Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE

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,
[4] CEA LIST, Point Courrier 172, 91191 Gif-sur-Yvette Cedex, France

Abstract. In this paper, we present several methods to improve the evaluation of homomorphic functions in TFHE, both for fully and for leveled homomorphic encryption. We propose two methods to manipulate packed data, in order to decrease the ciphertext expansion and optimize the evaluation of look-up tables and arbitrary functions in RingGSW based homomorphic schemes. We also extend the automata logic, introduced in [19, 12], 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 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-by-gate bootstrapping given in [12]. Finally, we propose concrete parameter sets and timing comparison for all our constructions.

Keywords: FHE, leveled, bootstrapping, LWE, GSW, packing, weighted automata, arithmetic

1 Introduction

Fully homomorphic encryption (FHE) allows arbitrary computations over encrypted data, without decrypting them. The first construction was proposed in 2009 by Gentry [20], which introduced a new technique called bootstrapping to handle the noise propagation in ciphertexts. Although many efforts have been done since this first proposal to improve FHE, it remains too slow for real world applications. The most promising constructions are [5, 21, 30]. We focus on constructions based on the LWE problem, introduced by Regev in 2005 [28], and its

Acknowledgements: This work has been supported in part by the CRYPTOCOMP project.

ring variants [25]. Some public implementations are available, namely Helib [22, 23], FV-NFlib [24] and SEAL [29], based on BGV [5, 18], and FHEW [17] and TFHE [14], based on GSW [21, 17, 12].

BGV-based schemes use in general slow operations, but they can treat a lot of bits at the same time, so they can pack and batch many operations in a SIMD manner, like in GPUs. Further, the set of operations that are efficient with BGV are very constrained by the parameter set. Some parameters allow very fast vectorial sums and products modulo a fixed modulus (as in AES). But with these parameters, a comparison, a classical addition, extracting one bit or more complicated bit operations (as in SHA-256) are very slow.

On the other hand, recent developments have shown that GSW operations can evaluate very fast independent elementary operations on bits, like in a CPU. In the TFHE scheme (presented in [12] and based on GSW [21] and its ring variant [17]) the elementary operations are all the binary gates. Therefore, it is easy to represent any function that has few gates, and the running time is simply proportional to their number. Some papers describing GSW-based schemes have already tried to perform multibit or packed/batched operations. In [4] it is proposed an extension of FHEW [17] on which the bootstrapping is used to evaluate non-linear functions with a few input bits. Unfortunately, the parameter sizes must increase exponentially with the number of bits in the plaintext space. But until this work, it was not clear how to perform efficient evaluations on packed data or batch operations, as it is in BGV-based schemes.

Homomorphic encryption falls in two families: leveved (LHE) and fully (FHE) homomorphic encryption. Informally, in LHE, for each function, there exist parameters that can homomorphically evaluate it . The structure of the function to be evaluated (multiplicative depth in BGV or depth of compositions of branching algorithms for GSW) translates into a noise overhead, and the parameters must be chosen large enough to support this noise bound. This concept is represented in the paper by the notion of parameter levels. In FHE, a single parameter set allows to evaluate any function. This generalized definition implies that FHE is a particular case of LHE.

In many FHE schemes, the elementary operations consist in leveled gates with a symmetric noise propagation formula, and where non-linear gates cost more than linear ones. The papers [3, 26] improve the efficiency of fully homomorphic implementations by optimizing the placement of bootstrapping between the gates throughout the circuit. This strategy does not really apply to GSW schemes that strongly rely on the asymmetric noise propagation formula, in which all circuits are expressed as deterministic automata or branching algorithms, because the depth of the circuit has a very small impact on the noise.

The TFHE construction of [12] proposes two modes of operations: a FHE mode composed of bootstrapped binary gates, and a LHE mode which can evaluate a deterministic automata or branching algorithms and which supports very large

To simplify, we include the key size and the noise rate.

depth transitions . Note however that in the LHE mode of [12], the inputs and the output are of different types, which makes it non-composable. In this paper we optimize both FHE/LHE modes, and we solve the non-composability constraint.

Our Contribution. In this paper, we improve the TFHE construction of [12] for both FHE and LHE modes.

We first propose a *blind rotation* algorithm that we describe in Section 2. In FHE mode, this algorithm contributes to the acceleration of the gate bootstrapping of [12], and its implementation is now included in the core of the TFHE library [14]. This algorithm is also one of the building block we use to improve the LHE mode of TFHE.

Because of the asymmetric noise propagation, operating over packed ciphertexts in GSW-based schemes is harder than in BGV-based schemes. We describe two different techniques, that we call *horizontal packing* and *vertical packing*, that can be used to improve the evaluation of leveled circuits. An arbitrary function from $\{0,1\}^n \to \mathbb{T}^p$ can be represented as a truth table with p columns and 2^s rows. By packing these coefficients horizontally, the homomorphic evaluation of the function can be batched, and the p outputs can be produced in parallel. This technique is classical, but is only efficient if p is very large. We propose another technique, called *vertical packing*, which packs the coefficients column-wise, and which achieve its maximal speed-up also when p is equal to 1.

We also extend the deterministic finite automata framework proposed in [19, 12] by working with *deterministic weighted finite automata*. For most multibit arithmetic functions, such as addition, multiplication and maximum value, these latter allow to compute the whole output in a running time that would have previously produced only a single bit.

In fact, thanks to the presence of weights (which could be seen as a memory that stores partial results), the final result is given in a single pass. When an arithmetic operation is evaluated by a deterministic automata, only the destination state matters. The only bit of information that is retained is whether or not the the destination state is accepting, and the rest of the path is forgotten. Thus, we need one automata evaluation for each bit of the result. Instead, by assigning a vector of weights on each transition, we are able to retain enough information along the path to get all the bits of the result at once in a single pass of the automata. Hence, the complexity of these operations is decreased by at least one order of magnitude. Furthermore, we propose a new homomorphic counter (called *TBSR*) that can support all the homomorphic basic operations related to the multiplication (such as incrementation, division by 2 and extraction of bits). This technique gives another speed-up by a factor equal to the bit-size of the input. We show how to use it to represent the

The TFHE construction is implemented and publicly available [14]. The actual running timings are $13ms$ for each bootstrapped binary gate in FHE mode, and $34\mu s$ per transition in LHE mode. The implementation also includes optimizations described in Section 2.

$O(d^2)$ (with d equal to the size of the input) schoolbook multi-addition or multiplication circuits, without increasing the homomorphic depth and with very low noise overhead.
Our last contribution solves the main problem of the leveled mode of TFHE, which is the non-composability, due to the fact that inputs and outputs are of different types. The inputs are in fact RingGSW ciphertexts, while the outputs are LWE ciphertexts. We introduce a new bootstrapping, called *circuit bootstrapping*, that allows to transform LWE ciphertexts back to RingGSW ciphertexts that can be reused as inputs in leveled circuits. The implementation of this circuit bootstrapping is publicly available [14] and it runs in $137ms$, improving all previous techniques. The introduction of the circuit bootstrapping closes the loop and allows all the new techniques previously described to be applied also in a FHE mode. To show how these techniques improve homomorphic evaluations, we propose several examples with concrete parameters and running time. For instance, we show that we can evaluate a 10 bits to 1 bit ($\{0,1\}^{10} \to \{0,1\}$) look-up table in $340\mu s$ and we can bootstrap the output in just $137ms$.

Paper organization. We first review mathematical definitions for the continuous LWE and RingGSW encryption over the torus and review the algorithmic procedures for the homomomorphic evaluation of gates. In particular, we extend the keyswitching algorithm to evaluate public or private $\mathbb{Z}$-module morphisms, and explain how it can be used to pack, unpack and move data across slots of a ciphertext. In Section 3, we show various techniques to speed-up operations on packed data: horizontal and vertical packing, our method to evaluate arithmetic functions via weighted automata and our TBSR counter technique. In Section 4, we introduce our *circuit bootstrapping* algorithm which makes it possible to connect gates of either RingGSW or LWE types and give the practical execution timings we have obtained. Section 5 depicts all our complexity results for different parameters set.

2 Preliminaries

This section introduces and revisits some basic concepts to understand the rest of the paper. The homomorphic constructions we present are based on the LWE problem, presented by Regev in 2005 [28], and on the GSW construction, proposed by Gentry-Sahai-Waters in 2013 [21]. We use the generalized definitions of TLWE and TGSW (the T stands for the torus representation) proposed in [12], and extend some of their results.

2.1 Background on TFHE

We denote by λ the security parameter. The set $\{0,1\}$ is written as $\mathbb{B}$. The real torus $\mathbb{R}/\mathbb{Z} = \mathbb{R} \mod 1$ of real numbers mod 1 is denoted by $\mathbb{T}$. $\mathfrak{R}$ is the ring $\mathbb{Z}[X]/(X^N+1)$ of integer polynomials modulo X^N+1, and $\mathbb{T}_N[X]$ is the module $\mathbb{R}[X]/(X^N+1) \mod 1$ of torus polynomials, where N is a power of 2. $\mathbb{B}_N[X]$

denotes the subset of $\mathfrak{R}$ of polynomials with binary coefficients. Note that $\mathbb{T}$ is a $\mathbb{Z}$-module and that $\mathbb{T}_N[X]$ is a $\mathfrak{R}$-module. The set of vectors of size n in E is denoted by E^n, and the set of $n \times m$ matrices with entries in E is noted $\mathcal{M}_{n,m}(E)$. As before, $\mathbb{T}^n$ (resp. $\mathbb{T}_N[X]^n$) and $\mathcal{M}_{n,m}(\mathbb{T})$ (resp. $\mathcal{M}_{n,m}(\mathbb{T}_N[X])$) are $\mathbb{Z}$-modules (resp. $\mathfrak{R}$-modules).

