practical analysis of LWE based schemes, which directly relates the security parameter to the error rate of LWE and the entropy of the LWE secret key, and we propose concrete parameter sets and timing comparison for all our constructions
Keywords: Fully Homomorphic Encryption, Bootstrapping, Lattices, LWE, GSW, boolean circuit, deterministic automata

1 Introduction

This paper is the complete and extended version of the two papers [17] and [19] published by the same authors at Asiacrypt 2016 and Asiacrypt 2017, respectively. It unifies the work presented in these papers, completes the proofs and adds some further results.

Since Gentry introduced in 2009 [28] the concept of bootstrapping, and proved that fully homomorphic encryption was achievable in polynomial time, many constructions have appeared, involving new mathematical and algorithmic concepts, improving efficiency and memory requirements. Nowadays, the most promising constructions [41, 8, 29] rely on two lattice-based problems: *approximate-GCD*, presented by Howgrave-Graham in 2001 [33], and *Learning With Errors (*LWE*)*, presented by Regev in 2005 [40] and its ring variants [37].

The literature distinguishes two families of homomorphic encryption schemes: leveved (LHE) and fully (FHE) homomorphic encryption. Informally, in LHE, for each function, there exist parameters that can homomorphically evaluate it. In FHE, a single parameter set allows to evaluate any function. With this (generalized) definition, FHE can be viewed as a particular case of LHE.
For a given security parameter, and also a class of functions to evaluate in the LHE case, the quality of a homomorphic scheme is measured in terms of expressivity of its elementary operations, key size, running time per elementary operation, and ciphertext overhead. In this work, we improve them all, both from a theoretical point of view, by abstracting the GSW construction, and by extending homomorphic operations to new computational models, coming from weighted automata theory, and also from a practical point of view, by providing complete algorithms and concrete parameters, as well as an open-source implementation.

Most naive homomorphic schemes use Gaussian noise to mask the plaintext, and the noise variance grows after each operation, until it reaches critical levels. As per [28], bootstrapping a ciphertext consists in homomorphically decrypting it, using a homomorphic encryption of its own secret key. In the end, we get an encryption of the same plaintext, but which noise only depends on the decryption circuit. The output noise is independent from the input noise, which confers to bootstrapping the unique ability to reduce the noise of a ciphertext. The simplest FHE schemes contain a single elementary operation: a NAND gate followed by a bootstrapping. Any polynomial time function can indeed be systematically written as a polynomial number of NAND gates, which can be evaluated homomorphically one by one using this bootstrapped NAND operation. Between 2009 and 2015, the running time and memory requirements to achieve this bootstrapped NAND gate has decreased across multiple generations of constructions, for instance a BGV-based [8] bootstrapping in the Helib library[31] and a GSW-based [29] bootstrapping in the FHEW library [24]. The last one obtains one bootstrapped NAND gate in 0.69ms single core, using a 1GB bootstrapping key, and with a ciphertext overhead of 10000 for at least 100 bits of security.

In this work, we present a gate bootstrapping algorithm, implemented in the TFHE library[20], that decreases these requirements to 13ms single core, using a 16MB bootstrapping key, with the same ciphertext overhead and a higher security parameter.

Despite these optimizations, bootstrapped bit operations are still about one billion times slower than their plaintext equivalents. Other trade-offs have been proposed, where elementary homomorphic operations consist in vectorial arith-

metic, which covers a large number of real life applications, in statistics or physics. The possibility to batch these operations in a SIMD manner compensates for the slow homomorphic operations, and provides a consequent apparent speed-up per element. Also, packing multiple plaintext bits on the same ciphertext asymptotically reduces the ciphertext expansion to a constant. The efficiency of these schemes crucially relies on the fact that plaintext computations are expressed on a ring structure, where addition is invertible, and also that it has enough parallelism to fill all the computation slots. This model doesn't apply for highly non-linear computations involving comparisons, tropical algebra, optimization on graphs, etc...

To this end, we propose new packed elementary LHE operations that are more generic: like for instance, arbitrary multibit lookup tables, or arbitrary deterministic classical or weighted automata. We show how to use the computation slots at their maximal capacity, even if the function itself is not SIMD, or has very few bits of output. In FHE mode, we also provide a circuit bootstrapping procedure, that takes a LWE ciphertext as input, reduces its noise, and converts it back to an GSW ciphertext suitable for subsequent packed operations. This allows us for instance to evaluate an arbitrary function from $\{0,1\}^{10} \to \{0,1\}$ in $340\mu s$ and to bootstrap the output $137ms$, thus improving upon all alternatives that output a bootstrapped GSW ciphertext.

Finally, we give a abstract view of our constructions (by simply changing a fundamental building block called the phase, we can obtain FHE cryptosystems based not only on LWE, RingLWE, Module-LWE, but also scale-invariant versions based on the approx-GCD, or on the NTRU function. We show how to instantiate a canonical version, based on RingLWE, directly from the two user parameters: security parameters, and depth of circuit in the case of LHE. We reduced the number of parameters to avoid dependency loops between them, and we keep the remaining ones as intrinsic as possible, to ease their setup: for example, scale invariant versions of lattice problems are expressed on the torus, Gaussian noises are represented by their standard deviation or their variance, the size of the key is measured in bits. In the last sections, we explain how to calculate the parameters using bounds coming from state-of-the-art cryptanalysis, and we provide the concrete values that we implemented in the open source library TFHE [20]. We provide the results and experimental running time at the end on the paper.

2 Background

In this section, we introduce some fundamental concepts that are used in the rest of the paper. In particular, we describe and revisit the LWE problem [40] before giving its generalization in section 3. We start by fixing some notations.

Notations. In the rest of the paper, we denote the security parameter as λ. We denote as $\mathbb{B}$ the set $\{0,1\}$ without any structure and by $\mathbb{T}$ the real Torus $\mathbb{R}/\mathbb{Z}$, the set of real numbers modulo 1. We denote by $\mathbb{Z}_N[X]$ the ring of polynomials

$\mathbb{Z}[X]/(X^N+1)$. $\mathbb{T}_N[X]$ denotes $\mathbb{R}[X]/(X^N+1) \mod 1$ and $\mathbb{B}_N[X]$ denotes the polynomials in $\mathbb{Z}_N[X]$ with binary coefficients. We denote by E^p the set of vectors of size p with entries in E and by $\mathcal{M}_{p,q}(E)$ the set of $p \times q$-size matrices with elements in E.

Definition 2.1 (R-module). *Let $(R, +, \times)$ be a commutative ring. We say that a set M is a R-module when $(M, +)$ is an abelian group, and when there exists an external operation $\cdot$ (product) which is bi-distributive and homogeneous. Namely, $\forall r, s \in R$ and $x, y \in M$, $1_R \cdot x = x$, $(r+s)\cdot x = r\cdot x + s\cdot x$, $r\cdot(x+y) = r\cdot x + r\cdot y$, and $(r \times s)\cdot x = r\cdot(s\cdot x)$.*

Remark 1. A R-module M shares many arithmetic operations and constructions with vector spaces: vectors M^p or matrices $\mathcal{M}_{p,q}(M)$ are also R-modules, and their left dot product with a vector in R^p or left matrix product in $\mathcal{M}_{k,p}(R)$ are both well defined.

By construction, any abelian group is a $\mathbb{Z}$-module by iteration of its own law. In this paper we largely use the torus $\mathbb{T}$, which is a $\mathbb{Z}$-module. It is not a ring since the mod 1 projection is not compatible with the real product. As instance, the product $0 \times \frac{1}{2}$, where 0 and $\frac{1}{2}$ are seen as elements of $\mathbb{T}$, is undefined in $\mathbb{T}$. Instead, the external product $\cdot$ between an element of $\mathbb{Z}$ and an element in $\mathbb{T}$ is correctly defined ($0 \cdot \frac{1}{2}$, where $0 \in \mathbb{Z}$ and $\frac{1}{2} \in \mathbb{T}$, is equal to $0 \in \mathbb{T}$).

More importantly, we recall that for all positive integers N and k, $(\mathbb{T}_N[X]^k, +, \cdot)$ is a $\mathbb{Z}_N[X]$-module.

2.1 Probability distributions

Most FHE schemes hide the plaintext with Gaussian noise. In this paper, we always quantify this noise via its standard deviation or its variance. The variance of a Gaussian distribution is equal to its average square norm divided by the dimension, so working with it leads to propagation formula that are natural. Most importantly, it avoids the additional $\sqrt{2\pi}$ factors (related to the noise parameter), which have often been a source of confusion in concrete implementations.

Gaussian Distributions Let $k \geq 1$ and $\sigma \in \mathbb{R}^+$. For all $\boldsymbol{x}, \boldsymbol{c} \in \mathbb{R}^k$, we denote by $\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{x}) = \exp(-\|\boldsymbol{x}-\boldsymbol{c}\|^2/2\sigma^2)$ the Gaussian function of center $\boldsymbol{c}$ and standard deviation σ. If $\boldsymbol{c}$ is omitted, then it is implicitly set to $\boldsymbol{0}$. Let S be a subset of $\mathbb{R}^k$, then $\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{S})$ denotes $\sum_{x\in S} \rho_{\sigma,\boldsymbol{c}}(\boldsymbol{x})$, if S discrete, or $\int_{x\in S} \rho_{\sigma,\boldsymbol{c}}(\boldsymbol{x}) \cdot d\boldsymbol{x}$, if S is continuous.

For all closed (continuous or discrete) additive subgroup $M \subseteq \mathbb{R}^k$, $\rho_{\sigma,\boldsymbol{c}}(M)$ is finite, and defines a *(restricted) Gaussian Distribution* $\mathcal{D}_{M,\sigma,\boldsymbol{c}}$ of standard deviation σ and center $\boldsymbol{c}$ over M, with the density function $\mathcal{D}_{M,\sigma,\boldsymbol{c}}(\boldsymbol{x}) = \rho_{\sigma,\boldsymbol{c}}(\boldsymbol{x})/\rho_{\sigma,\boldsymbol{c}}(M)$. Let L be a discrete subgroup of M, then the *Modular Gaussian distribution* $\mathcal{D}_{M/L,\sigma,\boldsymbol{c}}$ over M/L exists and is defined by the density $\mathcal{D}_{M/L,\sigma,\boldsymbol{c}}(\boldsymbol{x}) = \mathcal{D}_{M,\sigma,\boldsymbol{c}}(\boldsymbol{x}+L)$.

Subgaussian Distributions A distribution $\mathcal{X}$ over $\mathbb{R}$ is σ-subgaussian if and only if it satisfies the Laplace-transformation bound. Namely $\forall t \in \mathbb{R}$, the expectation verifies $\mathbb{E}(\exp(tX)) \leq \exp(\sigma^2 t^2/2)$. By Markov's inequality, this implies that the tails of X are bounded by the Gaussian function of standard deviation σ: $\forall x > 0, \mathbb{P}(|X| \geq x) \leq 2\exp(-x^2/2\sigma^2)$. As an example, the Gaussian distribution of standard deviation σ (i.e. parameter $\sqrt{2\pi}\sigma$), the equi-distribution on $\{-\sigma, \sigma\}$, and the uniform distribution over $[-\sqrt{3}\sigma, \sqrt{3}\sigma]$, which all have standard deviation σ, are σ-subgaussian[5]. If $\mathcal{X}$ and $\mathcal{X}'$ are two independent σ and σ'-subgaussian variables, then for all $\alpha, \beta \in \mathbb{R}$, $\alpha\mathcal{X} + \beta\mathcal{X}'$ is $\sqrt{\alpha^2\sigma^2 + \beta^2\sigma'^2}$-subgaussian.

Concentrated distribution on the Torus In general, distributions over the torus don't have expectation nor variance: for instance, it would be impossible to define the expectation of the uniform distribution over $\mathbb{T}$. However, when the support of the distribution is concentrated on a small interval, it is still possible to uniquely define these notions. A distribution $\mathcal{X}$ on the torus is *concentrated* if and only if its support is included in a ball of radius $\frac{1}{4}$ of $\mathbb{T}$, except for negligible probability. In this case, we define the *variance* $\mathsf{Var}(\mathcal{X})$ and the expectation $\mathbb{E}(\mathcal{X})$ of $\mathcal{X}$ as respectively $\mathsf{Var}(\mathcal{X}) = \min_{\bar{x}\in\mathbb{T}} \sum p(x)|x - \bar{x}|^2$ and $\mathbb{E}(\mathcal{X})$ as the position $\bar{x} \in \mathbb{T}$ which minimizes this expression. This definition of expectation by an optimization formula yields the same result as if we lift the distribution over any real interval of length $< \frac{1}{2}$, and compute its real expectation modulo 1. By extension, we say that a distribution $\mathcal{X}'$ over $\mathbb{T}^n$ or $\mathbb{T}_N[X]^k$ is concentrated if and only if each coefficient has an independent concentrated distribution on the torus. Then the expectation $\mathbb{E}(\mathcal{X}')$ is the vector of expectations of each coefficient, and $\mathsf{Var}(\mathcal{X}')$ denotes the maximum of each coefficient's variance.

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

Fact 2.2. Let $\mathcal{X}_1, \mathcal{X}_2$ be two independent concentrated distributions on either $\mathbb{T}, \mathbb{T}^n$ or $\mathbb{T}_N[X]^k$, and $e_1, e_2 \in \mathbb{Z}$ such that $\mathcal{X} = e_1 \cdot \mathcal{X}_1 + e_2 \cdot \mathcal{X}_2$ remains concentrated, then $\mathbb{E}(\mathcal{X}) = e_1 \cdot \mathbb{E}(\mathcal{X}_1) + e_2 \cdot \mathbb{E}(\mathcal{X}_2)$ and $\mathsf{Var}(\mathcal{X}) \leq e_1^2 \cdot \mathsf{Var}(\mathcal{X}_1) + e_2^2 \cdot \mathsf{Var}(\mathcal{X}_2)$.

Also, subgaussian distributions with small enough parameters are necessarily concentrated:

Fact 2.3. Every distribution $\mathcal{X}$ on either $\mathbb{T}, \mathbb{T}^n$ or $\mathbb{T}_N[X]^k$ where each coefficient is σ-subgaussian where $\sigma \leq 1/\sqrt{32\ln(2)(\lambda+1)}$ is a concentrated distribution: a fraction $1 - 2^{-\lambda}$ of its mass is in the interval $[-\frac{1}{4}, \frac{1}{4}]$.

2.2 Distance and Norms

We denote as $\|\cdot\|_p$ and $\|\cdot\|_\infty$ the standard norms for scalars and vectors over the real field or over the integers. By extension, the norms $\|P(X)\|_p$ and $\|P(X)\|_\infty$

[5] For the first two distributions, it is tight, but the uniform distribution over $[-\sqrt{3}\sigma, \sqrt{3}\sigma]$ is even 0.78σ-subgaussian

of a real or integer polynomial P are the norms of its coefficient vector. If P is a polynomial $\mod X^N - 1$, we take the norm of its unique representative of degree $\leq N-1$.

If $\boldsymbol{x}$ is an vector in $\mathbb{T}^k$, we note $\|\boldsymbol{x}\|_p = \min_{\boldsymbol{u} \in \boldsymbol{x} + \mathbb{Z}^k}(\|u\|_p)$ is the p-norm of the representative of $\boldsymbol{x}$ with all coefficients in $]-\frac{1}{2}, \frac{1}{2}]$. It satisfies the separation and the triangular inequalities, but it is not a norm because it lacks homogeneity[6], and $\mathbb{T}^k$ is not a vector space either. Instead, it is sub-homogeneous, i.e. it satisfies the property $\|m \cdot \boldsymbol{x}\|_p \leq |m| \|\boldsymbol{x}\|_p$, $\forall m \in \mathbb{Z}$. By extension, we define $\|P\|_p$ for a polynomial $P \in \mathbb{T}_N[X]$ as the p-norm of its unique representative in $\mathbb{R}[X]$ of degree $\leq N-1$ and with coefficients in $]-\frac{1}{2}, \frac{1}{2}]$.

Definition 2.4 (Infinity norm over $\mathcal{M}_{p,q}(\mathbb{T}_N[X])$). *Let $A \in \mathcal{M}_{p,q}(\mathbb{T}_N[X])$. We define the infinity norm of A as*

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

2.3 Learning With Errors problem revisited

The Learning With Errors (LWE) problem was introduced by Regev in 2005 [40]. The Ring variant of the same problem, called RingLWE, was introduced by Lyubashevsky, Peikert and Regev in 2010 [37]. Both variants are nowadays extensively used for the constructions of lattice-based Homomorphic Encryption schemes. In the original definition [40], a LWE sample has its right hand side on the torus and it is defined using continuous Gaussian distributions. Here, we work entirely on the real torus, employing the same formalism as the Scale Invariant LWE scheme in [16], or LWE scale-invariant normal form in [18]. Without loss of generality, we refer to it as LWE.

Definition 2.5 ((Scale-Invariant) LWE (adapted from [16]). *Let $n \geq 1$ be an integer, $\boldsymbol{s}$ be in $\mathbb{Z}^n$ and ξ a distribution over $\mathbb{R}$. We define $\mathsf{LWE}_{\boldsymbol{s},\xi}$ as the distribution over $\mathbb{T}^n \times \mathbb{T}$ obtained by sampling a pair $(\boldsymbol{a}, b)$, where the left member $\boldsymbol{a} \in \mathbb{T}^n$ is chosen uniformly random and the right member $b = \boldsymbol{a} \cdot \boldsymbol{s} + e$. The error e is a sample from the distribution ξ. Let $\mathcal{S}$ be a distribution over $\mathbb{Z}^n$. We can define the two following problems.*

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

Both the LWE search or decision problems are reducible to each other, and their average case is asymptotically as hard as worst-case lattice problems [40].

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

In practice, both problems are also intractable, and their hardness increases with the the entropy of the key set $\mathcal{S}$ (*i.e.* n if keys are binary) and $\alpha \in]0, \eta_\varepsilon(\mathbb{Z})[$.

Let $s \in \mathcal{S}$ be a fixed secret, we call *phase* the secret linear function φ_s from $\mathbb{T}^n \times \mathbb{T}$ to $\mathbb{T}$ defined as $\varphi_s(\boldsymbol{a}, b) = b - s \cdot \boldsymbol{a}$. In this case, if we compute the phase of a sample from the $\mathsf{LWE}_{s,\xi}$ distribution, the result is the error e, which is very small. In other words, the distribution $\mathsf{LWE}_{s,\xi}$ samples approximations of the kernel of the phase. We also remark that with this definition of phase, for all $\mu \in \mathbb{T}$, the trivial element $(\mathbf{0}, \mu)$ is a preimage of μ by φ_s.

This allows to reconstruct Regev's encryption scheme [40]. Given a discrete message space $\mathcal{M} \in \mathbb{T}$ (for instance $\{0, \frac{1}{2}\}$), a message $\mu \in \mathcal{M}$ is encrypted as an approximation of a random preimage $\varphi_s^{-1}(\mu)$. Concretely, we sum the *trivial* element $(\mathbf{0}, \mu)$ to a $\mathsf{LWE}_{\boldsymbol{s},\xi}$ sample. The semantic security of the scheme is by definition equivalent to the LWE decisional problem. To decrypt a sample $\boldsymbol{c} = (\boldsymbol{a}, b)$, we compute the phase $\varphi_s(\boldsymbol{c})$, which gives μ plus the error, and we round it to the nearest element in $\mathcal{M}$. Decryption is correct with overwhelming probability $1 - 2^{-p}$ provided that the Gaussian parameter α is $O(R/\sqrt{p})$ where R is the packing radius of $\mathcal{M}$.

Regev's encryption scheme has also an asymmetric variant, where the public key is a list of random $\mathsf{LWE}_{\boldsymbol{s},\xi}$ samples. Then, to encrypt a message $\mu \in \mathcal{M}$, one chooses a small random subset of the elements of the public key, and sums it to the trivial LWE sample $(\mathbf{0}, \mu)$ of μ.

3 Homomorphic arithmetic on the torus

In this section we describe the generalizations of the LWE problem and of the GSW construction over the real torus $\mathbb{T}$.

3.1 TLWE

In this section, we present a generalization of the LWE problem, following the footprint of [8] (that defined the General LWE problem) and [29]. We call this generalization TLWE.

In the previous example, the phase was derived from the settings of the LWE cryptosystem. In TLWE, the phase becomes the central building block. All other notions are deduced from the algebraic properties of this linear function: message space, ciphertext space, encryption, decryption. In particular, this abstraction allows to unify every scale-invariant FHE scheme, based not only on LWE, $\mathsf{RingLWE}$, Module-LWE [34], but also on other problems like Approx-GCD or NTRU.