Distance, Lipschitzian functions, Norms We use the standard ℓ_p-distance over $\mathbb{T}$, and use the (more convenient but improper) $\|x\|_p$ notation to denote the distance between 0 and x. Note that it satisfies $\forall m \in \mathbb{Z}, \|m \cdot \boldsymbol{x}\|_p \leq |m| \, \|\boldsymbol{x}\|_p$. For an integer or torus polynomial a modulo $X^N + 1$, we write $\|a\|_p$ the norm of its unique representative coefficients of degree $\leq N - 1$. the notion of lipschitzian function always refers to the ℓ_∞ distance: a function f is R-lipschitzian iff. $\|f(x) - f(y)\|_\infty \leq R \|x - y\|_\infty$ for all inputs x, y.

TLWE TLWE is a generalized and scale invariant version of the LWE problem, proposed by Regev in 2005 [28], over the Torus $\mathbb{T}$.

Given a small linear lipshtitzian function φ_s from $\mathbb{T}_N[X]^{k+1}$ to $\mathbb{T}_N[X]$ (that depends on the secret key) and which we'll call the *phase* function, the TLWE encryption of $\mu \in \mathbb{T}_N[X]$ simply consists in picking a ciphertext c which is a Gaussian approximation of a preimage $\varphi_s^{-1}(\mu)$. If the Gaussian noise is small enough, the distribution of $\varphi_s(c)$ (over the probability space Ω of all possible choices of Gaussian noise) remains *concentrated* around the message μ, *i.e.* included in a ball of radius $< \frac{1}{4}$ around μ. Because this distribution is concentrated, it allows to properly define the intuitive notions of *expectation* and *variance*, which would in general not exist over the Torus: in this case, the expectation of $\varphi_s(c)$ is the original message μ, and its variance is equal to the variance of the Gaussian noise that was added during encryption. We refer to [12] for a general definition of Ω-space, concentrated distribution, expectation, variance, Gaussian and sub-Gaussian distributions over the Torus.

More precisely, a TLWE secret key $\boldsymbol{s} \in \mathbb{B}_N[X]^k$ is a vector of k binary polynomials of degree N. We assume that each coefficient of the secret key is chosen uniformly, so the key has $n = kN$ bits of entropy.

Definition 2.1 (TLWE, phase). TLWE *ciphertexts or samples are* $\boldsymbol{c} = (\boldsymbol{a}, b) \in \mathbb{T}_N[X]^{k+1}$ *that fall in one of the three cases:*

- *Noiseless Trivial of* μ*:* $\boldsymbol{a} = \boldsymbol{0}$ *and* $b = \mu$*. Note that this sample is independent of the secret key.*
- *Fresh* TLWE *sample of* μ *of standard deviation* α*:* $\boldsymbol{a}$ *is uniformly in* $\mathbb{T}_N[X]^k$ *and* b *follows a continuous Gaussian of standard deviation* α *centered in* $\mu + \boldsymbol{s} \cdot \boldsymbol{a}$*, where the variance is* α^2*. In the following, we will write* $(\boldsymbol{a}, b) \in \mathsf{TLWE}_{\boldsymbol{s},\alpha}(\mu)$*.*
- *Combination of* TLWE *samples:* $\boldsymbol{c} = \sum_{j=1}^{p} r_j \cdot \boldsymbol{c_j}$ *is a* TLWE *sample, where* $\boldsymbol{c_1}, \ldots, \boldsymbol{c_p}$ *are* TLWE *sample under the same key and* $r_1, \ldots, r_p$ *in* $\mathbb{Z}$ *or* $\mathfrak{R}$*.*

The phase of a sample $\boldsymbol{c}$ *is defined as* $\varphi_{\boldsymbol{s}}(\boldsymbol{c}) = b - \boldsymbol{s} \cdot \boldsymbol{a}$*.*

Like in [12], we say that a TLWE sample $\boldsymbol{c}$ is *valid* iff 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. The message of a sample $\boldsymbol{c}$, written $\mathsf{msg}(\boldsymbol{c})$ is defined as the expectation of its phase over the Ω-probability space. We will write $\boldsymbol{c} \in \mathsf{TLWE}_{\boldsymbol{s}}(\mu)$ iff $\mathsf{msg}(\boldsymbol{c}) = \mu$. The error of a TLWE sample $\boldsymbol{c}$, $\mathsf{Err}(\boldsymbol{c})$ is then computed as $\varphi(\boldsymbol{c}) - \mathsf{msg}(\boldsymbol{c})$. The variance of the error will be denoted $\mathsf{Var}(\mathsf{Err}(\boldsymbol{c}))$ and its maximal amplitude $\|\mathsf{Err}(\boldsymbol{c})\|_\infty$.

The message of a fresh sample in $\mathsf{TLWE}_{\boldsymbol{s},\alpha}(\mu)$ is μ and its variance is α^2. The message function is linear: $\mathsf{msg}(\sum_{j=1}^p r_j \cdot \boldsymbol{c_j})$, where $c_j \in \mathsf{TLWE}_{\boldsymbol{s}}(r_j\mu_j)$ is equal to $\sum_j^p r_j\mu_j$ provided that the variance $\mathsf{Var}(\mathsf{Err}(\boldsymbol{c})) \leq \sum_{j=1}^p \|r_j\|_2^2 \cdot \mathsf{Var}(\mathsf{Err}(\boldsymbol{c_j}))$ and the maximal amplitude $\|\mathsf{Err}(\boldsymbol{c})\|_\infty \leq \sum_{j=1}^p \|r_j\|_1 \cdot \|\mathsf{Err}(\boldsymbol{c_j})\|_\infty$ remains small.

This definition of message has the great advantages to be linear, continuous, and that it works with infinite precision even over the continuous torus. In the practical case where the message is known to belong to a discrete subset $\mathcal{M}$ of $\mathbb{T}_N[X]$ and that the noise amplitude of $\boldsymbol{c}$ is smaller than the packing radius of $\mathcal{M}$, then the decryption algorithm can retrieve the message in practice by rounding the phase of the sample to its nearest element in $\mathcal{M}$. For example with $\mathcal{M} = \{(0, 1/2)\}[X]$, the packing is $1/4$ and thus the samples of variance smaller than $(1/2^{10})$ are decryptable with overwhelming probability.

Distinguishing TLWE encryptions of $\mathbf{0}$ from random samples in $\mathbb{T}_N[X]^k \times \mathbb{T}_N[X]$ is equivalent to the LWE problem initially defined by Regev [28] and its ring [25] and Scale invariant [6, 11, 13] variants.

The main parameters of TLWE are the noise rate α and the key entropy n, and the security parameter is a function of those parameters, as specified in [12, Sec.6]. By choosing $N = 1$ and k large, TLWE-problem is the (Scalar) binary-TLWE-problem. When N large and $k = 1$, TLWE is binary-RingLWE.

TGSW In the same line as TLWE, TGSW generalizes the GSW encryption scheme, proposed by Gentry, Sahai and Waters in 2013 [21]. The gadget matrix H is defined with respect to a base $B_g \in \mathbb{N}$ as the $((k+1)\ell) \times (k+1)$ matrix with ℓ repeated super-decreasing $\mathbb{T}$-polynomials $(1/B_g, \ldots, 1/B_g^\ell)$ as:

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

With this choice of gadget, it is possible to efficiently decompose elements of $\mathbb{T}_N[X]^{k+1}$ as a small linear combination of rows of H. As in [12], we use approximate decomposition. For a quality parameter $\beta \in \mathbb{R}_{>0}$ and a precision $\epsilon \in \mathbb{R}_{>0}$, we call $Dec_{H,\beta,\epsilon}(\boldsymbol{v})$ the (possibly randomized) algorithm that outputs a small vector $\boldsymbol{u} \in \mathfrak{R}^{(k+1)\ell}$, such that $\|\boldsymbol{u}\|_\infty \leq \beta$ and $\|\boldsymbol{u} \cdot H - \boldsymbol{v}\|_\infty \leq \epsilon$. In this

paper we will always use this gadget H with the decomposition in base B_g, so we have $\beta = B_g/2$ and $\epsilon = 1/2B_g^\ell$.

TGSW *samples.* Let $\boldsymbol{s} \in \mathbb{B}_N[X]^k$ be a TLWE secret key and $H \in \mathcal{M}_{(k+1)\ell,k+1}(\mathbb{T}_N[X])$ the gadget previously defined. A TGSW sample C of a message $\mu \in \mathfrak{R}$ is equal to the sum $C = Z + \mu \cdot H \in \mathcal{M}_{(k+1)\ell,k+1}(\mathbb{T}_N[X])$ where $Z \in \mathcal{M}_{(k+1)\ell,k+1}(\mathbb{T}_N[X])$ is a matrix such that each line is a random TLWE sample of 0 under the same key.

A sample $\boldsymbol{C} \in \mathcal{M}_{(k+1)\ell,k+1}(\mathbb{T}_N[X])$ is a valid TGSW sample iff there exists a unique polynomial $\mu \in \mathfrak{R}/H^\perp$ and a unique key $\boldsymbol{s}$ such that each row of $\boldsymbol{C} - \mu \bullet H$ is a valid TLWE sample of 0 w.r.t. the key $\boldsymbol{s}$. We denote $\mathsf{msg}(C)$ the message μ of $\boldsymbol{C}$. By extension, we can define the phase of a TGSW sample C as the list of the $(k+1)\ell$ TLWE phases of each line of C, and the error as the list of the $(k+1)\ell$ TLWE errors of each line of C.

In addition, if we linearly combine TGSW samples $C_1, \ldots, C_p$ of messages $\mu_1, \ldots, \mu_p$ with the same keys and independent errors, s.t. $C = \sum_{i=1} e_i \cdot C_i$ is a sample of message $\sum_{i=1}^p e_i \cdot \mu_i$. The variance $\mathsf{Var}(C) = \sum_{i=1}^p \|e_i\|_2^2 \cdot \mathsf{Var}(C_i)$ and noise infinity norm $\|\mathsf{Err}(C)\|_\infty = \sum_{i=1}^p \|e_i\|_1 \cdot \|\mathsf{Err}(C)\|_\infty$. And, the lipschitzian property of the phase is preserved, i.e. $\|\varphi_s(A)\|_\infty \leq (Nk+1)\|A\|_\infty$.

Homomorphic Properties. As GSW, TGSW inherits homomorphic properties. We can define the internal product between two TGSW samples and the external $\boxdot$ product already defined and used in [7, 12]. The external product is almost the GSW product [21], except that only one vector needs to be decomposed.

Definition 2.2 (External product). *We define the product* $\boxdot$ *as*

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

The following theorem on the noise propagation of the external product was shown in [12, Sec 3.2]:

Theorem 2.3 (External Product). *If A is a valid* TGSW *sample of message μ_A and $\boldsymbol{b}$ is 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 $\|\mathsf{Err}(A \boxdot \boldsymbol{b})\|_\infty \leq (k+1)\ell N\beta \|\mathsf{Err}(A)\|_\infty + \|\mu_A\|_1 (1 + kN)\epsilon + \|\mu_A\|_1 \|\mathsf{Err}(\boldsymbol{b})\|_\infty$ (worst case), where β and ϵ are the parameters used in the decomposition algorithm. If $\|\mathsf{Err}(A \boxdot \boldsymbol{b})\|_\infty \leq 1/4$ then $A \boxdot \boldsymbol{b}$ is a valid* TRLWE *sample. And assuming the heuristic 2.4, we have that $\mathsf{Var}(\mathsf{Err}(A \boxdot \boldsymbol{b})) \leq (k+1)\ell N\beta^2 \mathsf{Var}(\mathsf{Err}(A)) + (1+kN)\|\mu_A\|_2^2 \epsilon^2 + \|\mu_A\|_2^2 \mathsf{Var}(\mathsf{Err}(\boldsymbol{b}))$.*

There also exists an internal product between two TGSW samples, already presented in [21, 1, 19, 17, 12], and which consists in $(k+1)\ell$ independent $\boxdot$ products, and maps to the product of integer polynomials on plaintexts, and turns TGSW encryption into a ring homomorphism. Since we do not use this internal product in our constructions, so we won't detail it.

Independence heuristic All our average-case bounds on noise variances rely on the independence heuristic below. They usually corresponds to the square-root of the worst-case bounds which don't need this heuristic. As already noticed in [17], this assumption matches experimental results.

Assumption 2.4 (Independence Heuristic). We assume that all the error coefficients of TLWE or TGSW samples of the linear combinations we consider are independent and concentrated. In particular, we assume that they are σ-subgaussian where σ is the square-root of their variance.

Notations In the rest of the paper, the notation TLWE is used to denote the (scalar) binary TLWE problem, while for the ring mode, we use the notation TRLWE. TGSW is only used in ring mode with notation TRGSW, to keep uniformity with the TRLWE notation.

Sum-up of elementary homomorphic operations Table 1 summarizes the possible operations on plaintexts that we can perform in LHE mode, 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 mentionned, all samples contain noise, the user is free to discretize the message space accordingly to allow practical exact decryption. All these algorithms will be described in the next sections.

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.1)	$R^2\vartheta + n\log\left(\frac{1}{\alpha}\right)\mathrm{Cst}_{KS}$
	R-lipschitzian	$\mathsf{PrivKS}_{KS^{(f)}}(\mathfrak{c}_1,\ldots,\mathfrak{c}_p)$(Alg.2)	$R^2\vartheta + np\log\left(\frac{1}{\alpha}\right)\mathrm{Cst}_{KS}$
Ext. product	$b_1 \cdot \mu_2$	$C_1 \boxdot \boldsymbol{c_2}$ (Thm.2.3)	$b_1\vartheta_2 + \mathrm{Cst}_{\mathsf{TRGSW}}\vartheta_1$
CMux	$b_1?\mu_2 : \mu_3$	$\mathtt{CMux}(C_1, \boldsymbol{c_2}, \boldsymbol{c_3})$ (Lem.2.7)	$\max(\vartheta_2,\vartheta_3) + \mathrm{Cst}_{\mathsf{TRGSW}}\vartheta_1$
$\mathbb{T}$-non-linear	$X^{-\varphi(c_1)}\mu_2$	BlindRotate (Alg.3)	$\vartheta + n\mathrm{Cst}_{\mathsf{TRGSW}}$
Bootstrapping	$\mathrm{decrypt}(c)?m : 0$	Gate Bootstrapping (Alg.4)	Cst

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 $\mathbb{Z}[X]$.

2.2 Key switching revisited

In the following, we instantiate TRLWE and TRGSW with different parameter sets and we keep the same name for the variables $n, N, \alpha, \ell, B_g, \ldots$, but we alternate between bar over and bar under variables to differentiate input and output parameters. In order to switch between keys in different parameter sets, but also to switch between the scalar and polynomial message spaces $\mathbb{T}$ and $\mathbb{T}_N[X]$, we use slightly generalized notions of sample extraction and keyswitching. Namely, we give to keyswitching algorithms the ability to homomorphically evaluate linear morphisms f from any $\mathbb{Z}$-module $\mathbb{T}^p$ to $\mathbb{T}_N[X]$. We define two flavors, one for a publicly known f, and one for a secret f encoded in the keyswitching key. In the following, we denote $\mathsf{PubKS}(f, \mathsf{KS}, \mathfrak{c})$ and $\mathsf{PrivKS}(\mathsf{KS}^{(f)}, \mathfrak{c})$ the output of Algorithm 1 and Algorithm 2 on input the functional keyswitching keys KS and $\mathsf{KS}^{(f)}$ respectively and ciphertext $\mathfrak{c}$.

Algorithm 1 TLWE-to-TRLWE Public Functional Keyswitch

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

```
for i ∈ [[1, n]] do
    Let a_i = f(a_i^(1), ..., a_i^(p))
    let ã_i be the closest multiple of 1/2^t to a_i, thus ‖ã_i − a_i‖_∞ < 2^{−(t+1)}
    Binary decompose each ã_i = Σ_{j=1}^{t} ã_{i,j} · 2^{−j} where ã_{i,j} ∈ B_N[X]
end for
return (0, f(b^(1), ..., b^(p))) − Σ_{i=1}^{n} Σ_{j=1}^{t} ã_{i,j} × KS_{i,j}
```

Theorem 2.5. *(Public KeySwitch) Given p LWE ciphertexts $\mathfrak{c}^{(z)} \in \mathsf{TLWE}_{\mathfrak{K}}(\mu_z)$ and a public R-lipschitzian morphism f of $\mathbb{Z}$-modules, from $\mathbb{T}^p$ to $\mathbb{T}_{\underline{N}}[X]$, and $\mathsf{KS}_{i,j} \in \mathsf{TRLWE}_{\underline{K},\underline{\gamma}}(\frac{\mathfrak{K}_i}{2^j})$ with standard deviation $\underline{\gamma}$, Algorithm 1 outputs a* TRLWE *sample $\boldsymbol{c} \in \mathsf{TRLWE}_{\underline{K}}(f(\mu_1, \ldots, \mu_p))$ where:*
- $\|\mathsf{Err}(\boldsymbol{c})\|_\infty \leq R \|\mathsf{Err}(\mathfrak{c})\|_\infty + ntN\mathcal{A}_{\mathsf{KS}} + n2^{-(t+1)}$ (worst case),
- $\mathsf{Var}(\mathsf{Err}(\boldsymbol{c})) \leq R^2 \mathsf{Var}(\mathsf{Err}(\mathfrak{c})) + ntN\vartheta_{\mathsf{KS}} + n2^{-2(t+1)}$ (average case), where $\mathcal{A}_{\mathsf{KS}}$ and $\vartheta_{\mathsf{KS}} = \gamma^2$ are respectively the amplitude and the variance of the error of KS.