Definition 3.1 (Abstract TLWE problems). *Let I be an ideal of $\mathbb{Z}[X]$, we call $\mathfrak{R} = \mathbb{Z}[X]/I$ and $\mathbb{T}_I[X] = \mathbb{T}[X]/I$. A* phase function *is a lipschitzian morphism (*i.e. *linear map) from a $\mathfrak{R}$-module M to $\mathbb{T}_I[X]$. The general TLWE problem is*

parametrized by an error distribution ξ on M, and a family $(\varphi_s)_{s\in\mathcal{S}}$ of phase functions, indexed by a secret s. The homogeneous TLWE distribution *for the secret s is $\mathcal{U}_{\ker(\varphi_s)}+\xi$ (sum of the uniform distribution over $\ker(\varphi_s)$ and an error from ξ).* TLWE *is λ-secure if the following two problems cannot be solved in less than 2^λ bit operations, or with advantage $2^{-\lambda}$ by any PPT[7] adversary:*

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

If we instanciate this definition with $I = (X + 1)$, then we get $\mathfrak{R} = \mathbb{Z}$ and $\mathbb{T}_I[X] = \mathbb{T}$, and obtain scalar schemes. Setting $M = \mathbb{T}^{n+1}$ and the phase as $\varphi_s(\boldsymbol{a}, b) = b - \boldsymbol{sa}$, we retrieve the previous scale-invariant LWE. By choosing instead $M = (\mathbb{Z}/q\mathbb{Z})^{n+1}$ with phase $\varphi_s(a, b) = (b - sa)/q$ and discrete Gaussian error, we retrieve the well known LWE mod q. If we set $M = \mathbb{T}$ and take $\varphi_s(x) = p.x$ where p is a secret integer, then the TLWE problem consists in recognizing approximations of multiples of $1/p$, so the TLWE abstraction can express cryptosystems based on the (dual) approx-GCD problem. Now, if we take a different ideal, for instance $I = (X^N + 1)$, then the canonical choice for a phase: $\varphi_s(\boldsymbol{a}, b) = b - \boldsymbol{sa}$ expresses RingLWE and Module-LWE [34], depending on the dimension of $\boldsymbol{a}$. But again, other choices of phases are possible, for instance $\varphi_{(f,g)} : \mathbb{T}_N[X]^2 \to \mathbb{T}_N[X], (x, y) \mapsto fx - gy$ for small secret polynomials f, g, would allow to build FHE over scale invariant version of NTRU.

Definition 3.2 (Canonical TLWE problem). *Let $k \geq 1$ be an integer, N be a power of 2 and $\alpha \in \mathbb{R}_{\geq 0}$ be a standard deviation. The canonical* TLWE *instantiation is the following: the secret key space $\mathcal{S}$ is composed by the binary vectors $s \in \mathbb{B}_N[X]^k$ that we assume to be uniformly chosen with $n \approx kN$ bits of entropy. The phase φ_s is defined over $M = \mathbb{T}_N[X]^k \times \mathbb{T}_N[X]$ by $\varphi_s((\boldsymbol{a}, b)) = b - \boldsymbol{s}\cdot\boldsymbol{a}$. It is by definition n-lipschitzian. The error distribution ξ is $(\mathbf{0}, \mathcal{D}_{\mathbb{T}_N[X],\alpha})$ where $\mathcal{D}_{\mathbb{T}_N[X],\alpha}$ is the modular Gaussian distribution of standard deviation α over $\mathbb{T}_N[X]$. By definition, a homogeneous* TLWE *sample can be constructed as $(\boldsymbol{a}, \boldsymbol{s}\cdot\boldsymbol{a} + e)$ where $\boldsymbol{a}$ is uniformly drawn in $\mathbb{T}_N[X]^k$ (or in a sufficiently dense submodule[8]) and $e \leftarrow \mathcal{D}_{\mathbb{T}_N[X],\alpha}$.*

Furthermore, we define as *trivial* the samples having the mask $\boldsymbol{a} = \mathbf{0}$ and *noiseless* the samples having the standard deviation $\alpha = 0$.

Definition 3.2 can be viewed as the analogue of the Gereral-LWE problem of [8] over the torus. It considers a continuum among anticyclic Module-LWE instances, between LWE (for $N = 1$) and RingLWE (for $k-1$). However, we restrict

[7] Probabilistic Polynomial Time.

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

the definition of canonical TLWE problem to only these particular cyclotomic instances, because they are the most efficient to implement with fast fourier transform. Also, the Gaussian error distribution can be sampled directly on the coefficients of the polynomials, rather than the general definition on the Lagrange basis.

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

- The message space is the image $\mathbb{T}_I[X]$ of φ_s,
- The ciphertext space is the domain M of φ_s.
- The encryption of μ is an approximation of a random preimage $\varphi_s^{-1}(\mu)$. Abstractly, a sample from $\mathcal{U}_{\varphi_s^{-1}(\mu)}+\xi$, and in the canonical form, the sum of the trivial sample $(\mathbf{0}, \mu)$ plus a homogeneous sample from $\mathcal{U}_{\ker(\varphi_s)}+\xi$.
- The (approximate) decryption of a ciphertext $\boldsymbol{c}$ is its image $\varphi_s(\boldsymbol{c})$.

From a practical point of view, the fact that the phase is lipschitzian makes this decryption resilient to numerical errors, and allows to work with approximations. This cryptosystem is also additively homomorphic, by linearity of the phase. However, this cryptosystem is noisy, in a sense that after encrypting and decrypting a message μ, the result is not exactly μ, but a close approximation $\mu + e$ where $e \leftarrow \xi$ is a small error.

There are use-cases, like floating point computations [15] or in general differential privacy, where these approximations of the plaintext are considered valid. However, if we need an exact result, we have two options. The first one is the historical choice in Regev cryptosystem: restrict the message space to a discrete subset, whose packing radius is larger than the amplitude of ξ, and retrieve the exact plaintext by rounding the phase. If rounding is easy to setup in practice, its non-linearity complicates the correctness analysis, especially when the current sample is not fresh, but rather a linear combination of previous samples. Also, restricting the message space prevents some floating point applications and bounds plaintext operations to just small abelian groups. The second option, consists in taking $\mathbb{E}(\xi) = 0$, and thus, the plaintext becomes the expectation of the phase. This option does not require to restrict the message space and works with infinite precision over the continuous one. Furthermore, the continuity and linearity of the expectation ease the analysis of morphism properties and of the noise propagation, but it requires to properly define the probability space Ω, which we do now.

Definition 3.3 (The Ω-probability space). *Since samples are either independent (random, noiseless, or trivial) fresh $\boldsymbol{c} \leftarrow \mathsf{TLWE}_{\mathbb{T}_N[X],\boldsymbol{s},\alpha}(\mu)$, or linear combination $\tilde{\boldsymbol{c}} = \sum_{i=1}^{p} e_i \cdot \boldsymbol{c_i}$ of other samples, the probability space Ω is the product of the probability spaces of each individual fresh samples $\boldsymbol{c}$ with the* TLWE *distributions defined in definitions 3.2, and of the probability spaces of all the coefficients $(e_1, \ldots, e_p) \in \mathbb{Z}_N[X]^p$ or $\mathbb{Z}^p$ that are obtained with randomized algorithm.*

In other words, instead of viewing a TLWE sample as a fixed value which is the result of one particular event in Ω, we will consider all the possible values

at once, and make statistics on them.

We now define some important functions on TLWE samples: message, error, noise variance, and noise norm. These functions are well defined mathematically, and can be used in the analysis of various algorithms. However, they cannot be directly computed or approximated in practice.

Definition 3.4. *Let $\boldsymbol{c}$ be a random variable $\in \mathbb{T}_N[X]^{k+1}$, which we'll interpret as a* TLWE *sample. All probabilities are on the Ω-space. We say that $\boldsymbol{c}$ is a* valid TLWE *sample if and only if there exists a key $\boldsymbol{s} \in \mathbb{B}_N[X]^k$ such that the distribution of the phase $\varphi_{\boldsymbol{s}}(\boldsymbol{c})$ is concentrated. If $\boldsymbol{c}$ is trivial, all keys $\boldsymbol{s}$ are equivalent, else the mask of $\boldsymbol{c}$ is uniformly random, so $\boldsymbol{s}$ is unique. We then define:*

- *the* message *of $\boldsymbol{c}$, denoted as $\mathsf{msg}(\boldsymbol{c}) \in \mathbb{T}_N[X]$ is the expectation of $\varphi_{\boldsymbol{s}}(\boldsymbol{c})$;*
- *the* error*, denoted $\mathsf{Err}(\boldsymbol{c})$, is equal to $\varphi_{\boldsymbol{s}}(\boldsymbol{c}) - \mathsf{msg}(\boldsymbol{c})$;*
- *$\mathsf{Var}(\mathsf{Err}(\boldsymbol{c}))$ denotes the variance of $\mathsf{Err}(\boldsymbol{c})$, which is by definition also equal to the variance of $\varphi_{\boldsymbol{s}}(\boldsymbol{c})$;*
- *finally, $\|\mathsf{Err}(\boldsymbol{c})\|_\infty$ denotes the maximum amplitude of $\mathsf{Err}(\boldsymbol{c})$ (possibly with overwhelming probability).*

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

Fact 3.5. Given p valid and independent TLWE samples $\boldsymbol{c_1}, \ldots, \boldsymbol{c_p}$ under the same key $\boldsymbol{s}$, and p integer polynomials $e_1, \ldots, e_p \in \mathfrak{R}$, if the linear combination $\boldsymbol{c} = \sum_{i=1}^{p} e_i \cdot \boldsymbol{c_i}$ is a valid TLWE sample, it satisfies: $\mathsf{msg}(\boldsymbol{c}) = \sum_{i=1}^{p} e_i \cdot \mathsf{msg}(\boldsymbol{c}_i)$, with variance $\mathsf{Var}(\mathsf{Err}(\boldsymbol{c})) \leq \sum_{i=1}^{p} \|e_i\|_2^2 \cdot \mathsf{Var}(\mathsf{Err}(\boldsymbol{c_i}))$ and noise amplitude $\|\mathsf{Err}(\boldsymbol{c})\|_\infty \leq \sum_{i=1}^{p} \|e_i\|_1 \cdot \|\mathsf{Err}(\boldsymbol{c_i})\|_\infty$. If the last bound is $< \frac{1}{4}$, then $\boldsymbol{c}$ is necessarily a valid TLWE sample (under the same key $\boldsymbol{s}$).

3.2 TGSW

As presented in previous section, TLWE samples can be linearly combined to obtain a new sample encrypting the linear combination of the messages. But when it comes to non linear operations on the samples, TLWE seems to miss some properties. In order to repair this lack, several schemes based on the different variants of LWE have been proposed. Between them, the most known solutions are the BGV constructions [8] and the GSW constructions [29]. We focus on this latter and on the improvements proposed in [5]. The security of GSW is based on the LWE problem and the construction is fully homomorphic. In this section we present a generalized scale invariant version of the FHE scheme GSW [29],

that we call TGSW (in the same line as TLWE). The scheme relies on a gadget decomposition function, which we also extend to polynomials. But most importantly, the novelty is that our function is an approximate decomposition, up to some precision parameter. This allows to improve running time and memory requirements for a small amount of additional noise.

Definition 3.6 (Abstract Gadget Decomposition). *Let M be a $\mathfrak{R}$-module, (as in Definition 3.1). We say that an efficient algorithm $Dec_{H,\beta,\epsilon}(\boldsymbol{v})$ is a valid decomposition on the gadget $H \in M^{\ell'}$ with quality $\beta \in \mathbb{R}_{>0}$ and precision $\epsilon \in \mathbb{R}_{>0}$ if and only if, for any* TLWE *sample $\boldsymbol{v} \in \mathbb{T}_N[X]^{k+1}$, it efficiently and publicly outputs a small vector $\boldsymbol{u} \in \mathfrak{R}^{\ell'}$ such that $\|\boldsymbol{u}\|_\infty \leq \beta$ and $\|\boldsymbol{u} \cdot H - \boldsymbol{v}\|_\infty \leq \epsilon$. Furthermore, the expectation of $\boldsymbol{u} \cdot H - \boldsymbol{v}$ must to be equal to 0 when $\boldsymbol{v}$ is uniformly distributed in M.*

To fix the ideas, we give an efficient canonical example of gadget decomposition, whose purpose is to decompose canonical TLWE ciphertexts. Overall, the canonical gadget is a block diagonal matrix, each column block containing a geometric decreasing sequence of constant polynomials in $\mathbb{T} \subseteq \mathbb{T}_N[X]$, and the corresponding decomposition function is the greedy algorithm.

In theory, decomposition algorithms should be randomized to ensure that the distribution of all error coefficients remain independent. In practice, our average case theorems already rely on an independence Heuristic 3.11 that we describe later in this section, which explains why we use a deterministic canonical decomposition.

Lemma 3.7 (Canonical Gadget Decomposition). *Let $M = \mathbb{T}_N[X]^{k+1}$ be the domain of the canonical* TLWE*, and ℓ and B_g be two positive integers, the canonical gadget are the $\ell' = (k+1)\ell$ rows of the matrix $H \in \mathcal{M}_{(k+1)\ell,k+1}(\mathbb{T}_N[X])$ as in (1).*

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

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

Proof. Let $\boldsymbol{v} = (\boldsymbol{a}, b) = (a_1, \dots, a_k, b = a_{k+1}) \in \mathbb{T}_N[X]^{k+1}$ be a TLWE sample, given as input to Algorithm 1. Let $\boldsymbol{u} = [u_{1,1}, \dots, u_{k+1,\ell}] \in \mathfrak{R}^{(k+1)\ell}$ be the corresponding output by construction $\|\boldsymbol{u}\|_\infty \leq B_g/2 = \beta$.

Algorithm 1 Gadget Decomposition of a TLWE sample

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

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

Let $\boldsymbol{\epsilon_{\mathbf{dec}}} = \boldsymbol{u} \cdot H - \boldsymbol{v}$. For all $i \in [\![1, k+1]\!]$ and $j \in [\![0, N-1]\!]$, we have by construction

$$\boldsymbol{\epsilon_{\mathbf{dec}}}_{i,j} = \sum_{p=1}^{\ell} u_{i,p} \cdot \frac{1}{B_g^p} - a_{i,j} = \bar{a}_{i,j} - a_{i,j}.$$

Since $\bar{a}_{i,j}$ is defined as the nearest multiple of $\frac{1}{B_g^\ell}$ on the torus, we have $|\bar{a}_{i,j} - a_{i,j}| \leq 1/2B_g^\ell = \epsilon$.

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

We are now ready to define TGSW samples, and to extend the notions of phase of valid sample, message and error of the samples.

Definition 3.8 (Abstract TGSW samples). *Consider the* TLWE *cryptosystem of error distribution ξ and of secret phase φ_s on the $\mathfrak{R}$-module M, and its associated gadget decomposition $Dec_{H,\beta,\varepsilon}$ over $H \in M^{\ell'}$. We say that $C \in M^{\ell'}$ is a fresh* TGSW *sample of $\mu \in \mathfrak{R}$ if and only if $C = Z + \mu \bullet H$ where each element of $Z \in M^{\ell'}$ is an Homogeneous* TLWE *sample (of 0) and error ξ. Reciprocally, we say that an element $C \in M^{\ell'}$ is a valid* TGSW *sample for the key s if and only if there exists a unique polynomial $\mu \in \mathfrak{R}$ (modulo $H \cdot \mathfrak{R}$) such that row of $C - \mu \bullet H$ is a valid* TLWE *sample of 0 for the key $\boldsymbol{s}$. We call the polynomial μ the message of C, and we denote it by $\mathsf{msg}(C)$. By extension, the phase of C denoted as $\varphi_{\boldsymbol{s}}(C) \in \mathbb{T}_I[X]^{\ell'}$ is the vector of the ℓ'* TLWE *phases of each line of C. In the same way, we define the error of C, denoted $\mathsf{Err}(C)$, as the list of the ℓ'* TLWE *errors of each line of C.*

If one instantiate the previous definition with the canonical TLWE (Definition 3.2) and the canonical decomposition algorithm (Lemma 3.7), one obtains

the canonical TGSW samples over $\mathbb{T}_N[X]^{(k+1)\ell}$, of binary key $\boldsymbol{s} \in \mathbb{B}_N[X]^k$, and Gaussian error of standard deviation α. Fresh canonical TGSW samples of a message $\mu \in \mathbb{Z}_N[X]$ are denoted $\mathsf{TGSW}_{\boldsymbol{s},\alpha}(\mu)$. Since TGSW samples are essentially vectors of TLWE samples, they are naturally compatible with linear combinations. And both phase and message functions remain linear.

Fact 3.9. Given p valid TGSW samples $C_1, \ldots, C_p$ of messages $\mu_1, \ldots, \mu_p$ under the same key, and with independent error coefficients, and given p integer polynomials $e_1, \ldots, e_p \in fR$, the linear combination $C = \sum_{i=1}^{p} e_i \bullet C_i$ is a sample of $\mu = \sum_{i=1}^{p} e_i \cdot \mu_i$, with variance

$$\mathsf{Var}(C) = \left(\sum_{i=1}^{p} \|e_i\|_2^2 \cdot \mathsf{Var}(C_i) \right)^{1/2}$$

and noise infinity norm

$$\|\mathsf{Err}(C)\|_\infty = \sum_{i=1}^{p} \|e_i\|_1 \cdot \|\mathsf{Err}(C_i)\|_\infty .$$

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

Fact 3.10. For all $A \in \mathcal{M}_{p,k+1}(\mathbb{T}_N[X])$, $\|\varphi_{\boldsymbol{s}}(A)\|_\infty \leq (1 + kN) \|A\|_\infty$.

Heuristic In order to characterize the average case behaviour of our homomorphic operations, we shall rely on the heuristic assumption of independence below. This heuristic will only be used for practical average-case bounds. Our worst-case theorems and lemma based on the infinite norm do not use it at all.

Assumption 3.11 (Independence Heuristic). All the coefficients of the errors of TLWE or TGSW samples that occur in all the linear combinations we consider are independent and concentrated. More precisely, they are σ-subgaussian where σ is the square-root of their variance.

This assumption allows us to bound the variance of the noise instead of its norm, and to provide realistic average-case bounds which often correspond to the square root of the worst-case ones. The error can easily be proved subgaussian, since each coefficients are always obtained by convolving Gaussians or zero-centered bounded uniform distributions. What remains heuristic is the independence between all the coefficients. Indeed, dependencies between coefficients may affect the variance of their combinations in both directions. The independence of coefficients can be proved if we add enough entropy in the decomposition algorithm (and if we increase all the other parameters to compensate), but as noticed in [24], this work-around seems just to be a proof artifact, and is experimentally not needed. Since average case corollaries should reflect practical results, we leave the independence of subgaussian samples as a heuristic assumption. In Figure 11 we show an experimental validation of our independence assumption.

3.3 Products

Linear operations are not sufficient to achieve Fully Homomorphic Encryption. The original definition of GSW in [29] proposed a construction to achieve a homomorphic internal product between the integer messages of two GSW ciphertexts (which live in the ring $\mathfrak{R}$). Due to an asymetry in the noise propagation, it was noticed in [30] and [5] that GSW ciphertext are particularly suited to evaluate long chains of products, or branching programs. In [10], the authors noticed that for these circuits, a large part of the computations in the GSW internal product was subsequently unused, if the final goal was just to decrypt the message. Not performing these computations yields a huge polynomial speed-up. In this section, we provide an intrinsic explanation for the correctness of these partial computations, by defining an external product between a TGSW ciphertext and a TLWE samples, and prove that it is homomorphic to the external $\mathfrak{R}$-module product between the two plaintexts. A direct comparison between the external and internal product algorithms retroactively explains the speed-up of [10]. It also emphasis on the asymetric nature of TGSW products: the reason why branching algorithms or long chains of fresh multiplications are much more efficient to evaluate with GSW than balanced binary trees, is not only the asymetry in the noise propagation, but also because only the first ones can be mapped to the simple plaintext external product.

Definition 3.12 (External product). *We define the product $\boxdot$ as*

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

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

The formula is almost identical to the classical product defined in the original GSW scheme in [29], except that only one vector needs to be decomposed. For this reason, the following theorem shows that we get almost the same noise propagation formula, with an additional term that comes from the approximations in the decomposition.