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 $\varphi_{\mathfrak{K}}(\mathfrak{c}) = -\mathfrak{K} \cdot \mathfrak{c}$. A detailed proof for both the private and the public keyswitching is given in the full version.

Theorem 2.6. *(Private KeySwitch) Given p* TLWE *ciphertexts $\mathfrak{c}^{(z)} \in \mathsf{TLWE}_{\mathfrak{K}}(\mu_z)$, $\mathsf{KS}_{i,j} \in \mathsf{TRLWE}_{\underline{K},\underline{\gamma}}(f(0, \ldots, \frac{\mathfrak{K}_i}{2^j}, \ldots, 0))$ where f is a private R-lipschitzian morphism of $\mathbb{Z}$-modules, from $\mathbb{T}^p$ to $\mathbb{T}_{\underline{N}}[X]$, Algorithm 2 outputs a* TRLWE *sample $\boldsymbol{c} \in \mathsf{TRLWE}_{\underline{K}}(f(\mu_1, \ldots, \mu_p))$ where*
- $\|\mathsf{Err}(\boldsymbol{c})\|_\infty \leq R \|\mathsf{Err}(\mathfrak{c})\|_\infty + (n+1)R2^{-(n+1)} + p(n+1)\mathcal{A}_{\mathsf{KS}}$ (worst-case),

Algorithm 2 TLWE-to-TRLWE Private Functional Keyswitch

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

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

- $\mathsf{Var}(\mathsf{Err}(\boldsymbol{c})) \leq R^2\,\mathsf{Var}(\mathsf{Err}(\mathfrak{c})) + (n+1)R^2 2^{-2(n+1)} + p(n+1)\vartheta_{\mathsf{KS}}$ *(average case), where* $\mathcal{A}_{\mathsf{KS}}$ *and* $\vartheta_{\mathsf{KS}} = \gamma^2$ *are respectively the amplitude and the variance of the error of* KS.

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. To that end, we will 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.

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

2.3 Gate bootstrapping overview

This lemma on the evaluation of the `CMux`-gate extends Thm 5.1, Thm 5.2 in [12] from the message space $\{0, 1/2\}$ to $\mathbb{T}_N[X]$:

Lemma 2.7 (`CMux` Gate). *Let* $\boldsymbol{d_1}, \boldsymbol{d_0}$ *be* TRLWE *samples and let* $C \in \mathsf{TRGSW}_{\boldsymbol{s}}(\{0,1\})$. *Then,* $\mathsf{msg}(\mathtt{CMux}(C, \boldsymbol{d_1}, \boldsymbol{d_0})) = \mathsf{msg}(C)\,?\mathsf{msg}(\boldsymbol{d_1}):\mathsf{msg}(\boldsymbol{d_0})$. *And we have* $\|\mathsf{Err}(\mathtt{CMux}(C, \boldsymbol{d_1}, \boldsymbol{d_0}))\|_\infty \leq \max(\|\mathsf{Err}(\boldsymbol{d_0})\|_\infty, \|\mathsf{Err}(\boldsymbol{d_1})\|_\infty) + \eta(C)$, *where* $\eta(C) = (k+1)\ell N\beta \|\mathsf{Err}(C)\|_\infty + (kN+1)\epsilon$. *Furthermore, under Assumption 2.4, we have:* $\mathsf{Var}(\mathsf{Err}(\mathtt{CMux}(C, \boldsymbol{d_1}, \boldsymbol{d_0}))) \leq \max(\mathsf{Var}(\mathsf{Err}(\boldsymbol{d_0})), \mathsf{Var}(\mathsf{Err}(\boldsymbol{d_1})))+ \vartheta(C)$, *where* $\vartheta(C) = (k+1)\ell N\beta^2\,\mathsf{Var}(\mathsf{Err}(C)) + (kN+1)\epsilon^2$.

The proof is the same as for Thm 5.1 and Thm 5.2 in [12] because the noise of the output does not depend on the value of the TRLWE message.

Blind rotate In the following, we give faster sub-routine for the main loop of Algorithm 3 in [12]. The improvement consists in a new `CMux` formula in the for loop of the algorithm 3 instead of the formula in algorithm 3 of [12]. The BlindRotate algorithm multiplies the polynomial encrypted in the input TRLWE ciphertext by an encrypted power of X. Theorem 2.8 follows from the fact that algorithm 3 calls p times the `CMux` evaluation from Lemma 2.7.

Algorithm 3 BlindRotate

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

Theorem 2.8. *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 standard deviation, $\mathfrak{K} \in \mathbb{B}^n$ be a* TLWE *secret key and $K \in \mathbb{B}_N[X]^k$ be its* TRLWE *interpretation. Given one sample $\boldsymbol{c} \in \mathsf{TRLWE}_K(\boldsymbol{v})$ 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 3 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 $\vartheta_C = \alpha^2$ and $\mathcal{A}_C$ are the variance and amplitudes of $\mathsf{Err}(C_i)$.

We define $\mathsf{BlindRotate}(\boldsymbol{c}, (a_1, \ldots, a_p, b), C)$, the procedure described in Algorithm 3 that outputs the TLWE sample ACC as in Theorem 2.8.

Algorithm 4 Gate Bootstrapping TLWE-to-TLWE (calling algorithm 3)

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 $\overline{\mathfrak{c}} = (\bar{\mathfrak{a}}, \bar{\mathfrak{b}}) \in \mathsf{TLWE}_{\bar{\mathfrak{K}},\bar{\eta}}(x \cdot \mu_1)$
Let $\mu = \frac{1}{2}\mu_1 \in \mathbb{T}$ (Pick one of the two possible values)
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** $i \in [\![1, \underline{n}]\!]$
Let $v := (1 + X + \ldots + X^{\bar{N}-1}) \cdot X^{\frac{\bar{N}}{2}} \cdot \mu \in \mathbb{T}_{\bar{N}}[X]$
$\mathsf{ACC} \leftarrow \mathsf{BlindRotate}((\mathbf{0}, v), (a_1, \ldots, a_n, b), (\mathrm{BK}_1, \ldots, \mathrm{BK}_n))$
Return $(\mathbf{0}, \mu) + \mathsf{SampleExtract}(\mathsf{ACC})$

Gate Bootstrapping (TLWE-to-TLWE)

Theorem 2.9 (Gate Bootstrapping (TLWE-to-TLWE)). *Let* $\bar{H} \in \mathcal{M}_{(\bar{k}+1)\bar{\ell},\bar{k}+1}(\mathbb{T}_{\bar{N}}[X])$ *the gadget matrix 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* $BK_i \in \overline{\mathsf{TRGSW}}_{\bar{K},\bar{\alpha}}(\underline{\mathfrak{K}}_i)$ *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 4 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.* $\mathsf{Var}(\mathsf{Err}(BK_{\underline{\mathfrak{K}} \to \bar{K},\bar{\alpha}})) = \bar{\alpha}^2$*.*

Sketch of Proof. Algorithm 4 is almost the same as Algorithm 3 in [12] except that the main loop has been put in a separate algorithm (Algorithm 3) at line 2. In addition, the final KeySwitching has been removed which suppresses two terms in the norm inequality of the error. Note that the output is encrypted with the same key as the bootstrapping key. Another syntactic difference is that the input sample is a multiple of $1/2N$ (which can be achieved by rounding all its coefficients). Also, a small difference in the way we associate `CMux` operations removes a factor 2 in the noise compared to the previous gate bootstrapping procedure, and it is also faster.

Homomorphic operations (revisited) *via* Gate Bootstrapping. The fast bootstrapping of [17] and improved in [4] [12] is presented for `Nand` gates. They evaluate a single `Nand` operation and they refresh the result to make it usable for the next operations. Other elementary gates are presented: the `And`, `Or`, `Xor` (and trivially `Nor`, `Xnor`, `AndNot`, etc. since `NOT` is cheap and noiseless). The term *gate bootstrapping* refers to the fact that this fast bootstrapping is performed after every gate evaluation.

The ternary `Mux` gate ($\texttt{Mux}(c, d_0, d_1) = c?d_1 : d_0 = (c \wedge d_1) \oplus ((1-c) \wedge d_0)$, for $c, d_0, d_1 \in \mathbb{B}$) is generally expressed as a combination of 3 binary gates. As already mentioned in [17], we can improve the `Mux` evaluation by performing the middle $\oplus$ as a regular addition before the final 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, and is compensated by the fact that we save a factor 2 in the gate bootstrapping in the blind rotation from Algorithm 3. Overall, the ternary `Mux` gate can be evaluated in FHE mode by evaluating only

Actually, the gate bootstrapping technique can be used even if we do not need to evaluate a specific gate, but just to refresh noisy ciphertexts.

two gate bootstrappings and one public keyswitch. We call this procedure *native* `MUX`, which computes:

- $c \wedge d_1$ via a gate bootstrapping (Alg. 4) of $(\mathbf{0}, -\frac{1}{8}) + \boldsymbol{c} + \boldsymbol{d_1}$;
- $(1-c) \wedge d_0$ via a gate bootstrapping (Alg. 4) of $(\mathbf{0}, \frac{1}{8}) - \boldsymbol{c} + \boldsymbol{d_0}$;
- a final keyswitch on the sum (Alg. 1) which dominates the noise.