Theorem 3.13 (Worst-case External Product). *Let A be a valid* TGSW *sample of message μ_A and let $\boldsymbol{b}$ be a valid* TLWE *sample of message $\mu_{\boldsymbol{b}}$. Then $A \boxdot \boldsymbol{b}$ is a* TLWE *sample of message $\mu_A \cdot \mu_{\boldsymbol{b}}$ and*

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

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

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

way, as $\boldsymbol{b} = \mathsf{TLWE}(\mu_{\boldsymbol{b}})$, then by definition it is equal to $\boldsymbol{b} = \boldsymbol{z_b} + (\boldsymbol{0}, \mu_{\boldsymbol{b}})$, where $\boldsymbol{z_b}$ is a TLWE encryption of 0. Let

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

Let $\boldsymbol{u} = Dec_{H,\beta,\epsilon}(\boldsymbol{b}) \in \mathfrak{R}^{(k+1)\ell}$. By definition $A \boxdot \boldsymbol{b}$ is equal to

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

From definition 3.6, we have that $\boldsymbol{u} \cdot H = \boldsymbol{b} + \boldsymbol{\epsilon_{dec}}$, where $\|\boldsymbol{\epsilon_{dec}}\|_\infty = \|\boldsymbol{u} \cdot H - \boldsymbol{b}\|_\infty \leq \epsilon$. So

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

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

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

Taking the expectation, we get that $\mathsf{msg}(A \boxdot \boldsymbol{b}) = 0 + 0 + 0 + \mu_A \mu_{\boldsymbol{b}}$, and so $\mathsf{Err}(A \boxdot \boldsymbol{b}) = \varphi_{\boldsymbol{s}}(A \boxdot \boldsymbol{b}) - \mu_A \mu_{\boldsymbol{b}}$. Then thanks to Fact 3.10, we have

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

The result follows. □

We similarly obtain the more realistic average-case noise propagation, based on the independence heuristic 3.11, by bounding the Gaussian variance instead of the amplitude.

Corollary 3.14 (Average-case External Product). *Under the same conditions of theorem 3.13 and by assuming the heuristic 3.11, we have that*

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

Proof. Let $\vartheta_A = \mathsf{Var}(\mathsf{Err}(A)) = \mathsf{Var}(\varphi_{\boldsymbol{s}}(Z_A))$ and $\vartheta_{\boldsymbol{b}} = \mathsf{Var}(\mathsf{Err}(\boldsymbol{b})) = \mathsf{Var}(\varphi_{\boldsymbol{s}}(\boldsymbol{z_b}))$. By using the same notations as in the proof of theorem 3.13 we have that the error of $A \boxdot \boldsymbol{b}$ is $\mathsf{Err}(A \boxdot \boldsymbol{b}) = \boldsymbol{u} \cdot \mathsf{Err}(A) + \mu_A \cdot \varphi_{\boldsymbol{s}}(\boldsymbol{\epsilon_{dec}}) + \mu_A \cdot \mathsf{Err}(\boldsymbol{b})$ and thanks to assumption 3.11 and lemma 3.10, we have :

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

□

The last corollary describes exactly the classical internal product between two TGSW samples, already presented in [29, 5, 26, 24] with adapted notations. As we mentioned before, it consists in $(k+1)\ell$ independent computations of the $\boxdot$ product. As for the external product, we analyze the noise growth in both worst and average case.

Corollary 3.15 (Internal Product). *Let the product*

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

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

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

in the worst case. If $\|\mathsf{Err}(A \boxdot B)\|_\infty \leq 1/4$ we are guaranteed that $A \boxdot B$ is a valid TGSW *sample.*
Furthermore, by assuming the heuristic 3.11, we have that

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

in the average case.

Proof. Let A and B be two TGSW samples, and μ_A and μ_B their message. Let $\boldsymbol{h_i}$ denote the i-th row of the gadget matrix H. By definition, the i-th row of B encodes $\mu_B \cdot \boldsymbol{h_i}$, so the i-th row of $A \boxtimes B$ encodes $(\mu_A \mu_B) \cdot \boldsymbol{h_i}$. This proves that $A \boxtimes B$ encodes $\mu_A \mu_B$. Since the internal product $A \boxtimes B$ consists in $(k+1)\ell$ independent runs of the external products $A \boxdot \boldsymbol{b_i}$, the noise propagation formula directly follows from Theorem 3.13 and Corollary 3.14. □

3.4 `CMux` gate

With the homomorphic operations described until now it is possible to construct small circuits. To ease this construction, we now define the controlled selector gate (or `CMux` gate where `C` stands for controlled), which can be considered as the bridge between the external product arithmetic, and high level circuits.

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

In leveled circuits, the rule to build valid circuits using CMux gates is that all control wires (TGSW) are freshly generated by the user, and the data input ports of our gates can be either freshly generated or connected to a data output or to another gate. In Section 6.2, we propose an efficient way to transform a TLWE sample in a TGSW sample, in order to make the circuits entirely composable (and so relax the condition requiring freshly generated control wires).

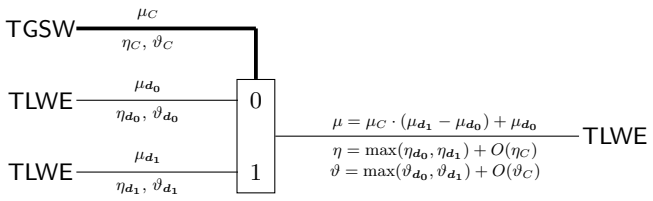


Fig. 1. CMux **gate -** The CMux gate takes in input a TGSW sample C with message μ_C, a TLWE sample $\boldsymbol{d_0}$ with message $\mu_{\boldsymbol{d_0}}$ and a TLWE sample $\boldsymbol{d_1}$ with message $\mu_{\boldsymbol{d_1}}$. It outputs a TLWE sample with message $\mu = \mu_C \cdot (\mu_{\boldsymbol{d_1}} - \mu_{\boldsymbol{d_0}}) + \mu_{\boldsymbol{d_0}}$. The η's and ϑ's represent respectively the noise in the worst and average case, for both the inputs and the output.

Lemma 3.16 (CMux **gate).** *Let* $\boldsymbol{d_0}, \boldsymbol{d_1} \in \mathsf{TLWE}_{\boldsymbol{s}}(\mathbb{T}_N[X])$ *and* $C \in \mathsf{TGSW}_{\boldsymbol{s}}(\{0,1\})$*. Then* $\mathsf{msg}(\mathtt{CMux}(C, \boldsymbol{d_1}, \boldsymbol{d_0})) = \mathsf{msg}(C)?\mathsf{msg}(\boldsymbol{d_1}):\mathsf{msg}(\boldsymbol{d_0})$*. Furthermore*

- $\|\mathsf{Err}(\mathtt{CMux}(C, \boldsymbol{d_1}, \boldsymbol{d_0}))\|_\infty \leq \max(\|\mathsf{Err}(\boldsymbol{d_0})\|_\infty, \|\mathsf{Err}(\boldsymbol{d_1})\|_\infty) + \eta(C)$*,*
- $\mathsf{Var}(\mathsf{Err}(\mathtt{CMux}(C, \boldsymbol{d_1}, \boldsymbol{d_0}))) \leq \max(\mathsf{Var}(\mathsf{Err}(\boldsymbol{d_0})), \mathsf{Var}(\mathsf{Err}(\boldsymbol{d_1}))) + \vartheta(C)$*, in the conditions of Assumption 3.11,*

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

Proof. The formulas for the noise in the worst and average cases are a consequence of Theorem 3.13 and Corollary 3.14. However, we need to explain why there is a max instead of the sum we would obtain by blindly applying these results. Let $\boldsymbol{d} = \boldsymbol{d_1} - \boldsymbol{d_0}$, recall that in the proof of Theorem 3.13, the expression of $C \boxdot \boldsymbol{d}$ is $\mathrm{Dec}_{H,\beta,\epsilon}(\boldsymbol{d}) \cdot Z_C + \mu_C \boldsymbol{\epsilon}_{\mathrm{dec}} + \mu_C \boldsymbol{z_d} + (\boldsymbol{0}, \mu_C \cdot \mu_d)$, where $C = Z_C + \mu_C \cdot H$ and $\boldsymbol{d} = \boldsymbol{z_d} + (\boldsymbol{0}, \mu_d)$, Z_C and $\boldsymbol{z_d}$ are respectively TGSW and TLWE samples of 0, and $\|\boldsymbol{\epsilon}_{\mathrm{dec}}\|_\infty \leq \epsilon$. Thus, $\mathtt{CMux}(C, \boldsymbol{d_1}, \boldsymbol{d_0})$ is the sum of four terms:

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

Thus, summing the three terms concludes the proof. For the average case, the formula is proven in the same way by using the results of Corollary 3.14 and replacing all norm inequalities by variance inequalities. □

Notations In the rest of the paper, the notation TLWE is used to denote the (canonical scalar) binary TLWE problem (i.e. the LWE problem described in Section 2). To distinguish it from the Ring mode, we introduce the notation TRLWE. The TGSW samples are only used in ring mode, but we use the notation TRGSW to keep uniformity with the TRLWE notation.

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

4 Building blocks for TFHE

TLWE and TRLWE samples are largely used in the rest of the paper, and the schemes we describe switch from a type to another constantly. To do that, three basic tools are used: the key switching, the sample extraction and the blind rotation. Each one of them is described in detail in next sections.

4.1 Key Switching revisited

We revisit the well known key switching procedure, largely described in the literature. The principal interest of key switching, as the name suggests, is to switch between keys in different parameter sets.

We show that this procedure as a larger potential. It allows to switch between the scalar and polynomial message spaces $\mathbb{T}$ and $\mathbb{T}_N[X]$, and more generally, it has the ability to homomorphically evaluate linear morphisms f from any $\mathbb{Z}$-module $\mathbb{T}^p$ to $\mathbb{T}_N[X]$. We define two key switching flavors, one for a publicly known f, and one for a secret f encoded in the key switching key.

In the following, we denote $\mathsf{PubKS}(f, \mathsf{KS}, \mathbf{c})$ and $\mathsf{PrivKS}(\mathsf{KS}^{(f)}, \mathbf{c})$ the output of Algorithm 2 and Algorithm 3, taking in input the functional key switching keys KS and $\mathsf{KS}^{(f)}$ respectively and a TLWE ciphertext $\mathbf{c}$.

As the inputs and the outputs are instantiated with different parameter sets and we want to keep the same name for the variables $n, N, \alpha, \ell, B_g, \ldots$, we add an under bar to the output parameters to distinguish them from the input parameters.

Theorem 4.1. *(Public Key Switching) Given p* TLWE *ciphertexts $\mathbf{c}^{(z)} \in \mathsf{TLWE}_{\mathfrak{K}}(\mu_z)$, a public R-lipschitzian morphism $f : \mathbb{T}^p \to \mathbb{T}_{\underline{N}}[X]$ of $\mathbb{Z}$-modules, and $\mathsf{KS}_{i,j} \in \mathsf{T(R)LWE}_{\underline{K},\underline{\gamma}}(\frac{\mathfrak{K}_i}{2^j})$ with standard deviation $\underline{\gamma}$, Algorithm 2 outputs a* T(R)LWE *sample $\boldsymbol{c} \in \mathsf{T(R)LWE}_{\underline{K}}(f(\mu_1, \ldots, \mu_p))$ such that:*

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

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

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

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

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

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

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

Applying the expectation on each side, we obtain $\mathsf{msg}(\boldsymbol{c})$ on the left, and $f(\mu_1, \dots, \mu_p)$ on the right, since all the error terms have expectation 0 and

f is linear. For the worst-case bound, we obtain that:

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

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

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

Remark 2. The TLWE-to-T(R)LWE public key switching procedure we described, allow to switch between the scalar message space $\mathbb{T}$ and the polynomial message space $\mathbb{T}_N[X]$. The same procedure can be used to perform a TLWE-to-TLWE public key switching and switch between scalar message spaces. Here is why we put parenthesis around the R of T(R)LWE. In practice, the key switching key is composed by TLWE encryptions of the old secret key, and the noise growth formulas still the same, with the factor N equal to 1. We use the TLWE-to-TLWE public key switching in Section 5.3.

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

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

Input: p TLWE ciphertexts $\mathfrak{c}^{(z)} \in \mathsf{TLWE}_{\mathfrak{K}}(\mu_z)$, a key switching key $\mathsf{KS}^{(f)}_{z,i,j} \in \mathsf{T(R)LWE}_{\underline{K}}(f(0,\ldots,0,\frac{\mathfrak{K}_i}{2^j},0,\ldots,0))$ where $f : \mathbb{T}^p \to \mathbb{T}_{\underline{N}}[X]$ is a secret R-lipschitzian morphism and $\frac{\mathfrak{K}_i}{2^j}$ is at position z (also, $\mathfrak{K}_{n+1} = -1$ by convention).
Output: A T(R)LWE sample $\boldsymbol{c} \in \mathsf{T(R)LWE}_{\underline{K}}(f(\mu_1,\ldots,\mu_p))$.
1: **for** $i \in [\![1, n+1]\!]$, $z \in [\![1, p]\!]$ **do**
2: Let $\tilde{c}_i^{(z)}$ be the closest multiple of $\frac{1}{2^t}$ to $\mathfrak{c}_i^{(z)}$, thus $|\tilde{c}_i^{(z)} - \mathfrak{c}_i^{(z)}| < 2^{-(t+1)}$
3: Binary decompose each $\tilde{c}_i^{(z)} = \sum_{j=1}^t \tilde{c}_{i,j}^{(z)} \cdot 2^{-j}$ where $\tilde{c}_{i,j}^{(z)} \in \{0,1\}$
4: **end for**
5: **return** $-\sum_{z=1}^p \sum_{i=1}^{n+1} \sum_{j=1}^t \tilde{c}_{i,j}^{(z)} \cdot \mathsf{KS}^{(f)}_{z,i,j}$

Theorem 4.2. *(Private Key Switching) Given p TLWE ciphertexts $\mathfrak{c}^{(z)} \in \mathsf{TLWE}_{\mathfrak{K}}(\mu_z)$, and $\mathsf{KS}^{(f)}_{i,j} \in \mathsf{T(R)LWE}_{\underline{K},\gamma}(f(0,\ldots,\frac{\mathfrak{K}_i}{2^j},\ldots,0))$ where $f : \mathbb{T}^p \to \mathbb{T}_{\underline{N}}[X]$ is a private R-lipschitzian morphism of $\mathbb{Z}$-modules, Algorithm 3 outputs a T(R)LWE sample $\boldsymbol{c} \in \mathsf{T(R)LWE}_{\underline{K}}(f(\mu_1,\ldots,\mu_p))$ such that:*

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

– $\mathsf{Var}(\mathsf{Err}(\boldsymbol{c})) \leq R^2\,\mathsf{Var}(\mathsf{Err}(\mathfrak{c})) + (n+1)R^2 2^{-2(t+1)} + pt(n+1)\underline{\vartheta}_{\mathsf{KS}}$ *(average case),*

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

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

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

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

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

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

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

The proof for the variance is similar.

4.2 Sample Packing and Sample Extraction.

A TRLWE message is a polynomial with N coefficients, which can be viewed as N slots over $\mathbb{T}$. It is easy to homomorphically extract a coefficient as a scalar TLWE sample with the same key. We recall that a binary TLWE key $\mathfrak{K} \in \mathbb{B}^n$ can be interpreted as a TRLWE key $K \in \mathbb{B}_N[X]^k$ having the same sequence of coefficients, and vice-versa.

Given a TRLWE sample $\boldsymbol{c} = (\boldsymbol{a}, b) \in \mathsf{TRLWE}_K(\mu)$ and a position $p \in [0, N-1]$, we call $\mathsf{SampleExtract}_p(\boldsymbol{c})$ the TLWE sample $(\mathfrak{a}, \mathfrak{b})$ where $\mathfrak{b} = b_p$ and $\mathfrak{a}_{N(i-1)+j+1}$ is the $(p-j)$-th coefficient of a_i (using the N-antiperiodic indexes). This extracted sample encodes the p-th coefficient μ_p with at most the same noise variance or amplitude as $\boldsymbol{c}$. In the rest of the paper, we will simply write $\mathsf{SampleExtract}(\boldsymbol{c})$ when $p = 0$.

In Section 5, we show how the KeySwitching and the SampleExtract procedures are used to efficiently pack data, unpack and move data across the slots, and how it differs from usual packing techniques.

4.3 Blind Rotate

The BlindRotate algorithm multiplies the polynomial encrypted in the input TRLWE ciphertext by an encrypted power of X. The effect produced is a rotation of the coefficients. The algorithm consists in two parts. The first one (line 3) is the rotation by a known power of X The second one (loop at line 4) is the rotation by a secret power of X, which is performed by using the `CMux` gate, described in Section 3.4.

Algorithm 4 BlindRotate

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

Theorem 4.3. *Let $H \in \mathcal{M}_{(k+1)\ell,k+1}(\mathbb{T}_N[X])$ the gadget matrix and $Dec_{H,\beta,\epsilon}$ its efficient approximate gadget decomposition algorithm with quality β and precision ϵ defining* TRLWE *and* TRGSW *parameters. Let $\alpha \in \mathbb{R}_{\geq 0}$ be a noise parameter, $\mathfrak{K} \in \mathbb{B}^n$ be a* TLWE *secret key and $K \in \mathbb{B}_N[X]^k$ be its* TRLWE *interpretation. Given one sample $\boldsymbol{c} \in \mathsf{TRLWE}_K(\boldsymbol{v})$ with $\boldsymbol{v} \in \mathbb{T}_N[X]$, $p+1$*

integers $a_1, \ldots, a_p$ *and* $b \in \mathbb{Z}/2N\mathbb{Z}$*, and* p TRGSW *ciphertexts* $C_1, \ldots, C_p$*, where each* $C_i \in \mathsf{TRGSW}_{K,\alpha}(s_i)$ *for* $s_i \in \mathbb{B}$*. Algorithm 4 outputs a sample* $\mathsf{ACC} \in \mathsf{TRLWE}_K(X^{-\rho} \cdot \boldsymbol{v})$ *where* $\rho = b - \sum_{i=1}^{p} s_i a_i$*, such that:*

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

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

Proof. Theorem 4.3 follows from the fact that algorithm 4 calls p times the `CMux` evaluation.

We define $\mathsf{BlindRotate}(\boldsymbol{c}, (a_1, \ldots, a_p, b), (C_1, \ldots, C_p))$, the procedure described in Algorithm 4 that outputs the TRLWE sample ACC as in Theorem 4.3.

5 Leveled Homomorphic Encryption

The main goal of Homomorphic Encryption is to perform computations on encrypted data. In previous sections we described all the different tools to manipulate the ciphertexts. In this section we show how to use them to construct homomorphic circuits. In particular, we describe the evaluation of a random function via its look-up table and we propose two packing techniques that can be used to accelerate the evaluation.

Various packing techniques have already been proposed for homomorphic encryption: the Lagrange embedding in Helib [32, 31], the diagonal matrices encoding in [39] or the CRT encoding in [6]. The message space is often a finite ring (e.g. $\mathbb{Z}/p\mathbb{Z}$), and the packing function is in general chosen as a ring isomorphism that preserves the structure of $\mathbb{Z}/p\mathbb{Z}^N$. This way, elementary additions or products can be performed simultaneously on N independent slots, and thus, packing is in general associated to the concept of batching a single operation on multiple datasets. These techniques can have some limitations, especially if in the whole program, each function is only run on a single dataset, and most of the slots are unused. This is particularly true in the context of GSW evaluations, where functions are split into many branching algorithms or automata, that are each executed only once.

In the rest of the paper, packing refers to the canonical coefficients embedding function, that maps N TLWE messages $\mu_0, \ldots, \mu_{N-1} \in \mathbb{T}$ into a single TRLWE message $\mu(X) = \sum_{i=0}^{N-1} \mu_i X^i$. This function is a $\mathbb{Z}$-module isomorphism. Messages can be homomorphically unpacked from any slot using the (noiseless) $\mathsf{SampleExtract}$ procedure, described in Section 4.2. Reciprocally, we can repack, move data across the slots, or clear some slots by using our public functional key switching from Algorithm 2 to evaluate respectively the canonical coefficient embedding function (i.e. the identity), a permutation, or a projection. Since these functions are 1-lipschitzian, by theorem 4.1, these keyswitch operations only induce a linear noise overhead. It is arguably more straightforward than

the permutation network technique used in Helib. But as in [6, 14, 21], our technique relies on a circular security assumption, even in the leveled mode since our keyswitching key encrypts its own key bits[9].

We now analyze how packing can speed-up TRGSW leveled computations, first for look-up tables or random functions, and then for most arithmetic functions.

5.1 Arbitrary functions and Look-Up Tables

The first class of functions we analyze are arbitrary functions $f : \mathbb{B}^d \to \mathbb{T}^s$. Such functions can be expressed via a Look-Up Table (LUT), containing the list of 2^d input values (each one composed by d bits) and corresponding LUT values for the s sub-functions (1 element in $\mathbb{T}$ per sub-function f_j). We note the LUT values with $\sigma_{j,h} \in \mathbb{T}$, where $j \in [\![0, s-1]\!]$ is the sub-function index, and $h \in [\![0, 2^d - 1]\!]$ is the input index.

In order to compute $f(x) = (f_0(x), f_1(x), \ldots, f_{s-1}(x))$, where $x = \sum_{i=0}^{d-1} x_i 2^i$ is a d-bit integer, the classical evaluation of such function, as proposed in [11, 17], consists in evaluating the s sub-functions $f_0, f_1, \ldots, f_{s-1}$ separately. Each of them consists in a binary decision tree composed by $2^d - 1$ CMux gates. The total complexity of the classical evaluation requires therefore to execute about $s \cdot 2^d$ CMux gates. Let's call $o_j = f_j(x) \in \mathbb{T}$ the j-th output of $f(x)$, for $j = 0, \ldots, s-1$. Figure 2 summarizes the idea of the computation of o_j.

In this section we present two techniques, that we call horizontal and vertical packing, that can be used to improve the evaluation of a LUT. The packing technique is the same in both cases: the idea is to pack N TLWE messages inside the polynomial coefficients of a single TRLWE ciphertext. The names horizontal and vertical refer to the two different ways to use such packing. Intuitively, they describe in which sense the data of the LUT are packed and manipulated in order to evaluate the function f.

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

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

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

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

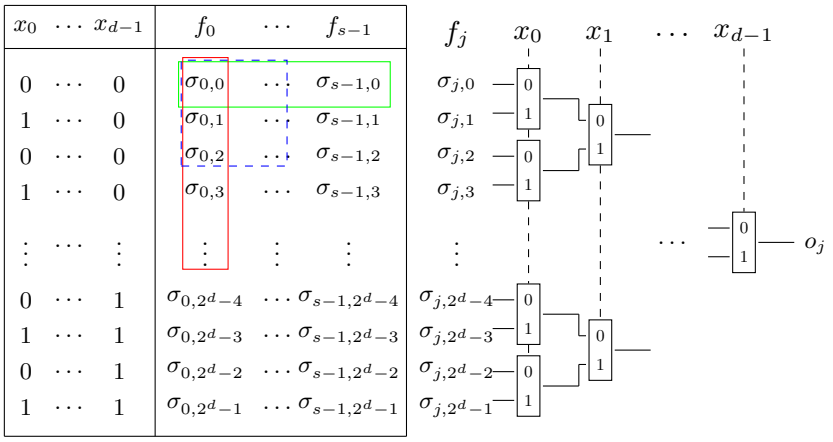


Fig. 2. LUT with `CMux` tree - Intuitively, the horizontal rectangle encircles the bits packed in the horizontal packing, while the vertical rectangle encircles the bits packed in the the vertical packing. The dashed square represents the packing in the case where the two techniques are mixed. The right part of the figure represents the evaluation of the sub-function f_j on $x = \sum_{i=0}^{d-1} x_i 2^i$ via a `CMux` binary decision tree.

Remark 3. In order to evaluate $f(x)$, the total amount of homomorphic `CMux` gates to be evaluated is $s(2^d - 1)$. If the function f is public, trivial samples of the LUT values $\sigma_{j,0}, \ldots, \sigma_{j,N-1}$ are used as inputs in the `CMux` gates. If f is private, the LUT values $\sigma_{j,0}, \ldots, \sigma_{j,N-1}$ are given encrypted. An analysis of the noise propagation in the binary decision `CMux` tree has already been given in [26] and [17].

Horizontal Packing (or Batching) The idea of the *horizontal packing* is to evaluate all the outputs of the function f together, instead of evaluating all the f_j separately. This is possible by using TRLWE samples, as the message space is $\mathbb{T}_N[X]$. In fact, we could encrypt up to N LUT values $\sigma_{j,h}$ (for a fixed $h \in [\![0, 2^d - 1]\!]$) per TRLWE sample and evaluate the binary decision tree as described before. The number of `CMux` gates to evaluate is $\lceil \frac{s}{N} \rceil (2^d - 1)$. This technique is optimal if the size s of the output is a multiple of N. Unfortunately, s is in general $\leq N$ and the number of gates to evaluate remains $2^d - 1$. The evaluation of the function f is then only s times faster than the non-packed approach. As not all the slots are used, this technique it is not optimal if s is small. The elementary Lemma 5.1 specifies the noise propagation and it follows immediately from Lemma 3.16 and from the construction of the binary decision `CMux` tree, which has depth d.

Lemma 5.1 (Horizontal Packing - Batching). *Let* $\boldsymbol{d}_0, \ldots, \boldsymbol{d}_{2^d-1}$ *be* TRLWE *samples*[10] *such that* $\boldsymbol{d}_h \in \mathsf{TRLWE}_K(\sum_{j=0}^{s} \sigma_{j,h} X^j)$ *for* $h \in [\![0, 2^d - 1]\!]$*. Here the* $\sigma_{j,h}$ *are the LUT values relative to an arbitrary function* $f : \mathbb{B}^d \to \mathbb{T}^s$*. Let* $C_0, \ldots, C_{d-1}$ *be* TRGSW *samples, such that* $C_i \in \mathsf{TRGSW}_K(x_i)$ *with* $x_i \in \mathbb{B}$ *(for* $i \in [\![0, d-1]\!]$*), and* $x = \sum_{i=0}^{d-1} x_i 2^i$*. Let* $\boldsymbol{d}$ *be the* TRLWE *sample output by the* f *evaluation of the binary decision* `CMux` *tree for the LUT (described in figure 2). Then, using the same notations as in lemma 3.16 and setting* $\mathsf{msg}(\boldsymbol{d}) = f(x)$*:*

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

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

Vertical Packing In order to improve the evaluation of the LUT, we propose a second optimization called *Vertical Packing*. As for the horizontal packing we use the TRLWE encryption to encode N values at the same time. But now, instead of packing the LUT values $\sigma_{j,h}$ with respect to a fixed $h \in [\![0, 2^d - 1]\!]$ i.e. "horizontally", we pack N values $\sigma_{j,h}$ "vertically", with respect to a fixed $j \in [\![0, s-1]\!]$. Then, instead of just evaluating a full `CMux` tree, we use a different approach.

If the LUT values are packed in "boxes", our technique first uses a packed `CMux` tree to select the right box, and then, a blind rotation (Algorithm 4) to find the right element inside the selected box. Figure 3 sums up the schematized idea of the entire procedure.

Now, suppose that one wants to evaluate the function f, or just one of its sub-functions f_j, on a fixed input $x = \sum_{i=0}^{d-1} x_i 2^i$. We assume we know the LUT associated to f_j as in figure 2. The output of $f_j(x)$ is just the LUT value $\sigma_{j,x}$ at position x.

Let $\delta = \log_2(N)$. We analyse the general case where 2^d is a multiple of $N = 2^\delta$. The LUT of f_j, which is a column of 2^d values, is now packed as $2^d/N$ TRLWE ciphertexts $\boldsymbol{d_0}, \ldots, \boldsymbol{d}_{2^{d-\delta}-1}$, where each $\boldsymbol{d}_k$ encodes N consecutive LUT values $\sigma_{j,kN}, \ldots, \sigma_{j,(k+1)N-1}$. To retrieve $f_j(x)$, we first need to select the block that contains $\sigma_{j,x}$. This block has index $p = \lfloor x/N \rfloor$, whose bits are the $d - \delta$ most significant bits of x. Since the TRGSW encryptions of these bits are among our inputs, one can use a `CMux` tree to select this block $\boldsymbol{d_p}$. Then, $\sigma_{j,x}$ is the ρ-th coefficient of the message of $\boldsymbol{d_p}$ where $\rho = x \mod N = \sum_{i=0}^{\delta-1} x_i 2^i$. The bits of ρ are the δ least significant bits of x, which are also available as TRGSW ciphertexts in our inputs. We can therefore use a blind rotation (Algorithm 4) to homomorphically multiply $\boldsymbol{d_p}$ by $X^{-\rho}$, which brings the coefficient $\sigma_{j,x}$ in position 0, and finally, we extract it with a SampleExtract. Algorithm 5 details the evaluation of $f_j(x)$.

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

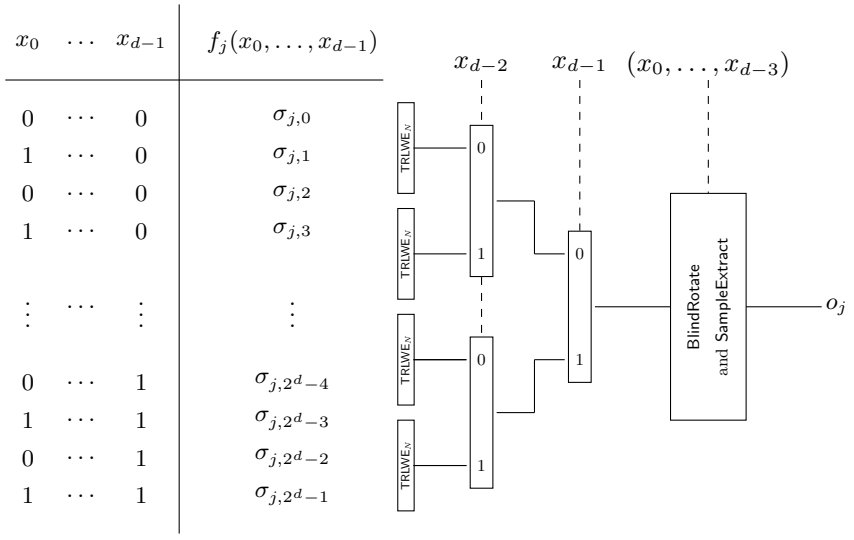


Fig. 3. Vertical packing for the evaluation of the f_j LUT - As described in algorithm 5, the image represents the idea of evaluation of the sub-function f_j on $x = \sum_{i=0}^{d-1} x_i 2^i$ via vertical packing technique. After "vertically" packing the LUT values $\sigma_{j,h}$ (for $h \in [\![0, 2^d - 1]\!]$) in groups of size N, inside TRLWE samples, a `CMux` tree, a blind rotation and a sample extract are evaluated. The `CMux` tree is initially used to select the TRLWE sample containing the output value. Then the output value is moved in the place of the constant coefficient of the TRLWE message by using the blind rotation (Algorithm 4) and extracted by using the sample extraction (Section 4.2). The bits of x are given as TRGSW samples and the final result $o_j = f_j(x)$ is extracted as a TLWE sample. In our example, we fixed $2^d = 4N$.

The entire cost of the evaluation of $f_j(x)$ with Algorithm 5 consists in $\frac{2^d}{N} - 1$ `CMux` gates and a single blind rotation, which corresponds to δ `CMux` gates. Overall, we get a speed-up by a factor N on the evaluation of each partial function, so a factor N in total.

Lemma 5.2 (Vertical Packing LUT of f_j). *Let $f_j : \mathbb{B}^d \to \mathbb{T}$ be a sub-function of the arbitrary function $f : \mathbb{B}^d \to \mathbb{T}^s$, with LUT values $\sigma_{j,0}, \dots, \sigma_{j,2^d-1}$. Let $\boldsymbol{d}_0, \dots, \boldsymbol{d}_{\frac{2^d}{N}-1}$ be* TRLWE *samples, such that $\boldsymbol{d}_p \in \mathsf{TRLWE}_K(\sum_{i=0}^{N-1} \sigma_{j,pN+i} X^i)$ for $p \in [\![0, \frac{2^d}{N} - 1]\!]$[11]. Let $C_0, \dots, C_{d-1}$ be* TRGSW *samples, such that $C_i \in \mathsf{TRGSW}_K(x_i)$, with $x_i \in \mathbb{B}$ and $i \in [\![0, d-1]\!]$.*

[11] If the sub-function f_j and its LUT are public, the LUT values $\sigma_{j,0}, \dots, \sigma_{j,2^d-1}$ can be given in clear. This means that the TRLWE samples $\boldsymbol{d}_p$, for $p \in [\![0, \frac{2^d}{N} - 1]\!]$ are given as trivial TRLWE samples $\boldsymbol{d}_p \leftarrow (\mathbf{0}, \sum_{i=0}^{N-1} \sigma_{j,pN+i} X^i)$ in input to algorithm 5.

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

Input: A list of $\frac{2^d}{N}$ TRLWE samples $\boldsymbol{d}_p \in \mathsf{TRLWE}_K(\sum_{i=0}^{N-1} \sigma_{j,pN+i} X^i)$ for $p \in [\![0, \frac{2^d}{N} - 1]\!]$, a list of d TRGSW samples $C_i \in \mathsf{TRGSW}_K(x_i)$, with $x_i \in \mathbb{B}$ and $i \in [\![0, d-1]\!]$,
Output: A TLWE sample $\mathfrak{c} \in \mathsf{TLWE}_{\mathfrak{K}}(o_j = f_j(x))$, with $x = \sum_{i=0}^{d-1} x_i 2^i$
1: Evaluate the binary decision `CMux` tree of depth $d - \delta$, with TRLWE inputs $\boldsymbol{d}_0, \ldots, \boldsymbol{d}_{\frac{2^d}{N}-1}$ and TRGSW inputs $C_\delta, \ldots, C_{d-1}$, and output a TRLWE sample $\boldsymbol{d}$
2: $\boldsymbol{d} \leftarrow \mathsf{BlindRotate}(\boldsymbol{d}, (2^0, \ldots, 2^{\delta-1}, 0), (C_0, \ldots, C_{\delta-1}))$
3: Return $\mathfrak{c} = \mathsf{SampleExtract}(\boldsymbol{d})$

Then algorithm 5 outputs a TLWE *sample* $\mathfrak{c}$ *such that* $\mathsf{msg}(\mathfrak{c}) = f_j(x) = o_j$ *where* $x = \sum_{i=0}^{d-1} x_i 2^i$ *and using the same notations as in Lemma 3.16 and Theorem 4.3, we have:*

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

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

Proof. The proof follows immediately from the results of lemma 3.16 and theorem 4.3, and from the construction of the binary decision `CMux` tree. In particular, the first `CMux` tree has depth $(d-\delta)$ and the blind rotation evaluates δ `CMux` gates, which brings a total factor d in the depth. As the `CMux` depth is the same as in horizontal packing, the noise propagation matches too.

Remark 4. As previously mentioned, the horizontal and vertical packing techniques can be mixed together to improve the evaluation of f. This combination is optimal in the case where s and d are both small or if $2^d \cdot s > N$. In particular, if we pack $x = s$ coefficients horizontally and $y = N/x$ coefficients vertically, we need $\lceil 2^d/y \rceil - 1$ `CMux` gates plus one vertical packing LUT evaluation in order to evaluate f, which is equivalent to $\log_2(y)$ `CMux` evaluations. The result is composed of the first x TLWE samples extracted. A practical example of the combination of the two techniques is given in Section 5.3.

5.2 Deterministic automata

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

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

In this section we present deterministic Weighted Finite Automata (det-WFA), a generalization of deterministic Finite Automata (det-FA) obtained by adding a weight in each transition. We detail the use of det-WFA to evaluate some arithmetic functions largely used in applications, such as addition (and multi-addition), multiplication, squaring, comparison and max, etc. We refer to [12] and [23] for further details.

Definition 5.3 (Deterministic weighted finite automata (det-WFA)). *A deterministic weighted finite automata (det-WFA) over a group $(S, \oplus)$ is a tuple $\mathfrak{A} = (Q, i, \Sigma, \mathcal{T})$, where Q is a finite set of states, i is the initial state, Σ is the alphabet, $\mathcal{T} \subseteq Q \times \Sigma \times S \times Q$ is the set of transitions. Every transition itself is a tuple $t = q \xrightarrow{\sigma,\nu} q'$ from the state q to the state q' by reading the letter σ with weight $w(t)$ equal to ν, and there is at most one transition per every pair (q, σ).*

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

Remark 5. In our applications, we fix the alphabet $\Sigma = \mathbb{B}$. Definition 5.3 restraints the WFA to the deterministic complete accessible (the non-deterministic case is not supported), and universally accepting case (i.e all the words are accepted). In the general (non-deterministic) case, the additive group would be replaced by the second law of a semi-ring $(S, \bullet, \oplus)$, and we would sum the weights, using the first law, of all accepting paths. However, since non-determinism is not supported, we want to keep the definition as simple as possible. In the rest of the paper we set $(S, \oplus)$ as $(\mathbb{T}_N[X], +)$.

Theorem 5.4 (Evaluation of det-WFA). *Let $\mathfrak{A} = (Q, i, \mathbb{B}, \mathcal{T})$ be a det-WFA with weights in $(\mathbb{T}_N[X], +)$, and let $|Q|$ denote the total number of states. Let $C_0, \ldots, C_{d-1}$ be d valid TRGSW_K samples of the bits of a word $\boldsymbol{\sigma} = \sigma_0 \ldots \sigma_{d-1}$. By evaluating at most $d \cdot |Q|$ `CMux` gates, Algorithm 6 outputs a TRLWE sample $\boldsymbol{d}$ that encrypts the weight $w(\boldsymbol{\sigma})$, such that (using the same notations as in lemma 3.16)*

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

where $\mathcal{A}_{\mathsf{TRGSW}}$ is an upper bound on the infinite norm of the error in the TRGSW samples and $\vartheta_{\mathsf{TRGSW}}$ is an upper bound of their variance. Moreover, if all the words connecting the initial state to a fixed state $q \in Q$ have the same length, then the upper bound on the number of `CMux` evaluated by Algorithm 6 decreases to $|Q|$.

Algorithm 6 Evaluation of a det-WFA

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

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

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

Proof. Let $q \in Q$ be a state, and $\boldsymbol{\sigma}$ a binary word, there exists a unique path starting from q and labelled by $\boldsymbol{\sigma}$. We note $w(q, \boldsymbol{\sigma})$ the weight of this path. Algorithm 6 evaluates the weights backwards from the last letter σ_{d-1} of the word to the first one. The invariant of the the main loop is that for all j in $[0, d]$ and $q \in Q$, if q is accessible at depth j, $c_{j,q}$ is a TRLWE sample of $w(q, (\sigma_j, \dots, \sigma_{d-1}))$. Its error amplitude satisfies

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

and its error variance satisfies

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

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

$\mathcal{T}_1(q)$ denote the states that are reached when reading a binary letter from the state q, and $w_0(q)$

$q \xrightarrow{0,w_0(q)} \mathcal{T}_0(q)$ and $q \xrightarrow{1,w_1(q)} \mathcal{T}_1(q)$. If q is accessible at depth j, then $\mathcal{T}_0(q)$ and $\mathcal{T}_1(q)$ are both accessible at depth $j+1$, and encode respectively $w(\mathcal{T}_0(q), \sigma_{j+1} \dots \sigma_{d-1})$ and $w(\mathcal{T}_1(q), \sigma_{j+1} \dots \sigma_{d-1})$. Therefore, after applying the $\mathtt{CMux}$, the message of $c_{j,q}$ is $w(q, \sigma_j \dots \sigma_{d-1})$. By applying the noise propagation inequalities of Lemma 3.16, we have that

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

in the worst case, and

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

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

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

Remark 6. This algorithm can also evaluate regular Deterministic Finite Automata (det-FA): in this case the weight of all transitions at depth $d-1$ that reach a final state is 0.5, and all other weights are 0.

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

Max In order to evaluate the Max function of two d-bit integers $x = \sum_{i=0}^{d-1} x_i 2^i$ and $y = \sum_{i=0}^{d-1} y_i 2^i$, with $x_i, y_i \in \mathbb{B}$ for $i \in [\![0, d-1]\!]$, we construct a det-WFA that takes in input all the bits of x and y, and outputs the maximal value between them. The idea is to enumerate the x_i and y_i, starting from the most significant bits down to the least significant ones. The det-WFA described in Figure 4 has 3 principal states (noted A, B, E) and 4 intermediary states (noted (A), (B), $(E,1)$, $(E,0)$), that keep track of which number is the maximum, and in case of equality what is the last value of x_i. A weight $+\frac{1}{2}X^i$ is added on all the transitions that reads the digit 1 from the maximum.