This native `Mux` is therefore bootstrappable with the same parameters as in [12]. More details are given in the full version. 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 with any binary gates and/or the native `Mux`, and we use the following experimental timings (see Section 5):

Gate bootstrapping mode	
Pre-bootstrap 1 bit	$t_{GB} = 13ms$
Time per any binary gate (`And`, `Or`, `Xor`, ...)	$t_{GB} = 13ms$
Time per `MUX`	$2t_{GB} = 26ms$

3 Leveled Homomorphic circuits

Various packing techniques have already been proposed for homomorphic encryption, for instance the Lagrange embedding in Helib [23, 22], the diagonal matrices encoding in [27] or the CRT encoding in [2]. 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 has 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 this paper, packing refers to the canonical coefficients embedding function, that maps N Scalar-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) SampleExtract procedure. Reciprocally, we can repack, move data across the slots, or clear some slots by using our public functional key switching from Algorithm 1 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 2.5, 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 [2, 10, 15], our technique relies on a

circular security assumption, even in the leveled mode since our keyswitching key encrypts its own key bits.

We now analyse how packing can speed-up TGSW leveled computations, first for lookup tables or arbitrary functions, and then for most arithmetic functions.

3.1 Arbitrary functions and Look-Up Tables

The first class of functions that we analyse are arbitrary functions $f : \mathbb{B}^d \to \mathbb{T}^s$. Such functions can be expressed with a Look-Up Table (LUT), containing a 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).

In order to compute $f(x)$, 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 [8, 12] consists in evaluating the s subfunctions separately, and each of them is 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 1 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.

Horizontal packing corresponds exactly to batching. In fact, it exploits the fact that the s subfunctions evaluate the same `CMux` tree with the same inputs on different data, which are 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 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 and obtain the s slots all at once, which is overall s times faster than the classical evaluation. Our *vertical packing* is very different from the batching technique. 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.

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 was already given in [19] and [12].

Horizontal Packing 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

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

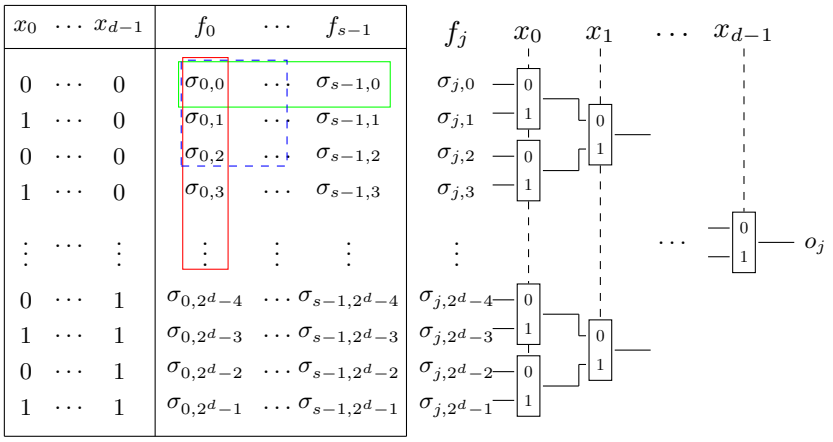


Fig. 1. LUT with `CMux` tree - Intuitively, the horizontal rectangle encircles the bits packed in the horizontal packing, while the vertical rectangle encircles the bits packed in the the vertical packing. The dashed square represents the packing in the case where the two techniques are mixed. The right 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.

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 $\leq N$, the number of gates to evaluate remains $2^d - 1$, which is only s times smaller than the non-packed approach, and is not advantageous if s is small. Lemma 3.1 specifies the noise propagation and it follows immediately from lemma 2.7 and from the construction of the binary decision `CMux` tree, which has depth d.

Lemma 3.1 (Horizontal Packing). *Let $\boldsymbol{d}_0, \ldots, \boldsymbol{d}_{2^d-1}$ be* TRLWE *samples 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 1). Then, using the same notations as in lemma 2.7 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.*

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

Algorithm 5 Vertical Packing LUT of $f_j : \mathbb{B}^d \to \mathbb{T}$ (calling algorithm 3)

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 $\mathbf{c} \in \mathsf{TLWE}_{\mathfrak{K}}(o_j = f_j(x))$, 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 $\mathbf{c} = \mathsf{SampleExtract}(\boldsymbol{d})$

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 3) to find the element inside the box.

We now explain how 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 1. For retrieving the output of $f_j(x)$, we just have to return the LUT value $\sigma_{j,x}$ in 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 $\sigma_{j,kN+0}, \ldots, \sigma_{j,kN+N-1}$. To retrieve $f_j(x)$, we first need to select the block that contains $\sigma_{j,x}$. This block has the index $p = \lfloor x/N \rfloor$, whose bits are the $d-\delta$ most significant bits of x. Since the TRGSW encryption 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 (Alg. 3) 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. Alg. 5 details the evaluation of $f_j(x)$.

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 3.2 (Vertical Packing LUT of f_j). *Let $f_j : \mathbb{B}^d \to \mathbb{T}$ be a sub-function of the arbitrary function f, with LUT values $\sigma_{j,0}, \ldots, \sigma_{j,2^d-1}$. Let $\boldsymbol{d}_0, \ldots, \boldsymbol{d}_{\frac{2^d}{N}-1}$ be* TRLWE *samples, such that $\boldsymbol{d}_p \in \mathsf{TRLWE}_K(\sum_{i=0}^{N-1} \sigma_{j,pN+i} X^i)$*

for $p \in [\![0, \frac{2^d}{N} - 1]\!]$. *Let* $C_0, \ldots, C_{d-1}$ *be* TRGSW *samples, such that* $C_i \in \mathsf{TRGSW}_K(x_i)$, *with* $x_i \in \mathbb{B}$ *and* $i \in [\![0, d-1]\!]$.

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 2.7 and 2.8, 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. (Sketch) The proof follows immediately from the results of lemma 2.7 and theorem 2.8, 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 1. As previously mentioned, the horizontal and vertical packing techniques can be mixed together to improve the evaluation of f in the case where s and d are both small, i.e. the previous two methodology cannot be applied separately but we have $2^d \cdot s > N$. In particular, if we pack $s = x$ 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.

3.2 Arithmetic operations via Weighted Automata

In [12], the arithmetic operations were evaluated via deterministic finite automata using `CMux` gates. It was made possible thanks to the fact that the messages were binary. In this paper, the samples on which we perform the arithmetic operations pack several torus values together. A more powerful tool is thus needed to manage the evaluations in an efficient way. Deteministic weighted finite automata (det-WFA) are deterministic finite automata where each transition contains an additional weight information. By reading a word, the outcome of a det-WFA is the sum of all weights encountered along the path (here, we work with an additive group), whereas the outcome of a deterministic finite automata (DFA) is just a boolean that states whether the destination state is accepting. The weights of a det-WFA can be seen as a memory that stores the bits of the partial result, all along the evaluation path. Let's take for example the evaluation of the MAX circuit, that takes in input two d-bit integers and returns the maximal value between them. With DFA, to retrieve all the d bits of the result we need d different automata, for a total of $O(d^2)$ transitions. By

If the sub-function f_j and its LUT are public, the LUT values $\sigma_{j,0}, \ldots, \sigma_{j,2^d-1}$ can be given in clear. This means that the TRLWE samples $\boldsymbol{d}_p$, 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.

introducing the weights, all the bits of the result are given in one pass after only $O(d)$ transitions. To our knowledge, our paper is the first one introducing this tool on the FHE context. In this section, 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 [9] and [16] for further details on the theory of weighted automata.

Definition 3.3 (Deterministic weighted finite automata (det-WFA)). *A deterministic weighted finite automata (det-WFA) over a group $(S, \oplus)$ is a tuple $\mathfrak{A} = (Q, i, \Sigma, \mathcal{T}, F)$, 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 and $F \subseteq Q$ is the set of final states. 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 is accepted by the det-WFA $\mathfrak{A}$ if $q_0 = i$ and $q_d \in F$. The weight $w(\boldsymbol{\sigma})$ of a word $\boldsymbol{\sigma}$ is 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. Note that every label induces a single path (i.e. there is only one possible path per word).

Remark 2. In our applications, we fix the alphabet $\Sigma = \mathbb{B}$. Definition 3.3 restraints the WFA to the deterministic (the non-deterministic case is not supported), complete and universally accepting case (i.e all the words are accepted). In the general case, the additive group would be replaced by a semi-ring $(S, \oplus, \otimes, 0, 1)$. In the rest of the paper we set $(S, \oplus)$ as $(\mathbb{T}_N[X], +)$.