Remark 7. In practice, to evaluate the MAX function, we convert the det-WFA in a circuit that counts $5d$ `CMux` gates. Roughly speaking, we have to read the automata in the reverse. We initialize 5 states A, B, E_0, E_1, E as null TRLWE samples. Then, for i from $d-1$ to 0, we update the states as follows:

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

Here the C_i^x and C_i^y are TRGSW encryptions of the bits x_i and y_i respectively, and they are the inputs. The output of the evaluation is the TRLWE sample E, which contains the maximal value.

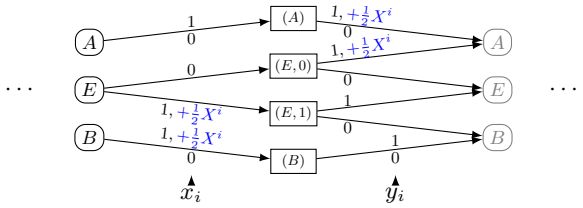


Fig. 4. Max: det-WFA - The states A and (A) mean that y is the maximal value, the states B and (B) mean that x is the maximal value, and finally, the states E, $(E,1)$ and $(E,0)$ mean that x and y are equals on the most significant bits. If the current state is A or B, the following state remains the same. The initial state is E. If the current state is E, after reading x_i there are two possible intermediate states: $(E,1)$ if $x_i = 1$ and $(E,0)$ if $x_i = 0$. After reading the value of y_i, the 3 possible states A, B and E are possible. The det-WFA is repeated as many times as the bit length of the integers evaluated and the weights are given in clear.

Overall, the next lemma, which is a direct consequence of Theorem 5.4, shows that the Max can be computed by evaluating only $5d$ `CMux` gates, instead of $\Theta(d^2)$ with classical deterministic automata.

Lemma 5.5 (Evaluation of Max det-WFA). *Let $\mathfrak{A}$ be the det-WFA of the Max, described in Figure 4. Let $C_0^x, \ldots, C_{d-1}^x, C_0^y, \ldots, C_{d-1}^y$ be TRGSW_K samples of the bits of x and y respectively. By evaluating $5d$ `CMux` gates (depth $2d$), the Max det-WFA outputs a TRLWE sample $\boldsymbol{d}$ encrypting the maximal value between x and y and (with same notations as in lemma 3.16)*

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

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

In a similar way, it is possible to construct a det-WFA for the **comparison** between two integers x and y, answering is $x > y$ or not. The result is computed by evaluating $4d$ `CMux` gates (and just $3d$ gates in the case of gate bootstrapping).

Multiplication For the multiplication we use the same approach and we construct a det-WFA which maps the schoolbook multiplication. We illustrate the construction on the example of the multiplication between two 2-bits integers $x = x_1x_0$ and $y = y_1y_0$. As shown in the top part of Figure 5, after an initial step of bit by bit multiplication, a multi-addition (shifted of one place on the left for every line) is performed. The bits of the final result $m = m_3m_2m_1m_0$ are computed as the sum of each column with carry.
The det-WFA computes the multiplication by keeping track of the partial sum of each column in the states, and by using the transitions to update these sums.

For the multiplication of 2-bits integers, the automaton (described in the bottom part of Figure 5) has 6 main states (i, c_0, c_{10}, c_{11}, c_{20}, c_{21}), plus 14 intermediary states (noted with capital letters and parenthesis) that store the last bit read. The value of the j-th (for $j \in [\![0,3]\!]$) output bit is put in a weight on the last transition of each column. The final weight is the result of the multiplication.

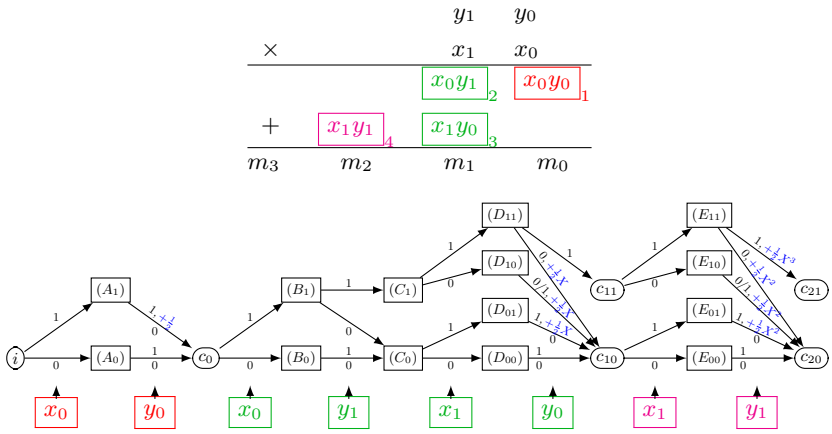


Fig. 5. Schoolbook 2-bits multiplication and corresponding det-WFA

For the generic multiplication of two d-bits integers, we can upper bound the number of states by $4d^3$, instead of $\Theta(d^4)$ with one classical automata per output bit. For a more precise number of states we wrote a C++ program to eliminate unreachable states and refine the leading coefficient. The depth is $2d^2$ and the noise evaluation can be easily deducted by previous results. The same principle can be used to construct the **multi-addition**, and its det-WFA is slightly simpler (one transition per bit in the sum instead of two).

5.3 Bit Sequence Representation

We now present another design which is specific to the multi-addition (or its derivatives), but which is faster than the generic construction with det-WFA. The idea is to build an homomorphic scheme that can represent small integers, say between 0 and $N = 2^p$, and which is dedicated to only the three elementary operations used in the multi-addition algorithm, namely:

1. Extract any of the bits of the value as a TLWE sample;
2. Increment the value by 1;
3. Integer division of the value by 2.

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

We represent integers modulo $2N$, with $N = 2^p$, by using their Bit Sequence Representation (BSR). For $j \in [\![0, p]\!]$ and $k, l \in \mathbb{Z}$, we call $B_{j,k}^{(l)}$ the j-th bit of $l + k$ in the little endian signed binary representation.

For each integer $k \in \mathbb{Z}$, $(B_{0,k}^{(l)}, B_{1,k}^{(l)}, ..., B_{p,k}^{(l)})$ is the (little endian signed) binary representation of $l + k \mod 2N$. When k isn't specified, $B_j^{(l)}$ represents the binary sequence of all the j-th bits of integers $l, l + 1, l + 2, \ldots$.

Let $l = 0$, observe that $B_0^{(0)} = (0, 1, 0, 1, ...)$ is 2-periodic, $B_1^{(0)} = (0, 0, 1, 1, 0, 0, 1, 1...)$ is 4-periodic and, more generally, for all $j \in [\![0, p]\!]$ and $l \in \mathbb{Z}$, $B_j^{(l)}$ is 2^j-antiperiodic and it is the left shift of $B_j^{(0)}$ by l positions. Therefore, it suffices to have $2^j \leq N$ consecutive values of the sequence to (blindly) deduce all the remaining bits.

We now suppose that an integer $l \in [\![0, N - 1]\!]$ is represented by its BSR, defined as $BSR(l) = [B_0^{(l)}, \ldots, B_p^{(l)}]$.

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

Fig. 6. BSR - The figure represents the $BSR(l)$. Here $l = 0$, $N = 2^3$. The first column corresponding to $k = 0$ contains the bits of l in the little endian representation.

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

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

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

More generally, we can increment the BSR by any integer in $[\![0, N - 1]\!]$, as in Figure 7.

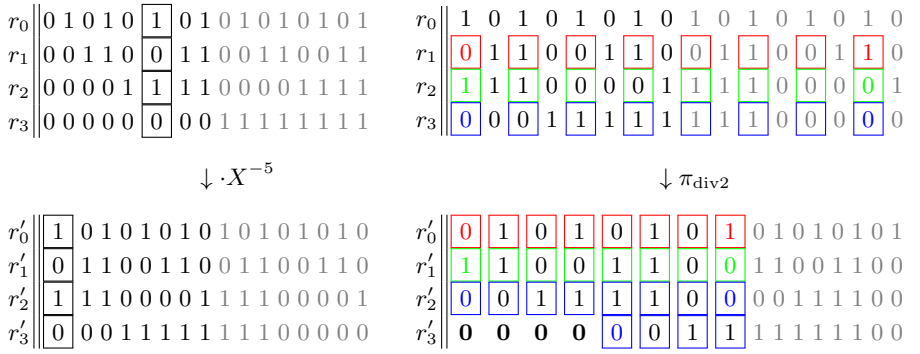


Fig. 7. TBSR - example of addition +5 and division by 2.

Integer division by two. Let $BSR(l) = [B_0^{(l)}, \dots, B_p^{(l)}]$ be the BSR of some unknown number $l \in [\![0, N-1]\!]$. Our goal is to compute the BSR of $\lfloor \frac{l}{2} \rfloor$, $BSR(\lfloor \frac{l}{2} \rfloor) = [B_0^{(\lfloor \frac{l}{2} \rfloor)}, \dots, B_p^{(\lfloor \frac{l}{2} \rfloor)}]$.

As the BSR is the representation of integers in base 2, when we want to perform the division by 2, it corresponds to eliminate the first line and keep just the odd columns of the BSR (see Figure 6). Thus we can set

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

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

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

TBSR We now explain how we can encode these BSR sequences on TRLWE ciphertexts, considering that all the coefficients need to be in the torus rather than in $\mathbb{B}$, and that we need to encode sequences that are either N-periodic or N-antiperiodic. We note the encoded BSR by TBSR.

Therefore, this is our basic encoding of the BSR sequences: let $BSR(l) = [B_0^{(l)}, \ldots, B_p^{(l)}]$ be the BSR of some unknown number $l \in [\![0, N-1]\!]$. For $j \in [\![0, p-1]\!]$, we represent $B_j^{(l)}$ with the polynomial

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

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

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

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

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

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

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

Finally, we call TBSR ciphertext of an unknown integer $l \in [0, N-1]$ a vector $C = [c_0, ..., c_p]$ of TRLWE ciphertexts of message $[\mu_0, \ldots, \mu_p]$.

Definition 5.6 (TBSR encryption). *We define the TBSR encryption as follows.*

- *Parameters and keys:* TRLWE *parameter N with secret key $K \in \mathbb{B}_N[X]$, and a circular-secure keyswitching key $\mathsf{KS}_{K \to K, \gamma}$ from K to itself, noted just* KS.
- $\mathsf{TBSRSet}(l)$*: return a vector of trivial* TRLWE *ciphertexts encoding the torus representation of $[B_0^{(l)}, \ldots, B_p^{(l)}]$.*
- $\mathsf{TBSREncrypt}(l)$*: return a vector of non-trivial* TRLWE *ciphertexts encoding the torus representation of $[B_0^{(l)}, \ldots, B_p^{(l)}]$.*
- $\mathsf{TBSRBitExtract}_j(C)$*: Return* $\mathsf{SampleExtract}(c_j)$ *(Section 4.2) when $j < p$.* [12]
- $\mathsf{TBSRIncrement}(C)$*: Return $X^{-1} \cdot C$.*

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

– $\mathsf{TBSRDiv2}(C)$: *Use* KS *to evaluate* π_{div2} *homomorphically on* C*. Since it is a 1-lipschitzian affine function, this consists in applying the public functional* $\mathsf{KeySwitch}$ *to* KS*, the linear part of* π_{div2} *and* C*, and then, translate the result by the constant part of* π_{div2}*.*

The following theorem is a direct consequence of Theorem 4.1 (with $n = N$). The correctness of the result has already been discussed.

Theorem 5.7 (TBSR operations). *Let* N*,* K *and* KS *be TBSR parameters and keys as defined before, and let* C *be a TBSR ciphertext of* $l \in [\![0, N-1]\!]$ *with noise amplitude* η *(and noise variance* ϑ*). Then for* $j \in [\![0, p-1]\!]$*,* $\mathsf{TBSRBitExtract}_j(C)$ *is a* $\mathsf{TLWE}_{\mathfrak{K}}$ *ciphertext of the* j*-th bit of* l*, over the message space* $\{0, \frac{1}{2}\}$*, with noise amplitude (resp. variance)* $\leq \eta$ *(resp.* $\leq \vartheta$*). If* $l \leq N-2$ *(if* $l = N-1$ *the result is not determined),* $\mathsf{TBSRIncrement}(C)$ *is a TBSR ciphertext of* $l+1$ *with noise amplitude (resp. variance)* $\leq \eta$ *(resp.* $\leq \vartheta$*).* $C' = \mathsf{TBSRDiv2}(C)$ *is a TBSR ciphertext of* $\lfloor l/2 \rfloor$ *such that:*

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

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

Using the TBSR counter for a multi-addition or a multiplication. The TBSR counter allows to perform a multi-addition or multiplication using the school-book elementary algorithms (see Algorithm 7 and Algorithm 8). This leads to a leveled multiplication circuit (with $\mathsf{KeySwitching}$) which is quadratic instead of cubic with weighted automata.

The following lemma analyzes the case of the multiplication and the result described is a consequence of Theorem 4.1. The correctness of the result can be deducted from the construction, presented in the Algorithm 8.

The formulas for the multi-addition can be easily found, and the correctness comes from the construction (Algorithm 7).

Lemma 5.8. *Let* N*,*B_g*,*ℓ *and* KS *be TBSR and* TRGSW *parameters with the same key* K*. We suppose that each TBSR ciphertext has* $p \leq 1 + \log(N)$ TRLWE *ciphertexts. Let* (A_i) *and* (B_i) *for* $i \in [0, d-1]$ *be* TRGSW*-encryptions of the bits of two* d*-bits integers (little endian), with the same noise amplitude* $\mathcal{A}$ *(resp. variance* ϑ*).*

Then, Algorithm 8 computes all the bits of the product within $2d^2 p$ `CMux` *and* $(2d-2)p$ *public key-switching, and the output ciphertexts satisfy:*

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

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

Algorithm 7 TBSR multi-addition

Input: The bitwise TRGSW-encryption $(A_{i,j})_{i\in[0,m-1],j\in[0,d-1]}$ of m integers $a_0,\ldots,a_{m-1}$ of d-bits each

Output: bitwise LWE encryptions of the sum $\mathrm{Out}_0,\ldots,\mathrm{Out}_d + log_2(d)$.

$C \leftarrow \mathsf{TBSRSet}(0)$
for $j = 0$ to $d-2$ **do**
  for $i = 0$ to $m-1$ **do** ▷ Sum the j-th column
    $C \leftarrow \mathsf{CMux}(A_{j,i}, \mathsf{TBSRIncr}(C), C)$;
  end for
  $\mathrm{Out}_i \leftarrow \mathsf{TBSRBitExtract_0}(C)$; ▷ Extract Out_i = the lsb
  $C \leftarrow \mathsf{TBSRDiv2}(C)$; ▷ and compute the carry
end for
for $i = 0$ to $m-1$ **do** ▷ Sum the last column
  $C \leftarrow \mathsf{CMux}(A_{d,i}, \mathsf{TBSRIncr}(C), C)$;
end for
$\mathrm{Out}_{d-1+k} \leftarrow \mathsf{TBSRBitExtract_k}(C)$ **for each** $k \in [0, \log_2(2d)]$;
return Out ▷ and output all the bits

Improving TBSR with horizontal packing.

First improvement: If the domain of the integers is $[0, y-1]$ where $xy = N$, we can use Horizontal packing to pack x different polynomials mod X^y+1 in a single TRLWE ciphertext mod X^N+1. We just replace the shift by X^{-1} by a multiplication by X^{-x}. This allows to store the $p = \log_2(y)+1$ sequences in only $\lceil p/x \rceil$ TRLWE ciphertexts, and this provides a factor x speedup compared to the basic scheme. For instance, if $N = 1024$ and the domain of the BSR integers is $[0, 127]$, which is enough to perform a multiplication with a 64 bit number, the 8 sequences can be packed in a single TRLWE ciphertext ($x = 8$,$y = 128$).

Second improvement: Even if the domain is as large as $[0, N-1]$, the first bit sequences have a small period. It is therefore possible to use the previous improvement to encode many of the first sequences as a single ciphertext, and leave the last N-antiperiodic one alone on its ciphertext. For instance, if $N = 1024$, the first 8 sequences $B_0^{(l)},\ldots,B_7^{(l)}$ are 128-periodic or 128-antiperiodic. They can be packed on a single TRLWE ciphertext. The next two sequences $B_8^{(l)}, B_9^{(l)}$ are 512-periodic/antiperiodic, and can be packed on a single TRLWE ciphertext. Finally, the lase sequence $B_1^{(l)}0$ is 1024-antiperiodic, and stays alone. As long as periodic sequences use the $\{0, \frac{1}{2}\}$ message space, and anti-periodic sequences use $\{-\frac{1}{4}, \frac{1}{4}\}$ (and the constant terms of f_{div2} are updated accordingly), all TBSR computations over $[0, 1023]$ can be done in only 3 TRLWE ciphertexts instead of 11, which gives a time-speedup of a factor $11/3 = 3.66$ compared to the basic scheme.

Algorithm 8 TBSR multiplication

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

Output: bitwise LWE encryptions of the sum $\mathrm{Out}_0, \ldots, \mathrm{Out}_d + log_2(d)$.

$C \leftarrow \mathsf{TBSRSet}(0)$
for $j = 0$ to $d-1$ **do**
  for $i = 0$ to j **do**
    $C \leftarrow \mathsf{CMux}(A_i, \mathsf{CMux}(B_{j-i}, \mathsf{TBSRIncr}(C), C), C)$;
  end for
  $\mathrm{Out}_j \leftarrow \mathsf{TBSRBitExtract}_0(C)$;
  $C \leftarrow \mathsf{TBSRDiv2}(C)$;
end for
for $j = d$ to $2d-3$ **do**
  for $i = j-d+1$ to $d-1$ **do**
    $C \leftarrow \mathsf{CMux}(A_i, \mathsf{CMux}(B_{j-i}, \mathsf{TBSRIncr}(C), C), C)$;
  end for
  $\mathrm{Out}_j \leftarrow \mathsf{TBSRBitExtract}_0(C)$;
  $C \leftarrow \mathsf{TBSRDiv2}(C)$;
end for
$C \leftarrow \mathsf{CMux}(A_{d-1}, \mathsf{CMux}(B_{d-1}, \mathsf{TBSRIncr}(C), C), C)$;
$\mathrm{Out}_{2d-2} \leftarrow \mathsf{TBSRBitExtract}_0(C)$;
$\mathrm{Out}_{2d-1} \leftarrow \mathsf{TBSRBitExtract}_1(C)$;
return *Out*

6 Bootstrapping

The schemes and the techniques described in the previous sections can be used in a leveled context, where the depth of the circuit to be evaluated is known in advance. We now present two bootstrappings. The fastest one, in the lineage of [28] and [24] must be performed after each gate in a circuit, and is therefore denoted as *gate-bootstrapping*. The second one, which we describe in Subsection 6.2, allows to execute a larger leveled circuit between each bootstrapping, and is is therefore denoted *circuit bootstrapping*.

6.1 Gate bootstrapping (TLWE-to-TLWE)

Given a TLWE sample $\mathsf{TLWE}_{\mathfrak{K}}(\mu) = (\mathfrak{a}, \mathfrak{b})$, the gate bootstrapping procedure constructs an encryption of μ under the same key $\mathfrak{K}$ but with a fixed amount of noise. As in [24], we use TRLWE as an intermediate encryption scheme to homomorphically evaluate the phase, but we use the external product from theorem 3.13 with a TRGSW encryption of the key $\mathfrak{K}$.