Theorem 3.4 (Evaluation of det-WFA). *Let $\mathfrak{A} = (Q, i, \mathbb{B}, \mathcal{T}, F)$ 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 $d \cdot |Q|$ $\mathtt{CMux}$ gates, we output a TRLWE sample $\boldsymbol{d}$ that encrypts the weight $w(\boldsymbol{\sigma})$, such that (using the same notations as in lemma 2.7)*
- $\|\mathsf{Err}(\boldsymbol{d})\|_\infty \leq d \cdot ((k+1)\ell N \beta \mathcal{A}_{\mathsf{TRGSW}} + (kN+1)\epsilon)$ (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 $\mathtt{CMux}$ to evaluate decreases to $|Q|$.

Proof. (Sketch) This theorem generalizes theorem 5.4 of [12] for det-WFA. The automaton is still evaluated from the last letter σ_{d-1} to the first one σ_0, using one TRLWE ciphertext $\boldsymbol{c}_{j,q}$ per position $j \in [\![0, d-1]\!]$ in the word and per state $q \in Q$. Before reading a letter, all the TRLWE samples $\boldsymbol{c}_{d,q}$, for $q \in Q$, are initialized to zero. When processing the j-th letter σ_j, each pair of transitions $q \xrightarrow{0,\nu_0} q_0$ and $q \xrightarrow{1,\nu_1} q_1$ is evaluated as $\boldsymbol{c}_{j,q} = \mathtt{CMux}(C_j, \boldsymbol{c}_{j+1,q_1} + (\mathbf{0}, \nu_1), \boldsymbol{c}_{j+1,q_0} + (\mathbf{0}, \nu_0))$.

The final result is $\boldsymbol{c}_{0,i}$, which encodes $w(\boldsymbol{\sigma})$ by induction on the CMux graph. Since translations are noiseless, the output noise corresponds to a depth-d of CMux. Like in [12], the last condition implies that only $|Q|$ of the $d|Q|$ CMux are accessible and need to be evaluated. □

MAX In order to evaluate the MAX circuit of two d-bit integers, $x = \sum_{i=0}^{d-1} x_i 2^i$ and $y = \sum_{i=0}^{d-1} y_i 2^i$, we construct a det-WFA that takes in input all the bits $x_{d-1}, \ldots, x_0$ of x and $y_{d-1}, \ldots, y_0$ of 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 2 has 3 principal states (A, B, E) and 4 intermediary states ((A), (B), $(E,1)$, $(E,0)$), which keeps 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. Overall, the next lemma, which is a direct consequence of Theorem 3.4, shows that the Max can be computed by evaluating only $5d$ CMux gates, instead of $\Theta(d^2)$ with classical deterministic automata.

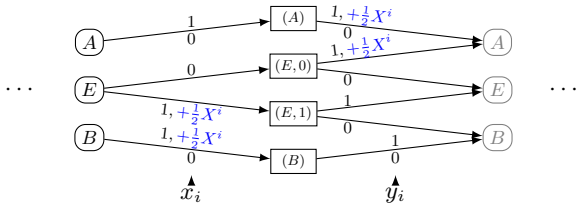


Fig. 2. Max: det-WFA - The states A and (A) mean that y is the maximal value, the states B and (B) mean that x is the maximal value, and finally, the states E, $(E,1)$ and $(E,0)$ mean that x and y are equals on the most significant bits. If the current state is A or B, the following state will stay 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.

Remark 3. 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:

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

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

Lemma 3.5 (Evaluation of Max det-WFA). *Let $\mathfrak{A}$ be the det-WFA of the Max, described in Figure 2. 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 $5d$ $\mathtt{CMux}$ 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 2.7)*
- $\|\mathsf{Err}(\boldsymbol{d})\|_\infty \leq 2d \cdot ((k+1)\ell N\beta\mathcal{A}_{\mathsf{TRGSW}} + (kN+1)\epsilon)$ (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.

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 $y = y_1y_0$. 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 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 figure 3) has 6 main states (i, c_0, c_{10}, c_{11}, c_{20}, c_{21}), plus 14 intermediary states that store the last bit read (noted with capital letters and parenthesis). The value of the i-th output bit is put in a weight on the last transition of each column.

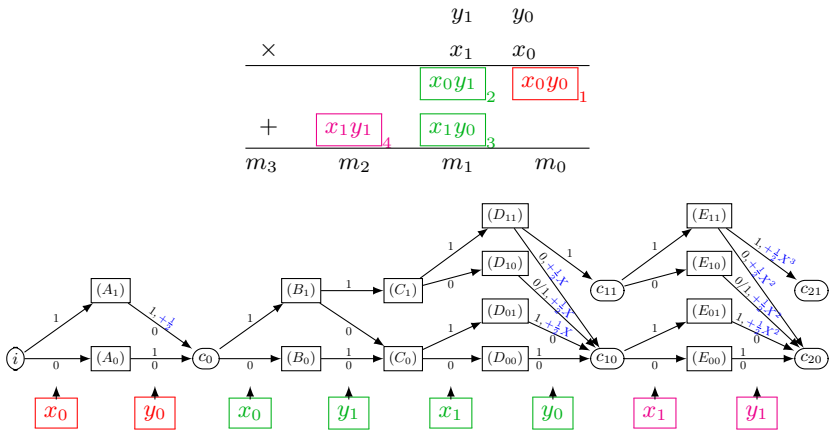


Fig. 3. Schoolbook 2-bits multiplication and corresponding det-WFA

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).

3.3 TBSR counter techniques

We now present another design which is specific to the multi-addition (or its derivatives), but which is faster than the generic construction with weighted automata. 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 and
3. Integer division of the value by 2.
We will now explain the basic idea, and then, we will show how to implement it efficiently on TRLWE ciphertexts.

For $j \in [0, p = \log_2(N)]$ and $k, l \in \mathbb{Z}$, we call $B_{j,k}^{(l)}$ the j-th bit of $k + l$ in the little endian signed binary representation. The latter form very simple binary sequence: $B_0^{(0)} = (0, 1, 0, 1, ...)$ is 2-periodic, $B_1^{(0)} = (0, 0, 1, 1, 0, 0, 1, 1...)$ is 4-periodic, more generally, for all $j \in [0, p]$ and $l \in \mathbb{Z}$, $B_j^{(l)}$ is 2^j-antiperiodic, and 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. And most importantly, 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 $l + k \mod 2N$. We now suppose that an integer l in $[0, N - 1]$ is represented by its Bit Sequence Representation, defined as $BSR(l) = [B_0^{(l)}, \ldots, B_p^{(l)}]$. And we see how to compute $BSR(l + 1)$ and $BSR(\lfloor l/2 \rfloor)$ using only copy and negations operations on bits at a fixed position which does not depend on l (blind computation). Then, we will see how to represent these operations homomorphically on TRLWE ciphertexts.

Increment: Let $U = [u_0, \ldots, u_p]$ be the BSR of some unknown number $l \in [0, N - 1]$. Our goal is to compute $V = [v_0, \ldots, v_p]$ which is the BSR of $l + 1$. Again, we recall that it suffices to define the sequence v_i 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: $v_{j,k} := u_{j,k+1}$ for all $k \in [0, N - 1]$. Indeed, this operation transforms each $B_{j,k}^{(l)}$ into $B_{j,k+1}^{(l)} = B_{j,k}^{(l+1)}$, and the output V is the BSR of $l + 1$.

Integer division by two: Let $U = [u_0, \ldots, u_p]$ be the BSR of some unknown number $l \in [0, N - 1]$. Our goal is to compute $V = [v_0, \ldots, v_p]$ which is the BSR of $\lfloor \frac{l}{2} \rfloor$. First, we note that the integer division by 2 corresponds to a right shift over the bits. Thus for $j \in [0, p-1]$ and $k \in \mathbb{N}$, we can set $v_{j,k} = u_{j+1,2k}$. Indeed,

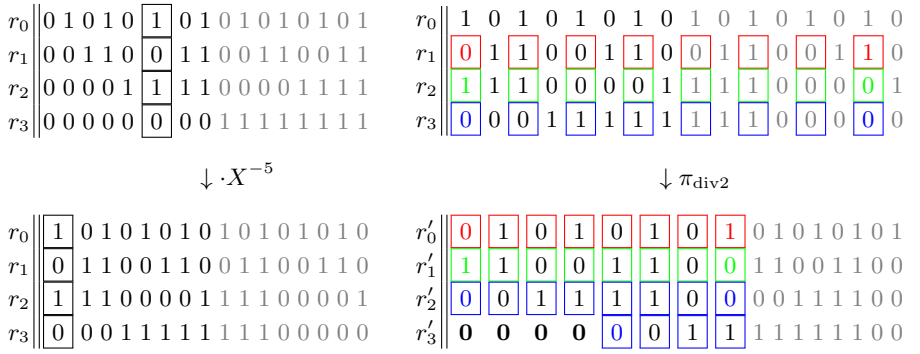


Fig. 4. TBSR - example of addition +5 and division by 2.

$u_{j+1,2k}$ is the $j+1$-th bit of $l+2k$ is the j-th bit of its half $\lfloor l/2 \rfloor + k$, which is our desired $v_{j,k} = B_{j,k}^{(\lfloor l/2 \rfloor)}$. This is unfortunately not enough to reconstruct the last sequence v_p, since we have no information on the $p+1$-th bits in U. 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 $v_{p,k} = 0$. Then, we just need to note that $(u_{p,0}, \ldots, u_{p,N-1})$ is $N-l$ times 0 followed by l times 1, and our target $(v_{p,N/2}, \ldots, v_{p,N-1})$ 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 $(u_{p,0}, u_{p,2}, \ldots, u_{p,N-2})$. To summarize, division by 2 corresponds to the following blind transformation:

$$\begin{cases} v_{j,k} & = u_{j+1,2k} \text{ for } j \in [0, p-1], k \in [0, N-1] \\ v_{p,k} & = 0 \text{ for } k \in [0, \frac{N}{2} - 1] \\ v_{p,N/2+k} & = u_{p,2k} \text{ for } k \in [0, \frac{N}{2} - 1] \end{cases}$$

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. Furthermore, since the cyclic shift of coefficients is heavily used in the increment operation, we would like to make it correspond to the multiplication by X, which has a similar behaviour on coefficients of torus polynomials. Therefore, this is our basic encoding of the BSR sequences: Let $U = [u_0, \ldots, u_p]$ be the BSR of some unknown number $l \in [0, N-1]$, For $j \in [0, p-1]$, we represent u_j with the polynomial $\mu_i = \sum_{k=0}^{N-1} \frac{1}{2} u_{j,k} X^k$, and we represent the last u_p with the polynomial $\mu_p = \sum_{k=0}^{N-1} (\frac{1}{2} u_{p,k} - \frac{1}{4}) X^k$. This simple rescaling between the bit representation U and the torus representation $M = [\mu_0, \ldots, \mu_p]$ is bijective. Using this encoding, the integer division transformation presented above immediately rewrites into this affine function, which 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 3.6 (TBSR encryption).

- *Params and keys:* TRLWE *parameters N with secret key $K \in \mathbb{B}_N[X]$, and a circular-secure keyswitching key $KS_{K\to K,\gamma}$ from K to itself, noted just KS.*
- TBSRSet(l)*: return a vector of trivial* TRLWE *ciphertexts encoding the torus representation of $[B_0^{(l)},\ldots,B_p^{(l)}]$.*
- TBSRBitExtract$_j(C)$*: Return* SampleExtract$_0(c_j)$ *when $j < p$.*
- TBSRIncrement(C)*: Return $X^{-1}.C$.*
- TBSRDiv2(C)*: Use KS to evaluate π_{div2} homomorphically on C. Since it is a 1-lipschitzian affine function, this means: apply the public functional* KeySswitch *to KS, the linear part of π_{div2} and C, and then, translate the result by the constant part of π_{div2}.*

Theorem 3.7 (TBSR operations). *Let N,K, and KS be TBSR parameters..., and C a TBSR ciphertext of l with noise amplitude η (or noise variance ϑ). Then for $j \in [0,p-1]$,* TBSRBitExtract$_j(C)$ *is a $LWE_{\mathfrak{K}}$ 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 $l \leq N-2$,* TBSRIncrement(C) *is a TBSR ciphertext of $l+1$ with noise amplitude (resp. variance) $\leq \eta$ (resp. $\leq \vartheta$). $C' =$* TBSRDiv2(C) *is a TBSR ciphertext of $\lfloor l/2 \rfloor$ such that:*
- $\|\mathsf{Err}(C')\|_\infty \leq \mathcal{A} + N^2 t \mathcal{A}_{\mathsf{KS}} + N 2^{-(t+1)}$ (worst-case);
- $\mathsf{Var}(\mathsf{Err}(C')) \leq \vartheta + N^2 t \vartheta_{\mathsf{KS}} + N 2^{-2(t+1)}$ (average case).

Proof. (sketch) Correctness has already been discussed, the noise corresponds to the application of a public keyswitch on the same key: with $n = N$.

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. This leads to a leveled multiplication circuit with KeySwitching which is quadratic instead of cubic with weighted automata.

Lemma 3.8. *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

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

ciphertexts. and 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* $\mathcal{A}_A$ *(resp. variance* ϑ_A*).*

Then, there exists an algorithm (see the full version for more details) that computes all the bits of the product within $2d^2p$ *CMux and* $(2d-2)p$ *public keyswitch, and the output ciphertexts satisfy:*
- $\|\mathsf{Err}(Out)\|_\infty \leq 2d^2((k+1)\ell N\beta\mathcal{A}_A+(kN+1)\epsilon)+(2d-2)(N^2t\mathcal{A}_{\mathsf{KS}}+N2^{-(t+1)})$;
- $\mathsf{Var}(\mathsf{Err}(Out)) \leq 2d^2((k+1)\ell N\beta^2\vartheta_A + (kN+1)\epsilon^2) + (2d-2)(N^2t\vartheta_{\mathsf{KS}} + N2^{-2(t+1)})$.

4 Combining leveled with bootstrapping

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 [17, 12], and the external product 2.3 from [12, 7] has contributed to accelerate leveled operations. Unfortunately, these bootstrapping cannot output GSW ciphertexts. Previous solutions proposed in [21, 1, 19] 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 μh_i, where 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 [12] 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.

4.1 Circuit Bootstrapping (TLWE-to-TRGSW)

Our circuit bootstrapping, detailed in algorithm 6, 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

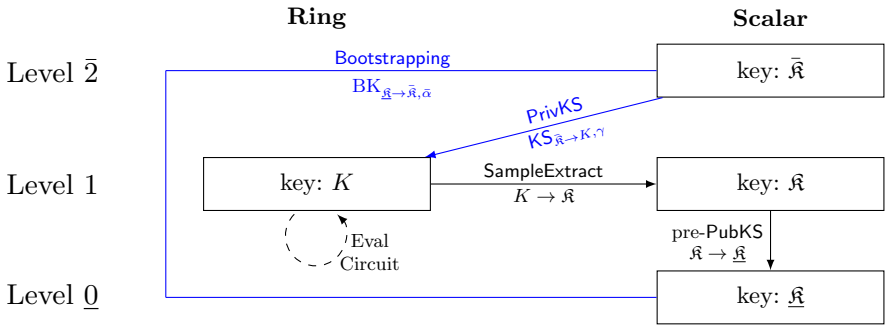


Fig. 5. The figure represents the three levels of encryption on which our construction shifts. The arrows show the operations that can be performed inside each level or how to move from a level to another. In order to distinguish the objects with respect to their level, we adopted the intuitive notations "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).

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, $\underline{\alpha} \geq 2^{-11}$). 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, $\alpha \geq 2^{-30}$). 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 [12]. Level 2 corresponds to ciphertexts with small noise (typically, $\bar{\alpha} \geq 2^{-50}$). This level corresponds to the limit of what can be mapped over native 64-bit operations. Practical values and details are given in section 5.

Our circuit bootstrapping consists in three parts:

–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.

–TLWE-**to-**TLWE **Bootstrapping** (algorithm 4): 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 4) on $\underline{\mathbf{c}}$, to obtain ℓ TLWE ciphertexts $\bar{\mathbf{c}}^{(1)}, \ldots, \bar{\mathbf{c}}^{(\ell)}$ where $\bar{\mathbf{c}}^{(w)} \in \mathsf{TLWE}_{\bar{\mathfrak{K}},\bar{\eta}}(\mu \cdot \frac{1}{\bar{B}_g{}^w})$, 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

Algorithm 6 Circuit Bootstrapping (calling algorithms 4 and 2)

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 $u \leq k$, 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 $C = (\boldsymbol{c}^{(u,w)})_{1\leq u\leq k+1, 1\leq w\leq \ell}$

$\bar{\alpha}$, we expect the output noise $\bar{\eta}$ to remain smaller than level 1 value.

–TLWE**-to-**TRLWE **private key-switching** (algorithm 2): 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 (Alg.2).

Theorem 4.1 (Circuit Bootstrapping Theorem). *Let $n, \alpha, N, k, B_g, \ell, H, \epsilon$ denote TRLWE/TRGSW level 1 parameters, and the same variables names with underbars/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 6 outputs a level 1 TRGSW sample $C \in \mathsf{TRGSW}_K(\mu)$ such that*

- $\|\mathsf{Err}(C)\|_\infty \leq \underline{n}(\bar{k}+1)\bar{\ell}\bar{N}\bar{\beta}\mathcal{A}_{BK} + \underline{n}(1+\bar{k}\bar{N})\bar{\epsilon} + \bar{n}2^{-(t+1)} + \bar{n}t\mathcal{A}_{\mathsf{KS}}$ *(worst);*
- $\mathsf{Var}(\mathsf{Err}(C)) \leq \underline{n}(\bar{k}+1)\bar{\ell}\bar{N}\bar{\beta}^2\bar{\vartheta}_{BK} + \underline{n}(1+\bar{k}\bar{N})\bar{\epsilon}^2 + \bar{n}2^{-2(t+1)} + \bar{n}t\vartheta_{\mathsf{KS}}$ *(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. (sketch) The output TRGSW ciphertext is correct, because by construction, the i-th TRLWE component $\boldsymbol{c}^{(u,w)}$ has the correct message $\mathsf{msg}(\mu \cdot H_i) =$

$\mu K_u/B_g^w$. $\boldsymbol{c}^{(u,w)}$ is obtained by chaining one TLWE-to-TLWE bootstrapping (algorithm 4) with one private key switchings, as in algorithm 2. The values of maximal amplitude and variance of $\mathsf{Err}(C)$ are directly obtained from the partial results of lemma 2.9 and theorem 2.6. In total, Algorithm 6 performs exactly ℓ bootstrappings (Algorithm 4), and $\ell(k+1)$ private key switchings (Algorithm 2). □

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 [19]. 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 [8] 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 [19] require a quadratic number of internal level 2 TRGSW products, which is much slower than what we propose. Finally, the CRT variant in [1] and [19] 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.