Definition 6.1 (Bootstrapping key). *Let $\mathfrak{K} \in \mathbb{B}^n$, $\mathfrak{K}' \in \mathbb{B}_N[X]^{n'}$ and α be a standard deviation. We define the bootstrapping key $BK_{\mathfrak{K}\to\mathfrak{K}',\alpha}$ as the sequence of n TGSW samples where $BK_i \in \mathsf{TRGSW}_{\mathfrak{K}',\alpha}(\mathfrak{K}_i)$.*

Algorithm 9 Bootstrapping TLWE-to-TLWE (calling algorithm 4)

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

Theorem 6.2 (Bootstrapping TLWE-to-TLWE). *Let $\bar{H}$ be the gadget matrix in $\mathcal{M}_{(\bar{k}+1)\bar{\ell},\bar{k}+1}(\mathbb{T}_{\bar{N}}[X])$ and $Dec_{\bar{H},\bar{\beta},\bar{\epsilon}}$ its efficient approximate gadget decomposition algorithm, with quality $\bar{\beta}$ and precision $\bar{\epsilon}$ defining* TRLWE *and* TRGSW *parameters. Let $\underline{\mathfrak{K}} \in \mathbb{B}^{\underline{n}}$ and $\bar{\mathfrak{K}} \in \mathbb{B}^{\bar{n}}$ be two* TLWE *secret keys, and $\bar{K} \in \mathbb{B}_{\bar{N}}[X]^{\bar{k}}$ be the* TRLWE *interpretation of the key $\bar{\mathfrak{K}}$, and let $\bar{\alpha} \in \mathbb{R}_{\geq 0}$ be a standard deviation. Let $BK_{\underline{\mathfrak{K}} \to \bar{\mathfrak{K}},\bar{\alpha}}$ be a bootstrapping key, composed by the $\underline{n}$* TRGSW *encryptions $BK_i \in \mathsf{TRGSW}_{\bar{K},\bar{\alpha}}(\underline{\mathfrak{K}}_i)$ for $i \in [\![1, \underline{n}]\!]$. Given one constant $\mu_1 \in \mathbb{T}$, and one sample $\underline{\mathfrak{c}} \in \mathbb{T}^{\underline{n}+1}$ whose coefficients are all multiples of $\frac{1}{2\bar{N}}$, Algorithm 9 outputs a* TLWE *sample $\bar{\mathfrak{c}} \in \mathsf{TLWE}_{\bar{\mathfrak{K}}}(\mu)$ where $\mu = 0$ iff. $|\varphi_{\underline{\mathfrak{K}}}(\underline{\mathfrak{c}})| < \frac{1}{4}$, $\mu = \mu_1$ otherwise and such that:*

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

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

Proof. By using the definitions of $\tilde{b}$ and $\tilde{\boldsymbol{a}}$ given at line 2 of the Algorithm 9, we define $\tilde{\varphi} \stackrel{def}{=} \tilde{b} - \sum_{i=1}^{\underline{n}} \tilde{a}_i s_i \mod 2N$. We have

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

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

At line 3, the test vector $v := (1 + X + \ldots + X^{\bar{N}-1}) \cdot X^{\frac{\bar{N}}{2}} \cdot \mu$ is defined such that for all $p \in [0, 2N]$, the constant term of $X^p \cdot v$ is either μ if $p \in]\!]\frac{\bar{N}}{2}, \frac{3\bar{N}}{2}[\![$, and $-\mu$ otherwise.

At line 4, a blind rotation (Algorithm 4) is applied to the test vector. The result is $\mathsf{msg}(ACC) = X^{-\tilde{\varphi}} \cdot v$ and the error (from the results shown in Theorem 4.3 and as the TRLWE encryption of the test vector is noiseless trivial) is:

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

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

The SampleExtract at line 5 doesn't add any noise, so the error after bootstrapping remains the same as after the blind rotation.

After the blind rotation, the message in the accumulator is $\mathsf{msg}(ACC) = X^{-\tilde{\varphi}} \cdot v$. After SampleExtract, the message is equal to the constant term of $\mathsf{msg}(ACC)$, i.e. $-\mu$ if $\tilde{\varphi} \in]\frac{\bar{N}}{2}, \frac{3\bar{N}}{2}]$, and μ otherwise. The addition with $(\mathbf{0}, \mu)$ makes the message equal to 0 if $\tilde{\varphi} \in]\frac{\bar{N}}{2}, \frac{3\bar{N}}{2}]$, and equal to μ_1 otherwise.

In other words, $|\varphi_{\mathfrak{K}}(\mathfrak{a}, \mathfrak{b})| < 1/4 - \delta$, then $-1/4 + \delta \leq \varphi_{\mathfrak{K}}(\mathfrak{a}, \mathfrak{b}) < 1/4 - \delta$, and thus using Equation (2), we obtain that $\tilde{\varphi} \in]-\frac{N}{2}, \frac{N}{2}[$ and thus, the message is equal to μ_1. And if $|\varphi_{\mathfrak{K}}(\mathfrak{a}, \mathfrak{b})| > 1/4 + \delta$ then $\varphi_{\mathfrak{K}}(\mathfrak{a}, \mathfrak{b}) > 1/4 + \delta$ or $\varphi_{\mathfrak{K}}(\mathfrak{a}, \mathfrak{b}) < -1/4 - \delta$ and using Equation (2), we obtain that the message is equal to 0. □

We first provide a comparison between the bootstrapping of Algorithm 9, the Algorithm 1, 2 in [24] and the Algorithm 3 in [17].

– Like [24] and [17], we rescale the computation of the phase of the input TLWE sample so that it is modulo $2N$ (line 2) and we map all the corresponding operations in the multiplicative cyclic group $\{1, X, \ldots, X^{2N-1}\}$. Since our TLWE samples are described over the real torus, the rescaling is done explicitly. This rescaling may induce a cumulated rounding error of amplitude at most $\delta \approx \sqrt{n}/4N$ in the average case and $\delta \leq (n+1)/4N$ in the worst case. In the best case, this amplitude can even be zero ($\delta = 0$) if in the actual representation of TLWE samples, all the coefficients are restricted to multiple of $\frac{1}{2N}$.
– As in [24] and [17], messages are encoded as roots of unity in $\mathcal{R}$. Our accumulator is a TRLWE sample (as in [17]) instead of a TRGSW sample (as in [24]). Also accumulator operations use the external product from Theorem 3.13 instead of the slower classical internal product. The test vector $(1+X+\ldots+X^{N-1})$ is embedded in the accumulator from the very start, when the accumulator is still noiseless while in [24], it is added at the very end. This removes a factor $\sqrt{N}$ to the final noise overhead.
– Instead of the explicit loop proposed in [24] and in [17], we directly use the blind rotation Algorithm 4. As in [17], all the TRGSW ciphertexts of $X^{-\tilde{a}_i \mathfrak{K}_i}$ required to update the accumulator internal value are computed dynamically as a very small polynomial combination of BK_i in the for loop of the Algorithm 4. This completely removes the need to decompose each $\tilde{a}_i$ on an additional base B_r, and to precompute all possibilities in the bootstrapping key. In other words, this makes our bootstrapping key 46 times smaller than in [24], for the exact same noise overhead. Besides, due to this squashing technique, two accumulator operations were performed per iteration instead of one in our case. This gives us an additional $2\times$ speed up. Also, a small difference in the way we associate `CMux` operations in Algorithm 4 removes a factor 2 in the noise compared to the previous gate bootstrapping procedure in [17], and it is also faster.

The Algorithm 9 takes in input a TLWE ciphertext, and depending on its phase, it outputs either a ciphertext of 0 or of μ with a noise amplitude that

Algorithm 10 Gate Bootstrapping (calling algorithms 9 and 2)

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

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

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

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

is independent on the input. However, the input and output ciphertexts are not encrypted with the same key, since both keys $\underline{\mathfrak{K}}$ and $\overline{\mathfrak{K}}$ have not the same parameters. The next elementary theorem fixes this by applying a key-switching (Theorem 4.1) at the end of the bootstrapping.

Theorem 6.3 (Gate Bootstrapping TLWE-to-TLWE). *Let $\bar{H}$ be the gadget matrix in $\mathcal{M}_{(\bar{k}+1)\bar{\ell},\bar{k}+1}(\mathbb{T}_{\bar{N}}[X])$ and $Dec_{\bar{H},\bar{\beta},\bar{\epsilon}}$ its efficient approximate gadget decomposition algorithm, with quality $\bar{\beta}$ and precision $\bar{\epsilon}$ defining* TRLWE *and* TRGSW *parameters. Let $\underline{\mathfrak{K}} \in \mathbb{B}^{\underline{n}}$ and $\bar{\mathfrak{K}} \in \mathbb{B}^{\bar{n}}$ be two* TLWE *secret keys, and $\bar{K} \in \mathbb{B}_{\bar{N}}[X]^{\bar{k}}$ be the* TRLWE *interpretation of the key $\bar{\mathfrak{K}}$, and let $\bar{\alpha} \in \mathbb{R}_{\geq 0}$ be a standard deviation. Let $BK_{\underline{\mathfrak{K}}\to\bar{\mathfrak{K}},\bar{\alpha}}$ be a bootstrapping key, composed by the $\underline{n}$* TRGSW *encryptions $BK_i \in \mathsf{TRGSW}_{\bar{K},\bar{\alpha}}(\underline{\mathfrak{K}}_i)$ for $i \in [\![1,\underline{n}]\!]$. Let $\mathsf{KS} = \mathsf{KS}_{\bar{\mathfrak{K}}\to\underline{\mathfrak{K}},\underline{\gamma},t} = \left(\mathsf{KS}_{i,j}^{(id)}\right)_{i,j}$, with $\mathsf{KS}_{i,j}^{(id)} \in \mathsf{TRLWE}_{\mathfrak{K}',\gamma}(\frac{\underline{\mathfrak{K}}_i}{2^j})$ be a key-switching key defined as in Theorem 4.2 (with the function f equal to the identity function $id : \mathbb{T} \to \mathbb{T}$). Given one constant $\mu_1 \in \mathbb{T}$, and one sample $\underline{\mathfrak{c}} \in \mathbb{T}^{\underline{n}+1}$ whose coefficients are all multiples of $\frac{1}{2\bar{N}}$, Algorithm 10 outputs a* TLWE *sample $\underline{\mathfrak{c}}' \in \mathsf{TLWE}_{\underline{\mathfrak{K}}}(\mu)$ where $\mu = 0$ iff. $|\varphi_{\underline{\mathfrak{K}}}(\underline{\mathfrak{c}})| < \frac{1}{4}$, $\mu = \mu_1$ otherwise and such that:*

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

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

Fully Homomorphic Boolean Gates. In [24], the homomorphic evaluation of a `NAND` gate between LWE samples is achieved with 2 additions (one with a noiseless trivial sample) and a gate bootstrapping (Algorithm 10).

We chose the parameters such that $\mathsf{Var}(\mathsf{Err}(\underline{\mathfrak{c}}')) < \frac{1}{16}$ and we denote as $\underline{\mathfrak{c}}' = \mathsf{Bootstrap}(\underline{\mathfrak{c}})$ the output of the gate bootstrapping (Algorithm 9, with $\mu_1 = \frac{1}{4}$) plus key-switching (Algorithm 3) procedures applied to $\underline{\mathfrak{c}}$.

Let consider two TLWE samples $\underline{\mathfrak{c}}_1$ and $\underline{\mathfrak{c}}_2$, with message space $\{0, 1/4\}$ and $\|\mathsf{Err}(\underline{\mathfrak{c}}_1)\|_\infty, \|\mathsf{Err}(\underline{\mathfrak{c}}_2)\|_\infty \leq \frac{1}{16}$. The result of the bootstrapped `NAND` gate is obtained by computing $\underline{\mathfrak{c}} = (\mathbf{0}, \frac{5}{8})$-$\underline{\mathfrak{c}}_1$-$\underline{\mathfrak{c}}_2$, plus a bootstrapping (gate bootstrapping

and key-switching). Indeed the possible values for the messages of $\underline{\mathbf{c}}$ are $\frac{5}{8}, \frac{3}{8}$ if either $\underline{\mathbf{c}}_1$ or $\underline{\mathbf{c}}_2$ encode 0, and $\frac{1}{8}$ if both encode $\frac{1}{4}$. Since the noise amplitude $\|\mathsf{Err}(\underline{\mathbf{c}})\|_\infty$ is $< \frac{1}{8}$, then $|\varphi_{\underline{\mathfrak{K}}}(\underline{\mathbf{c}})| > \frac{1}{4}$ iff. $\mathrm{NAND}(\mathsf{msg}(\underline{\mathbf{c}}_1), \mathsf{msg}(\underline{\mathbf{c}}_)) = 1$. This explains why it suffices to bootstrap $\underline{\mathbf{c}}$ with parameters $\mu_1 = \frac{1}{4}$ to get the answer.

By using a similar approach, it is possible to directly evaluate with a single bootstrapping all the basic gates:

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

The $\mathrm{HomXOR}(\underline{\mathbf{c}}_1, \underline{\mathbf{c}}_2)$ gate can be achieved also by performing $\mathsf{Bootstrap}\left(2 \cdot (\underline{\mathbf{c}}_1+\underline{\mathbf{c}}_2)\right)$.

Remark 8. The term *gate bootstrapping* refers to the fact that this fast bootstrapping is performed after every gate evaluation, but it can be used even if we do not need to evaluate a specific gate and we just want to refresh noisy ciphertexts.

The ternary `MUX` gate ($\mathtt{MUX}(c, d_0, d_1) = c?d_1 : d_0 = (c \wedge d_1) \oplus ((1-c) \wedge d_0)$, for $c, d_0, d_1 \in \mathbb{B}$) is generally expressed as a combination of 3 binary gates. As already mentioned in [24], we can improve the `MUX` evaluation by performing the middle $\oplus$ as a regular addition before the final $\mathsf{KeySwitching}$. Indeed, this xor has at most one operand which is true, and at this location, it only affects a negligible amount of the final noise. Overall, the ternary `MUX` gate can be evaluated in FHE mode by evaluating only two gate bootstrappings and one public key-switching. We call this procedure *native* `MUX`, and we note it HomMUX as the other bootstrapped homomorphic gates, which computes:

- $c \wedge d_1$ via a gate bootstrapping (Algorithm 9) of $(\mathbf{0}, -\frac{1}{8}) + \underline{\mathbf{c}} + \underline{\mathfrak{d}}_1$;
- $(1-c) \wedge d_0$ via a gate bootstrapping (Algorithm 9) of $(\mathbf{0}, \frac{1}{8}) - \underline{\mathbf{c}} + \underline{\mathfrak{d}}_0$;
- a final public key-switching (Algorithm 2) on the sum, which dominates the noise.

This HomMUX is therefore bootstrappable with the same parameters for the other binary gates. In the rest of the paper, when we compare different homomorphic techniques, we refer to the gate bootstrapping mode as the technique consisting in evaluating small circuits expressed by using this bootstrapped gates.

6.2 Circuit bootstrapping (TLWE-to-TRGSW)

In the previous sections, we presented efficient leveled algorithms for some arithmetic operations, but the input and output have different types (e.g. TLWE/TRGSW) and we can't compose these operations, like in a usual algorithm. In fully homomorphic mode, connecting the two becomes possible if we

have an efficient bootstrapping between TLWE and TRGSW ciphertexts. Fast bootstrapping procedures have been proposed in [24, 17], and the external product 3.13 from [17, 10] has contributed to accelerate leveled operations. Unfortunately, these bootstrapping cannot output GSW ciphertexts. Previous solutions proposed in [29, 5, 26] based on the internal product are not practical. In this section, we propose an efficient technique to convert back TLWE ciphertexts to TRGSW, that runs in 137ms. We call it *circuit bootstrapping*.

Our goal is to convert a TLWE sample with large noise amplitude over some binary message space (e.g amplitude $\frac{1}{4}$ over $\{0, \frac{1}{2}\}$), into a TRGSW sample with a low noise amplitude $< 2^{-20}$ over the integer message space $\{0, 1\}$.

In all previous constructions, the TLWE decryption consists in a circuit, which is then evaluated using the internal addition and multiplication laws over TRGSW ciphertexts. The target TRGSW ciphertext is thus the result of an arithmetic expression over TRGSW ciphertexts. Instead, we propose a more efficient technique, which reconstructs the target directly from its very sparse internal structure. Namely, a TRGSW ciphertext of a message $\mu \in \{0, 1\}$ is a vector of $(k + 1)\ell$ TRLWE ciphertexts. Each of these TRLWE ciphertexts encrypts the same message as $\mu \cdot \boldsymbol{h}_i$, where $\boldsymbol{h}_i$ is the corresponding line of the gadget matrix H. Depending on the position of the row (which can be indexed by $u \in [1, k + 1]$ and $j \in [1, \ell]$), this message is $\mu - K_u \cdot Bg^{-j}$ where K_u is the u-th polynomial of the secret key and $K_{k+1} = -1$. So we can use ℓ times the TLWE-to-TLWE bootstrapping of [17] to obtain a TLWE sample of each message in $\{\mu Bg^{-1}, \ldots, \mu B_g^{-\ell}\}$. Then we use the private key-switching technique to "multiply" these ciphertexts by the secret $-K_u$, to reconstruct the correct message.

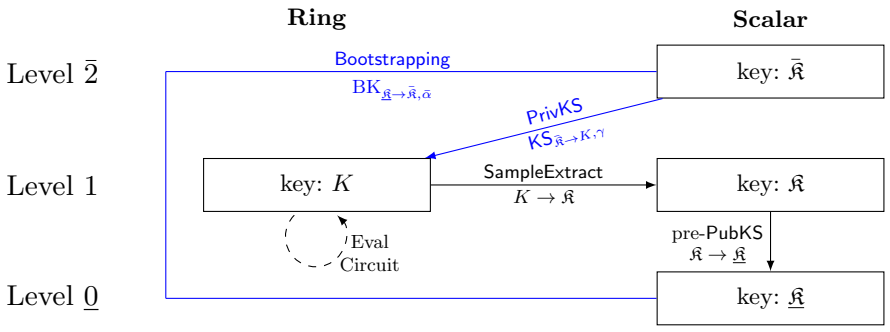


Fig. 8. The figure represents the three levels of encryption on which our construction shifts. The arrows show the operations that can be performed inside each level or how to move from a level to another. In order to distinguish the objects with respect to their level, we adopted the intuitive notations "superior bar" for level 2, "no bar" for level 1 and "under bar" for level 0. We highlight in blue the different stages of the circuit bootstrapping (whose detailed description is given below).

Our circuit bootstrapping, detailed in Algorithm 11, crosses 3 levels of noise and encryption. Each level has its own key and parameters set. In order to distinguish the different levels, we use an intuitive notation with bars. The upper bar will be used for level 2 variables, the under bar for the level 0 variables and level 1 variables will remain without any bar. The main difference between the three levels of encryption is the amount of noise supported. Indeed, the higher the level is, the smaller is the noise. Level 0 corresponds to ciphertexts with very large noise (typically, $\underline{\alpha} \approx 2^{-11}$). Level 0 parameters are very small, computations are almost instantaneous, but only a very limited amount of linear operations are tolerated. Level 1 corresponds to medium noise (typically, $\alpha \approx 2^{-30}$). Ciphertexts in level 1 have medium size parameters, which allows for relatively fast operations, and for instance a leveled homomorphic evaluation of a relatively large automata, with transition timings described in Section 5 of [17]. Level 2 corresponds to ciphertexts with small noise (typically, $\bar{\alpha} \approx 2^{-50}$). This level corresponds to the limit of what can be mapped over native 64-bit operations. Practical values and details are given in Section 8.

Our circuit bootstrapping consists in three parts:

1. TLWE-to-TLWE **Pre-keyswitch**: The input of the algorithm is a TLWE sample with a large noise amplitude over the message space $\{0, \frac{1}{2}\}$. Without loss of generality, it can be keyswitched to a level 0 TLWE ciphertext $\underline{\mathbf{c}} = (\underline{\mathbf{a}}, \underline{b}) \in \mathsf{TLWE}_{\underline{\mathfrak{K}},\underline{\eta}}(\mu \cdot \frac{1}{2})$, of a message $\mu \in \mathbb{B}$ with respect to the small secret key $\underline{\mathfrak{K}} \in \mathbb{B}^{\underline{n}}$ and a large standard deviation $\underline{\eta} \in \mathbb{R}$ (typically, $\underline{\eta} \leq 2^{-5}$ to guaranty correct decryption with overwhelming probability). This step is standard.
2. TLWE-to-TLWE **Bootstrapping** (Algorithm 9): Given a level 2 bootstrapping key $\mathrm{BK}_{\underline{\mathfrak{K}} \to \bar{\mathfrak{K}},\bar{\alpha}} = (\mathrm{BK}_i)_{i \in [\![1,\underline{n}]\!]}$ where $\mathrm{BK}_i \in \mathsf{TRGSW}_{\bar{K},\bar{\alpha}}(\underline{\mathfrak{K}}_i)$), we use ℓ times the TLWE-to-TLWE Bootstrapping algorithm (algorithm 9) on $\underline{\mathbf{c}}$, to obtain ℓ TLWE ciphertexts $\bar{\mathbf{c}}^{(1)}, \ldots, \bar{\mathbf{c}}^{(\ell)}$ where $\bar{\mathbf{c}}^{(w)} \in \mathsf{TLWE}_{\bar{\mathfrak{K}},\bar{\eta}}(\mu \cdot \frac{1}{B_g{}^w})$, with respect to the same level 2 secret key $\bar{\mathfrak{K}} \in \mathbb{B}^{\bar{n}}$, and with a fixed noise parameter $\bar{\eta} \in \mathbb{R}$ which does not depend on the input noise. If the bootstrapping key has a level 2 noise $\bar{\alpha}$, we expect the output noise $\bar{\eta}$ to remain smaller than level 1 value.
3. TLWE-to-TRLWE **private key-switching** (Algorithm 3): Finally, to reconstruct the final TRGSW ciphertext of μ, we simply need to craft a TRLWE ciphertext which has the same phase as $\mu \cdot \boldsymbol{h_i}$, for each row of the gadget matrix H. Since $\boldsymbol{h_i}$ contains only a single non-zero constant polynomial in position $u \in [1, k+1]$ whose value is $\frac{1}{B_g^w}$ where $w \in [1, \ell]$, the phase of $\mu \cdot \boldsymbol{h_i}$ is $\mu K_u \cdot \frac{1}{B_g^w}$ where K_u is the u-th term of the key K. If we call f_u the (secret) morphism from $\mathbb{T}$ to $\mathbb{T}_N[X]$ defined by $f_u(x) = K_u \cdot x$, we just need to apply f_u homomorphically to the TLWE sample $\bar{\mathbf{c}}^{(w)}$ to get the desired TRLWE sample. Since f_u is 1-lipschitzian (for the infinity norm), this operation be done with additive noise overhead via the private functional keyswitch (Algorithm 3).

Algorithm 11 Circuit Bootstrapping (calling algorithms 9 and 3)

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

Output: A level 1 TRGSW sample $C \in \mathsf{TRGSW}_{K,\eta}(\mu)$

1: **for** $w = 1$ **to** ℓ
2: $\quad \bar{\mathbf{c}}^{(w)} \leftarrow \mathsf{Bootstrapping}_{\mathrm{BK},\frac{1}{B_g^w}}(\underline{\mathbf{c}})$
3: $\quad$ **for** $u = 1$ **to** $k+1$
4: $\quad\quad \boldsymbol{c}^{(u,w)} = \mathsf{PrivKS}(\mathsf{KS}^{(f_u)}, \bar{\mathbf{c}}^{(w)})$
5: Return $C = (\boldsymbol{c}^{(u,w)})_{1\le u\le k+1, 1\le w\le \ell}$

Theorem 6.4 (Circuit Bootstrapping Theorem). *Let $n, \alpha, N, k, B_g, \ell, H, \epsilon$ denote* TRLWE/TRGSW *level 1 parameters, and the same variables names with underbars/upperbars for level 0 and 2 parameters. Let $\underline{\mathfrak{K}} \in \mathbb{B}^{\underline{n}}$, $\mathfrak{K} \in \mathbb{B}^n$ and $\bar{\mathfrak{K}} \in \mathbb{B}^{\bar{n}}$, be a level 0, 1 and 2* TLWE *secret keys, and $\underline{K}, K, \bar{K}$ their respective* TRLWE *interpretation. Let $BK_{\underline{\mathfrak{K}}\to\bar{\mathfrak{K}},\bar{\alpha}}$ be a bootstrapping key, composed by the $\underline{n}$* TRGSW *encryptions $BK_i \in \mathsf{TRGSW}_{\bar{K},\bar{\alpha}}(\underline{\mathfrak{K}}_i)$ for $i \in [\![1,\underline{n}]\!]$. For each $u \in [\![1, k+1]\!]$, let f_u be the morphism from $\mathbb{T}$ to $\mathbb{T}_N[X]$ defined by $f_u(x) = K_u \cdot x$, and $\mathsf{KS}^{f_u}_{\bar{\mathfrak{K}}\to K,\gamma} = (\mathsf{KS}^{(u)}_{i,j} \in \mathsf{TRLWE}_{K,\gamma}((\bar{\mathfrak{K}}_i K_u \cdot 2^{-j})))_{i\in[\![1,\bar{n}]\!], j\in[\![1,t]\!]}$ be the corresponding private-key-switching key. Given a level 0* TLWE *sample $\underline{\mathbf{c}} = (\underline{\mathbf{a}}, \underline{b}) \in \mathsf{TLWE}_{\underline{\mathfrak{K}}}(\mu \cdot \frac{1}{2})$, with $\mu \in \mathbb{B}$, the algorithm 11 outputs a level 1* TRGSW *sample $C \in \mathsf{TRGSW}_K(\mu)$ such that*

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

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

Proof. The output TRGSW ciphertext is correct, because by construction, the i-th TRLWE component $\boldsymbol{c}^{(u,w)}$ has the correct message $\mathsf{msg}(\mu \cdot \boldsymbol{h}_i) = \mu K_u / B_g^w$. $\boldsymbol{c}^{(u,w)}$ is obtained by chaining one TLWE-to-TLWE bootstrapping (Algorithm 9) with one private key-switchings, as in Algorithm 3. The values of maximal amplitude and variance of $\mathsf{Err}(C)$ are directly obtained from the partial results of Theorem 6.2 and Theorem 4.2. In total, Algorithm 11 performs exactly ℓ bootstrappings (Algorithm 9), and $\ell(k+1)$ private key switchings (Algorithm 3). □

Comparison with previous bootstrappings for TGSW. The circuit bootstrapping we just described evaluates a quasilinear number of level-2 external products, and a quasilinear number of level 1 products in the private keyswitchings. With the parameters proposed in the next section, it runs in 0.137 seconds for a 110-bit security parameter, level 2 operations take 70% of the running time, and the private keyswitch the remaining 30%.

Our circuit bootstrapping is not the first bootstrapping algorithm that outputs a TRGSW ciphertext. Many constructions have previously been proposed and achieve valid asymptotical complexities, but very few concrete parameters are proposed. Most of these constructions are recalled in the last section of [26]. In all of them, the bootstrapped ciphertext is obtained as an arithmetic expression on TRGSW ciphertexts involving linear combinations and internal products. First, all the schemes based on scalar variants of TRGSW suffer from a slowdown of a factor at least quadratic in the security parameter, because the products of small matrices with polynomial coefficients (via FFT) are replaced with large dense matrix products. Thus, bootstrapping on TGSW variants would require days of computations, instead of the 0.137 seconds we propose. Now, assuming that all the bootstrapping uses (Ring) instantiations of TRGSW, the design in [11] based on the expansion of the decryption circuit via Barrington theorem, as well as the expression as a minimal deterministic automata of the same function in [26] require a quadratic number of internal level 2 TRGSW products, which is much slower than what we propose. Finally, the CRT variant in [5] and [26] uses only a quasi-linear number of products, but since it uses composition between automata, these products need to run in level 3 instead of level 2, which induces a huge slowdown (a factor 240 in our benchs), because elements cannot be represented on 64-bits native numbers.

Sum-up of elementary Homomorphic Operations Table 1 summarizes the possible operations on plaintexts that we can perform with TFHE, and their correspondence over the ciphertexts. All these operations are expressed on the continuous message space $\mathbb{T}$ for TLWE and $\mathbb{T}_N[X]$ for TRLWE. As previously mentioned, all samples contain noise: if an exact decryption is required, the message space must be chosen accordingly.

Observe that the operations involving keyswitching or bootstrapping can also be used to change the encryption key. If we assume the circular security, we can keep the same key after these operations. Otherwise, the keys are given as a chain, where every secret key is encrypted with the following one, and the each bootstrapping or keyswitching moves from the current key to the next one.

7 Security analysis

On the asymptotical side, TLWE samples can be equivalently rescaled and rounded to their closest binLWE representative, which in turn can be reduced to standard LWE with full secret using the modulus-dimension reduction [9] or group-switching techniques [26]. Therefore, the semantic security of TFHE is asymptotically equivalent to worst case lattice problems. Similarily, the ring and module variants of TRLWE can be mapped to their closest bin-RingLWE instances, whose semantic security is known to be equivalent to worst-case ideal lattice problems [34], and also [3] for an asymptotical equivalence between ring and module LWE problems for large modulus.

Operation	Plaintext	Ciphertext	Variance
Translation	$\mu + w$	$\boldsymbol{c} + (0, w)$	ϑ
Rotation	$X^{u_i}\mu$	$X^{u_i}\boldsymbol{c}$	ϑ
$\mathbb{Z}[X]$-linear	$\sum v_i\mu_i$	$\sum v_i\boldsymbol{c_i}$	$\sum \|v_i\|_2^2\vartheta_i$
SampleExtract	$\sum \mu_i X^i \to \mu_p$	SampleExtract (Sect. 2.2)	ϑ
$\mathbb{Z}$-linear	$f(m_1,\ldots,m_p)$	$\mathsf{PubKS}_{\mathsf{KS}}(f, \mathfrak{c}_1,\ldots,\mathfrak{c}_p)$(Alg.2)	$R^2\vartheta + n\log\left(\frac{1}{\alpha}\right)\mathrm{Cst}_{KS}$
	R-lipschitzian	$\mathsf{PrivKS}_{KS^{(f)}}(\mathfrak{c}_1,\ldots,\mathfrak{c}_p)$(Alg.3)	$R^2\vartheta + np\log\left(\frac{1}{\alpha}\right)\mathrm{Cst}_{KS}$
Ext. product	$b_1 \cdot \mu_2$	$C_1 \boxdot \boldsymbol{c_2}$ (Thm.3.13)	$b_1\vartheta_2 + \mathrm{Cst}_{\mathsf{TRGSW}}\vartheta_1$
CMux	$b_1?\mu_2 : \mu_3$	$\mathtt{CMux}(C_1, \boldsymbol{c_2}, \boldsymbol{c_3})$ (Lem.3.16)	$\max(\vartheta_2,\vartheta_3) + \mathrm{Cst}_{\mathsf{TRGSW}}\vartheta_1$
$\mathbb{T}$-non-linear	$X^{-\varphi(c_1)}\mu_2$	BlindRotate (Alg.4)	$\vartheta + n\mathrm{Cst}_{\mathsf{TRGSW}}$
Bootstrapping	$\mathrm{decrypt}(\mathfrak{c})?m : 0$	Gate Bootstrapping (Alg.9)	Cst
		Circuit Bootstrapping (Alg.11)	

Table 1. TFHE elementary operations - In this table, all μ_i's denote plaintexts in $\mathbb{T}_N[X]$ and $\boldsymbol{c}_i$ the corresponding TRLWE ciphertext. The m_i's are plaintexts in $\mathbb{T}$ and $\mathfrak{c}$ their TLWE ciphertext. The b_i's are bit messages and C_i their TRGSW ciphertext. The ϑ_i's are the noise variances of the respective ciphertexts. In the translation, w is in $\mathbb{T}_N[X]$. In the rotation, the u_i's are integer coefficients. In the $\mathbb{Z}[X]$-linear combination, the v_i's are integer polynomials in $\mathbb{Z}[X]$.

In this section, we will rather focus on the practical hardness of TFHE, and express its effective security parameter λ directly as a function of the entropy of the secret n and the error standard deviation α.

Our analysis is based on the methodology of [4] and [1]. In their work, they review many classes of attacks against LWE, ranging from a direct BDD approach with standard lattice reduction, sieving algorithms, a variant of BKW [7], and resolution of LWE via man in the middle attacks. In general, they found out that there is no single-best attack against all possible parameters, however, according to their results table [4, Section8, Tables 7,8] for the range of dimensions and noise used for FHE, it appears that the SIS-distinguisher attack is often the most efficient attack (related to the Lindner-Peikert [35] model, and also used in the parameter estimation of [24]). Since q is not a parameter in our definition of TLWE, we need to adapt their results. This section relies on the following heuristics concerning the experimental behaviour of lattice reduction algorithms. They have been extensively verified and used in practice.

1. The fastest lattice reduction algorithms in practice are blockwise lattice algorithms (like BKZ-2.0[13], D-BKZ [38], or the slide reduction with large blocksize [27, 38]).
2. Practical blockwise lattice reduction algorithms have an intrinsic quality $\delta > 1$ (which depends on the blocksize), and given a m-dimensional real basis B of volume V, they compute short vectors of norm $\delta^m V^{1/m}$.
3. The running time of BKZ-2.0 (expressed in bit operations) as a function of the quality parameter is: $\log_2(t_{\mathrm{BKZ}})(\delta) = \frac{0.009}{\log_2(\delta)^2} - 27$ (According to the extrapolation by Albrecht et al [2] of Liu-Nguyen datasets [36]).

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

5. For mid-range dimensions and polynomially small noise, the SIS-distinguisher plus lattice reduction algorithms combined with the search-to-decision is the best attack against LWE; (but this point is less clear, according to the analysis of [2], at least, this attack model tends to overestimate the power of the attacker, so it should produce more conservative parameters).

6. Except for small polynomial speedups in the dimension, we don't know better algorithms to find short vectors in random anti-circulant lattices than generic algorithms. This folklore assumption seems still up to date at the time of writing. The most recent asymptotic attacks [22] against ideal lattice do not reach polynomial noise rates, and they are not practical.

The SIS-based distinguisher attack against the LWE problem consists in finding a small integer combination that cancels the left hand side of homogeneous LWE samples. If applying the same combination to the right hand side does not make it small, we deduce that our inputs are not LWE samples, but rather uniformly random samples. Such SIS-distinguisher has in general a small advantage ε. To recover the full key, we use the well known search to decision reduction, which is particularly tight for TLWE: guess that the first key bit is zero, randomize the first coordinates of each sample, and use the distinguisher about $1/\varepsilon^2$ times to amplify its advantage to $\Theta(1)$, and to confirm whether the result are still TLWE samples, *i.e.* if our guess of the first key bit is correct. Once a key bit is found, getting the other bits involves solving lower-dimensional TLWE problems, and are significantly easier, therefore we consider that the complexity of the attack is the time needed to find the first key bit. We also extend the analysis of [4] to handle the continuous torus.

Let $(\boldsymbol{a_1}, b_1), \ldots, (\boldsymbol{a_m}, b_m)$ be either m TLWE samples of error stabdard deviation α or m uniformly random samples of $\mathbb{T}^{n+1}$, we need to find a small combination $v_1, \ldots, v_m$ of samples such that $\sum v_i \boldsymbol{a_i}$ is small. Most previous models, that work on a discrete group, would require that this term is exactly zero. By allowing approximations, we may find valid solutions in smaller dimension m than the usual bound $n \log n$. In particular, even $m < n$ would make sense. Now, consider the $(m+n)$-dimensional lattice, generated by the rows of the following basis $B \in \mathcal{M}_{n+m,n+m}(\mathbb{R})$:

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

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

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

To that end, we apply the fastest algorithm (BKZ-2.0 or slide reduction) directly to $f_q(B)$, which outputs a vector $f_q(\boldsymbol{w})$ of standard deviation $\delta^{n+m}/\sqrt{n+m}$ where $\delta \in]1, 1.1]$ is the quality of the reduction.

Once we obtain a such vector $\boldsymbol{w}$, all we need is to analyse the term $\sum_{i=1}^{m} v_i b_i = \sum_{i=1}^{m} v_i(\boldsymbol{a_i}\boldsymbol{s} + e_i) = \boldsymbol{s} \cdot \sum_{i=1}^{m}(v_i \boldsymbol{a_i}) + \sum_{i=1}^{m} v_i \boldsymbol{e_i} = \boldsymbol{s} \cdot \boldsymbol{x} + \boldsymbol{v} \cdot \boldsymbol{e}$.

It has Gaussian distribution of variance $\sigma^2 = \frac{\delta^{2(m+n)}}{q^2} \cdot \frac{nS^2}{m+n} + \frac{q^{2n/m}\delta^{2(m+n)}\alpha^2 m}{m+n} = \delta^{2(m+n)}\left(\frac{S^2}{q^2} \cdot \frac{n}{m+n} + q^{2n/m}\alpha^2 \frac{m}{m+n}\right)$. Here $S = \frac{\|\boldsymbol{s}\|}{\sqrt{n}} \approx \frac{1}{\sqrt{2}}$. This distribution may be distinguished from the uniform distribution with advantage ε when σ^2 is equal to the smoothing variance $\frac{1}{2\pi}\eta_\varepsilon^2(\mathbb{Z})$. To summarize, the security parameter of LWE is (bounded by) the solution of the following system of equations

$$\lambda(n, \alpha) = \log_2(t_{\text{attack}}) = \min_{0<\varepsilon<1} \log_2\left(\frac{1}{\varepsilon^2} t_{\text{BKZ}}(n, \alpha, \varepsilon)\right) \tag{3}$$

$$\log_2(t_{\text{BKZ}})(n, \alpha, \varepsilon) = \frac{0.009}{\log_2(\delta)^2} - 27 \tag{4}$$

$$\ln(\delta)(n, \alpha, \varepsilon) = \max_{\substack{m>1 \\ q>1}} \frac{1}{2(m+n)}\left(\ln\left(\frac{1}{2\pi}\eta_\varepsilon^2(\mathbb{Z})\right) - \ln\left(\frac{S^2}{q^2}\frac{n}{m+n} + q^{\frac{2n}{m}}\alpha^2\frac{m}{m+n}\right)\right) \tag{5}$$

$$\frac{1}{2\pi}\eta_\varepsilon^2(\mathbb{Z}) \approx \frac{1}{2\pi^2}\ln\left(\frac{2}{\varepsilon}\right). \tag{6}$$

Here, Eq. (3) means that we need to run the distinguisher $\frac{1}{\varepsilon^2}$ times (by Chernoff's bound), and we need to optimize the advantage ε accordingly. [13] Eq.(4) is the heuristic prediction of the running time of lattice reduction. In Eq.(5) q and m need to be chosen in order to maximize the targeted approximation factor of the lattice reduction step.

Differentiating Equation (5) in q, we find that its maximal value is

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

Replacing this value and setting $t = \frac{n}{m+n}$, Equation (5) becomes:

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

Finally, by differentiating this new expression in t, the maximum of δ is reached for $t_{\text{best}} = \frac{\ell_1}{2(\ell_1-\ell_2)}$, because $\ell_1 > \ell_2$, which gives the best choices of m and q and δ. Finally, we optimize ε numerically in Eq.(3).

All previous results are summarized in Figure 9, which displays the security parameter λ as a function of $n, \log_2(\alpha)$.

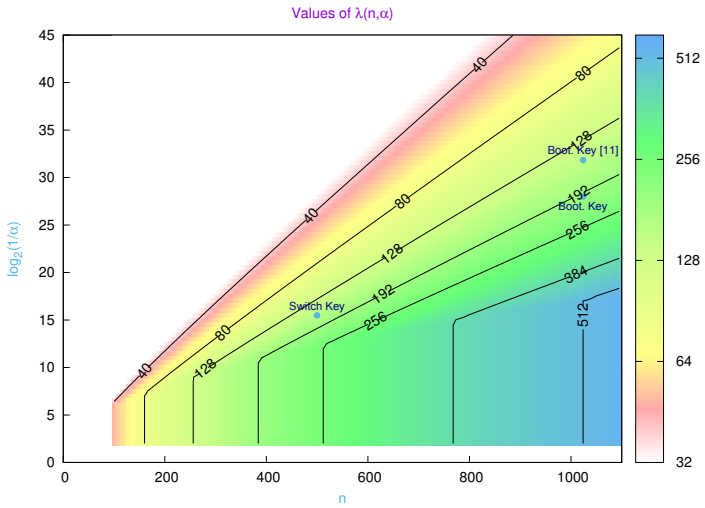
8 Applications, practical parameters and running time estimates

Concrete Parameters

Gate bootstrapping Parameters. From a theoretical point of view, our scale invariant scheme is defined over the real torus $\mathbb{T}$, where all the operations are modulo 1. In practice, since we can work with approximations, we chose to rescale the elements over $\mathbb{T}$ by a factor 2^{32}, and to map them to 32-bit integers. Thus, we take advantage of the native and automatic mod 2^{32} operations, including for the external multiplication with integers. Except for some FFT operations, this seems more stable and efficient than working with floating point numbers and reducing modulo 1 regularly. Polynomials mod $X^N + 1$ are either represented as the classical list of the N coefficients, or using the Lagrange half-complex representation, which consists in the complex ($2 \cdot 64$bits) evaluations of the polynomial over the roots of unity $\exp(i(2j+1)\pi/N)$ for $j \in [\![0, \frac{N}{2}[\![$. Indeed, the

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

Fig. 9. Security parameter λ as a function of n and α for LWE samples



This curve shows the security parameter levels λ (black levels) as a function of $n = kN$ (along the x-axis) and $\log_2(1/\alpha)$ (along the y-axis) for TLWE *(also holds for bin-LWE), considering both the attack of this section and the collision attack in time $2^{n/2}$.*

$\frac{N}{2}$ other evaluations are the conjugates of the first ones, and do not need to be stored. The conversion between both representations is done using a dedicated assembly implementation of the Fast Fourier Transform (FFT) for the anticyclic ring (twice as using the generic library *FFTW* [25] used in by [24]). Note that the direct FFT transform is $\sqrt{2N}$ lipschitzian, so the lagrange half-complex representation tolerates approximations, and 53bits of precision is indeed more than enough, provided that the real representative remains small. However, the modulo 1 reduction, as well as the gadget decomposition in base $\boldsymbol{h}$, are not compatible with the Lagrange representation: we therefore need to regularly transform the polynomials to and from their classical representation.

In the TFHE library [20], the gate bootstrapping uses the following parameters.

- TLWE samples use $\underline{n} = 500$, and standard deviation $\underline{\alpha} = 2^{-7}$, so their amplitude is $< \frac{1}{16}$. A TLWE sample has $32 \cdot (n+1)$ bits ≈ 2 KBytes.
- TRLWE samples use $N = 1024, k = 1$. This corresponds to $(k+1) \cdot N \cdot 32$ bits ≈ 8 KBytes.
- TRGSW samples use $\ell = 2$,$B_g = 1024$. This defines the gadget H and its decomposition $\mathrm{Dec}_{\boldsymbol{h},\beta,\epsilon}$ where $\beta = 512$ and $\epsilon = 2^{-21}$. A TRGSW sample has $(k+1) \cdot \ell$ TLWE samples ≈ 32 KBytes.
- The Bootstrapping Key has n TGSW samples ≈ 15.6 MBytes. Its noise standard deviation is $\alpha = 3.73 \cdot 10^{-9}$. Since we have a lower noise overhead,

this is higher than the standard deviation $\approx 2.59 \cdot 10^{-10}$ of [24], (i.e. ours is more secure), but in counterpart, our TLWE key is binary. Section 7 predicts 198 bits of security for this key.
- We chose $t = 16$ bits for the key switching: the decomposition has precision 2^{-17}, and the KS has $k \cdot N \cdot t$ LWE samples ≈ 32 MBytes. We set the noise stdev to $\gamma = 2.16 \cdot 10^{-5}$, which is estimated to $\lambda = 159$-bits of security by Section 7.

- Correctness: The final error variance after bootstrapping is $1.63 \cdot 10^{-5}$, by Theorem 6.3. It corresponds to a standard deviation of $\sigma = 4.05 \cdot 10^{-3}$. In [24], the final standard deviation is larger $1.08 \cdot 10^{-2}$. In other words, the noise amplitude after our bootstrapping is $< \frac{1}{16}$ with very high probability $\mathsf{erf}(1/16\sqrt{2}\sigma) \geq 1 - 2^{-150}$ (better than $\geq 1 - 2^{-32}$ in [24]).

Note that the size of the key switching key can be reduced by a factor $n + 1 = 501$ if all the masks are the output of a pseudo random function; we may for instance just give the seed. The same technique can be applied to the bootstrapping key, on which the size is only reduced by a factor $k + 1 = 2$.

The source code of our implementation is available on github `https://github.com/tfhe/tfhe`. We implemented the FHE scheme in C/C++, and run the bootstrapping algorithm on a 64-bit single core (i7-4910MQ) at 2.90GHz. This seems to correspond to the machine used in [24]. We measured a running time of 13ms per bootstrapping, and of $34\mu s$ per external product, both using the Lagrange half-complex representation. Profiling the execution shows that the FFTs and complex multiplications are taking 66% of the total time. The remaining time is mostly taken by the keyswitch operation, and the decomposition in base $\boldsymbol{h}$. As a consequence, all binary gates are executed in $13ms$, and the native bootstrapped MUX (also described in Section 6.1) gate takes $26ms$ single core time.

Fig. 10. Parameters and security of the Gate bootstrapping

	n	α	$\boldsymbol{\lambda}$	$\varepsilon_{\text{best}}$	m_{best}	q_{best}	δ_{best}
Switch key	500	$2.43 \cdot 10^{-5}$	159	2^{-14}	497	178.7	1.0052
Boot. key	1024	$3.73 \cdot 10^{-9}$	198	2^{-10}	1037	14696.	1.0046
Boot.key, [24]	1024	$2.59 \cdot 10^{-10}$	149	2^{-7}	1051	60351	1.0052

This table precises the parameters for the keyswitching key and the bootstrapping key for our implementation and for the one in [24].

Table 10 shows that the strength of the lattice reduction is compatible with the values announced in [24]. Our model predicts that the lattice reduction phase is harder ($\delta = 1.0052$ in our analysis and $\delta = 1.0064$ in [24]), but the value of ε is bigger in our case. Overall, the security of their parameters-set is evaluated by our model to 149-bits of security, which is larger than the

$\geq$ 100-bits of security announced in [24]. The main reason is that we take into account the number of times we need to run the SIS-distinguisher to obtain a non negligible advantage. Since our scheme has a smaller noise propagation overhead, we were able to raise the input noise levels in order to strengthen the system, so with the parameters we chose in our implementation, our model predicts 198-bits of security for the bootstrapping key and 163-bits for the keyswitching key (which becomes the bottleneck).

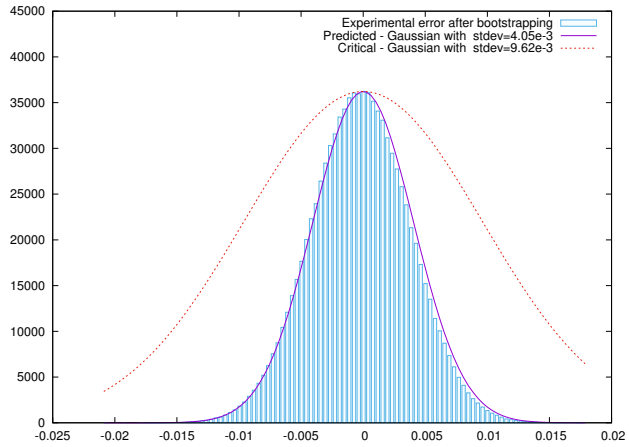


Fig. 11. Accuracy of the average case theorems, and experimental validation of the independence assumption - We generated several random Boolean circuits of depth larger than 10000, composed by a million of bootstrapped homomorphic gates. We executed them and measured the actual noise of the ciphertexts. The (light blue) histogram is the measured error distribution after every bootstrapped gate, the (purple) plain line represents the Gaussian distribution whose variance is predicted by Theorem 6.3 using the parameters of Section 8, and the (red) dashed line represents the critical Gaussian distribution for an amplitude of $\frac{1}{16}$. The experimental results confirms that our average case theorem predicts the output variance accurately, and that the noise distribution after bootstrapping is Gaussian. This experimentally validates our independence heuristic assumption 3.11, even when ciphertexts are re-used in a very large depth homomorphic circuit.

Gate bootstrapping mode	
Gate bootstrapping (1 bit)	$t_{GB} = 13ms$
Time per any binary gate (HomAND, HomOR, HomXOR, ...)	$t_{GB} = 13ms$
Time per HomMUX	$2t_{GB} = 26ms$

Circuit Bootstrapping Starting from all these considerations, we implemented our circuit bootstrapping as a proof of concept. The code is available in the

experimental repository of TFHE [20]. We perform a Circuit Bootstrapping in 0.137 seconds. One of the main constraints to obtain this performance is to ensure that all the computations are feasible and correct under 53 bits of floating point precision, in order to use the fast FFT. This requires to refine the parameters of the scheme. We verified the accuracy of the FFT with a slower but exact Karatsuba implementation of the polynomial product.

In our three levels, we used the following TRLWE and TRGSW parameter sets, which have at least 152-bits of security, according to the security analysis from Section 7.

Level	**Minimal noise stdev** α	n	B_g	ℓ	λ
0	$\underline{\alpha} = 6.10 \cdot 10^{-5}$	$\underline{n} = 500$	N.A.	N.A.	194
1	$\alpha = 3.29 \cdot 10^{-10}$	$n = 1024$	$B_g = 2^8$	$\ell = 2$	152
2	$\bar{\alpha} = 1.42 \cdot 10^{-14}$	$\bar{n} = 2048$	$\bar{B}_g = 2^9$	$\bar{\ell} = 4$	289

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

Level	t	**KS noise stdev** γ	**Usage**
$1 \to 0$	$\underline{t} = 12$	$\underline{\gamma} = 6.10 \cdot 10^{-5}$	Circuit Bootstap, Pre-KS
$2 \to 1$	$\bar{t} = 30$	$\bar{\gamma} = 3.29 \cdot 10^{-10}$	Circuit Bootstap, Step 4 in Alg. 11
$1 \to 1$	$t = 24$	$\gamma = 2.38 \cdot 10^{-8}$	TBSR

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

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

And finally, this table summarizes the timings (Core i7-4910MQ laptop), noises overhead, and maximal depth of all our primitives.

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

Time Comparison With these parameters, we analyse the (single-core) execution timings for the evaluation of the LUT, MAX and Multiplication in LHE and FHE mode.

In the LHE mode (left hand side of Fig. 12), all inputs are fresh ciphertexts (either TRLWE or TRGSW) and we compare the previous versions [17] (without packing/batching or gate bootstapping) with the new optimizations i.e. horizontal/vertical packing; with weighted automata or with TBSR techniques. In the FHE mode (right hand side of Fig. 12), all inputs and outputs are TLWE samples on the $\{0, \frac{1}{2}\}$ message space with noise amplitude $\frac{1}{4}$. Each operation starts by

bootstrapping its inputs. We compare the gate-by-gate bootstapping strategy with the mixed version where we use leveled encryption with circuit bootsrapping. Our goal is to identify which method is better for each of the 6 cases. We observe that compared to the gate bootstrapping, we obtain a huge speed-up for the homomorphic evaluation of arbitrary function in both LHE and FHE mode, in particular, we can evaluate a 8 bits to 1 bit lookup table and bootstrap the output in just 137ms, or evaluate an arbitrary 8 bits to 8 bits function in $1.096s$, and an arbitrary 16 bits to 8 bits function in $2.192s$ in FHE mode. For the multiplication in LHE mode, it is better to use the weighted automata technique when the number is less than 128 bits, and the TBSR counter after that. In the FHE mode, the weighted automata becomes faster than gate-bootstrapping after 4 bits of inputs, then the TBSR optimization becomes faster for > 64 bits inputs.

9 Conclusion

In this paper we presented a complete overview of the TFHE construction, from an abstract LWE and GSW point of view, to a concrete and efficient instantiation. We presented many improvements to the leveled and fully homomorphic modes: several ways to work over packed ciphertexts, new computation models that use TBSR counters and deterministic weighted automata techniques, as well as two efficient bootstrappings that operate in gate or circuit mode. We also provided a dedicated security analysis explaining how to chose the parameters depending on the desired security level, and we presented implementation details with practical timings for several applications using the techniques previously described.

References

1. M. R. Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in helib and SEAL. In *EUROCRYPT 2017*, pages 103–129, 2017.
2. M. R. Albrecht, C. Cid, J. Faugère, R. Fitzpatrick, and L. Perret. On the complexity of the BKW algorithm on LWE. *Designs, Codes and Cryptography*, 74/2:325–354, 2015.
3. M. R. Albrecht and A. Deo. Large modulus ring-lwe >= module-lwe. In *ASIACRYPT 2017*, 2017.
4. M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *J. Mathematical Cryptology*, 9(3):169–203, 2015.
5. J. Alperin-Sheriff and C. Peikert. Faster bootstrapping with polynomial error. In *Crypto*, pages 297–314, 2014.
6. D. Benarroch, Z. Brakerski, and T. Lepoint. Fhe over the integers: Decomposed and batched in the post-quantum regime. Cryptology ePrint Archive, 2017/065.
7. A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. of ACM*, 50(4):506–519, 2003.
8. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.
9. Z. Brakerski, A. Langlois, C. Peikert, O. Regev, and D.Stehlé. Classical hardness of learning with errors. In *Proc. of 45th STOC*, pages 575–584. ACM, 2013.

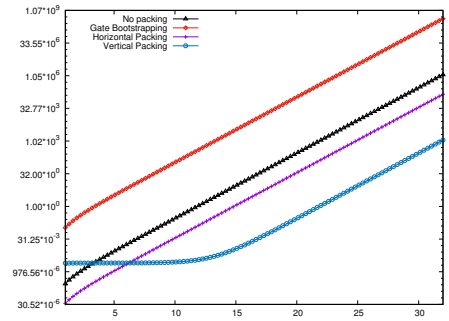
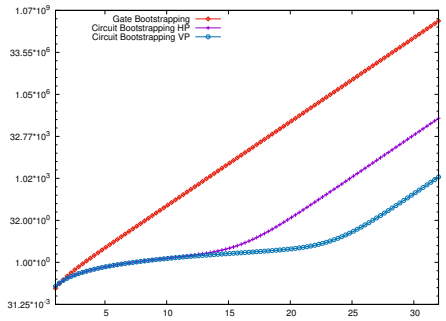
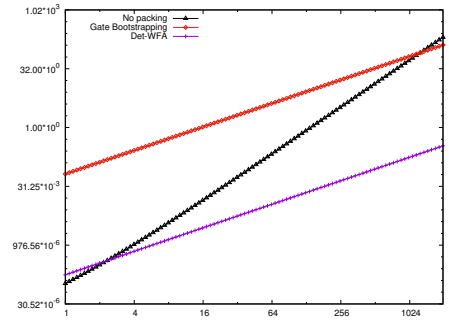
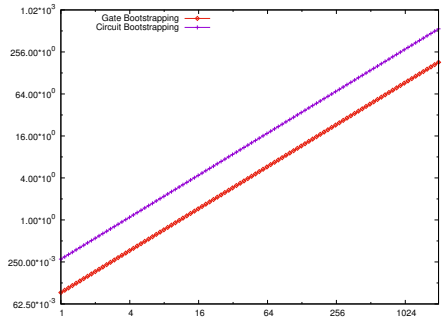
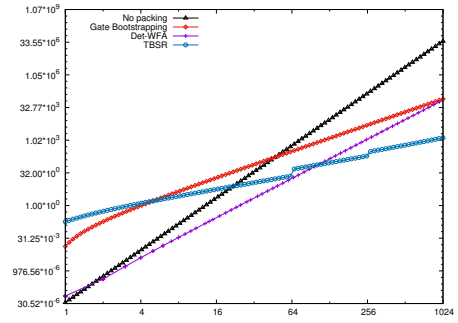
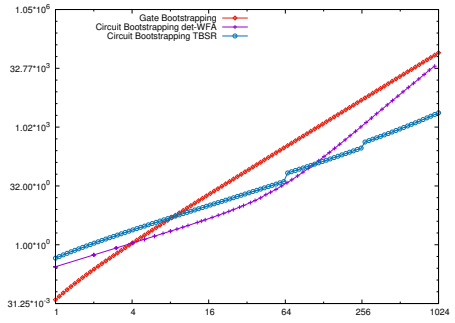
Leveled homomorphic encryption	Fully homomorphic encryption
	
(a.) Leveled LUT (d-bits input and $s = 8$-bits output) - No packing: $s(2^d - 1)t_{XP}$ - Gate bootst.: $s(2^d - 1)t_{GB}$ - Horizontal Packing (HP): $(2^d - 1)t_{XP}$ - Vertical Packing (VP): $s(2^d/N - 1 + \log N)t_{XP}$.	**(b.)** LUT (d-bits input and $s = 8$-bits output) - Gate bootst.: $(d + s(2^d - 1))t_{GB}$ - Circuit bootst. with HP: $dt_{CB} + (2^d - 1)t_{XP}$ - Circuit bootst. with VP: $dt_{CB} + s(2^d/N - 1 + \log N)t_{XP}$.
	
(c.) Leveled Max (2 inputs of d-bits each) - No packing: $3(d(d + 1)/2)t_{XP}$ - Gate bootst.: $5dt_{GB}$ - Det Weighted automata (Det WFA): $5dt_{XP}$	**(d.)** Max (2 inputs of d-bits each) - Gate bootst.: $7dt_{GB}$ - Circuit bootst. with VP: $2dt_{CB} + 5dt_{XP}$
	
(e.) Leveled Multiplication (2 inputs of d-bits each) - No packing: $(d^4 + o(d^4))t_{XP}$ - Gate bootst.: $(6d^2 - 5d)t_{GB}$ - Det WFA: $\Theta(d^3)t_{XP}$ (computed by optimization program) - TBSR: $2d^2t_{XP} + (2d - 2)t_{KS}$	**(f.)** Multiplication (2 inputs of d-bits each) - Gate bootst.: $(6d^2 - 3d)t_{GB}$ - Circuit bootst with Det WFA.: $2dt_{CB} + \Theta(d^3)t_{XP}$ (computed by optimization program) - Circuit bootst with TBSR: (computed by optimization program)

Fig. 12. The y coordinate represents the running time in seconds (in logscale) , the x coordinate represents the number of bits in the input (in logscale for c,d,e,f).

10. Z. Brakerski and R. Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In *Crypto'2016*, volume 9814, pages 190–213, 2016.
11. Z. Brakerski and V. Vaikuntanathan. Lattice-based FHE as secure as PKE. In *ITCS*, pages 1–12, 2014.
12. A. L. Buchsbaum, R. Giancarlo, and J. R. Westbrook. On the determinization of weighted finite automata. *SIAM Journal on Computing*, 30(5):1502–1531, 2000.
13. Y. Chen and P. Q. Nguyen. BKZ 2.0: Better lattice security estimates. In *Proc. of Asiacrypt*, pages 1–20, 2011.
14. J. H. Cheon, J. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, and A. Yun. Batch fully homomorphic encryption over the integers. In *EUROCRYPT 2013*, 2013.
15. J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *Asiacrypt 2017*, 2016. `http://eprint.iacr.org/2016/421`.
16. J. H. Cheon and D. Stehlé. Fully homomophic encryption over the integers revisited. In *EUROCRYPT 2015*, pages 513–536. Springer, 2015.
17. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, pages 3–33. Springer, 2016.
18. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. A homomorphic lwe based e-voting scheme. In *PQ Cryptography*, pages 245–265. Springer, 2016.
19. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In *Advances in Cryptology - ASIACRYPT 2017*. Springer, 2017.
20. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. TFHE: Fast fully homomorphic encryption library. https://tfhe.github.io/tfhe/, August 2016.
21. J. Coron, T. Lepoint, and M. Tibouchi. Scale-invariant fully homomorphic encryption over the integers. In *PKC 2014*, pages 311–328, 2014.
22. R. Cramer, L. Ducas, and B. Wesolowski. Short stickelberger class relations and application to ideal-svp. In *Eurocrypt 2017*, 2016.
23. M. Droste and P. Gastin. Weighted automata and weighted logics. In *Handbook of weighted automata*, pages 175–211. Springer, 2009.
24. L. Ducas and D. Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In *Eurocrypt*, pages 617–640, 2015.
25. M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".
26. N. Gama, M. Izabachène, P. Q. Nguyen, and X. Xie. Structural lattice reduction: Generalized worst-case to average-case reductions. *ePrint Archive, 2014/283*, 2014.
27. N. Gama and P. Q. Nguyen. Predicting Lattice Reduction. In *Eurocrypt*, 2008.
28. C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.
29. C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Crypto'13*, 2013.
30. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Attribute-based encryption for circuits. *Journal of the ACM (JACM)*, 62(6):45, 2015.
31. S. Halevi and I. V. Shoup. Helib - an implementation of homomorphic encryption. https://github.com/shaih/HElib/, September 2014.

32. S. Halevi and V. Shoup. Algorithms in helib. In *Crypto'2014*, pages 554–571, 2014.
33. N. Howgrave-Graham. Approximate integer common divisors. In *CaLC*, volume 1, pages 51–66. Springer, 2001.
34. A. Langlois and D. Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015.
35. R. Lindner and C. Peikert. Better key sizes (and attacks) for lwe-based encryption. In *Proc. of CT-RSA*, volume 6558 of *LNCS*. Springer-Verlag, 2011.
36. M. Liu and P. Q. Nguyen. Solving bdd by enumeration: An update. In *Proc. of CT-RSA*, volume 7779 of *LNCS*, pages 293–309. Springer-Verlag, 2013.
37. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, pages 1–23, 2010.
38. D. Micciancio and M. Walter. Practical, predictable lattice basis reduction. In *Proc. of Eurocrypt 2016*, volume 9665 of *LNCS*, pages 820–849. Springer-Verlag, 2016.
39. M. A. R. Hiromasa and T. Okamoto. Packing messages and optimizing bootstrapping in gsw-fhe. In *PKC '15*, pages 699–715, 2015.
40. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.
41. M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *Eurocrypt*, pages 24–43, 2010.