5 Comparison and practical parameters

We now explicit the practical parameters for our scheme, and we give the running time comparison for the evaluation of the homomorphic circuits described before in LHE and FHE mode (with or without the new optimization techniques).

In [12] the timing for the gate bootstrapping was $52ms$. We improved it to $13ms$: a speed up of a factor 2 is due to the dedicated assembly FFT for $X^N + 1$ in double precision. An additional speed ups (by a factor 1.5) is due to a new choice of parameters, for the same security level (in particular the ℓ TRGSW parameter is reduced to 2 instead of 3). Finally, we replaced the core of the gate bootstrapping with the simpler CMux and blindRotate (Algorithm 3)

described in Section 2, which gives the last 1.33x speed-up. For the same reason, the external product is now executed in $34\mu s$. We added these optimizations to the public repository of the TFHE library [14]. A experimental measurement of the noise confirmed the average case bounds on the variance, predicted under the independance assumption.

As a consequence, all binary gates are executed in $13ms$, and the native bootstrapped MUX (also described in Section 2) gate takes $26ms$ on a 64-bit single core (i7-4910MQ) at 2.90GHz. 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 [14]. 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.

Concrete Parameters In our three levels, we used the following TRLWE and TRGSW parameter sets, which have at least 110-bits of security, according to the security analysis in [12].

Level	**Minimal noise** α	n	B_g	ℓ
0	$\underline{\alpha} = 2^{-15.33}$	$\underline{n} = 500$	N.A.	N.A.
1	$\alpha = 2^{-32.33}$	$n = 1024$	$B_g = 2^8$	$\ell = 2$
2	$\bar{\alpha} = 2^{-45.33}$	$\bar{n} = 2048$	$\bar{B_g} = 2^9$	$\bar{\ell} = 4$

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	γ	**Usage**
$1 \to 0$	$\underline{t} = 12$	$\underline{\gamma} = 2^{-14}$	Circuit Bootstap, Pre-KS
$2 \to 1$	$\bar{t} = 30$	$\bar{\gamma} = 2^{-31}$	Circuit Bootstap, Step 4 in Alg. 6
$1 \to 1$	$t = 24$	$\gamma = 2^{-24}$	TBSR

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

Output TLWE	**Fresh** TRGSW **in LHE**	TRGSW **Output of CB**	**Bootst. key**
$\vartheta \leq 2^{-10,651}$	$\vartheta = 2^{-60}$	$\vartheta \leq 2^{-47.03}$	$\vartheta_{BK} = 2^{-88}$

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	$t_{XP} = 34\mu s$	$2^{-20.86}$	3466
PubKS$_{TBSR}$	$t_{KS} = 180ms$	$2^{-23.42}$	16384

More details on these parameter choices are provided in the full version.

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. 6), all inputs are fresh ciphertexts (either TRLWE or TRGSW) and we compare the previous versions [12] (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. 6), 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.

Conclusion

In this paper we improved the efficiency of TFHE, by proposing some new packing techniques. For the first time we use det-WFA in the context of homomorphic encryption to optimize the evaluation of arithmetic circuits, and we introduced the TBSR counter. By combining these optimizations, we obtained a significant timing speed-up and decrease the ciphertext overhead for TLWE and TRGSW based encryption. We also solved the problem of non universal composability of TFHE leveled gates, by proposing the efficient circuit bootstrapping that runs in 134ms; we implemented it in the TFHE project [14].

References

1. J. Alperin-Sheriff and C. Peikert. Faster bootstrapping with polynomial error. In *Crypto*, pages 297–314, 2014.
2. D. Benarroch, Z. Brakerski, and T. Lepoint. Fhe over the integers: Decomposed and batched in the post-quantum regime. Cryptology ePrint Archive, 2017/065.
3. F. Benhamouda, T. Lepoint, C. Mathieu, and H. Zhou. Optimization of bootstrapping in circuits. In *ACM-SIAM*, pages 2423–2433, 2017.
4. J. Biasse and L. Ruiz. FHEW with efficient multibit bootstrapping. In *LATINCRYPT 2015*, pages 119–135, 2015.
5. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.
6. 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.

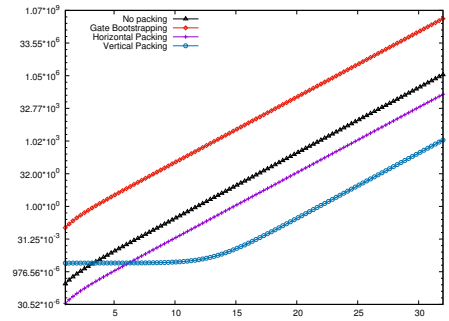
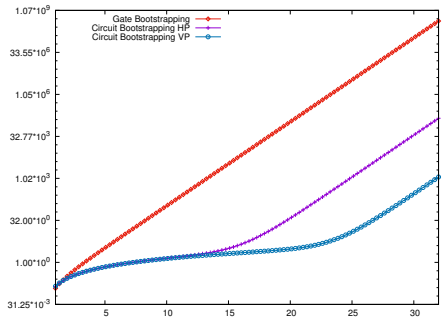
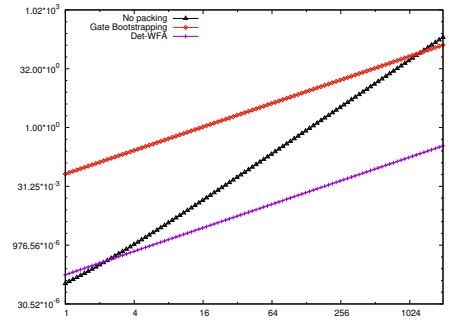
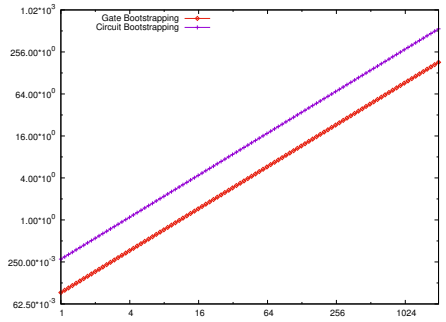
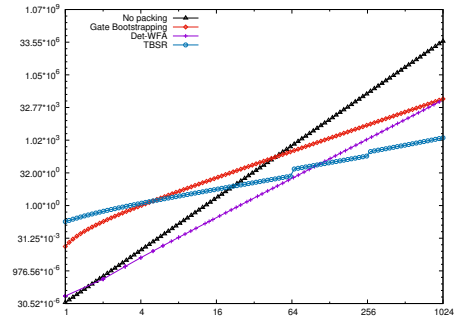
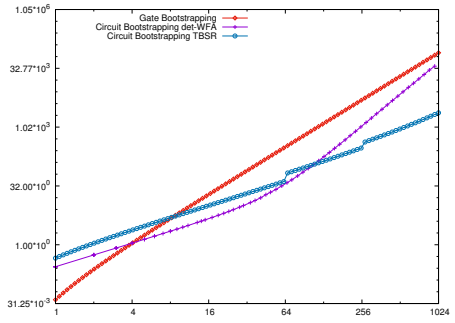
Leveled homomorphic encryption	Fully homomorphic encryption
	
(a.) Leveled LUT (d-bits input and $s = 8$-bits output) - No packing: $s(2^d - 1)t_{XP}$ - Gate bootst.: $s(2^d - 1)t_{GB}$ - Horizontal Packing (HP): $(2^d - 1)t_{XP}$ - Vertical Packing (VP): $s(2^d/N - 1 + \log N)t_{XP}$.	**(b.)** LUT (d-bits 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. 6. 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).

7. Z. Brakerski and R. Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In *Crypto'2016*, volume 9814, pages 190–213, 2016.
8. Z. Brakerski and V. Vaikuntanathan. Lattice-based FHE as secure as PKE. In *ITCS*, pages 1–12, 2014.
9. 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.
10. 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*.
11. J. H. Cheon and D. Stehlé. Fully homomophic encryption over the integers revisited. In *EUROCRYPT 2015*, pages 513–536. 2015.
12. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *ASIACRYPT 2016*, pages 3–33. Springer, 2016.
13. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. A homomorphic lwe based e-voting scheme. In *PQ Cryptography*, pages 245–265. Springer, 2016.
14. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. TFHE: Fast fully homomorphic encryption library. https://tfhe.github.io/tfhe/, August 2016.
15. J. Coron, T. Lepoint, and M. Tibouchi. Scale-invariant fully homomorphic encryption over the integers. In *PKC 2014*, pages 311–328, 2014.
16. M. Droste and P. Gastin. Weighted automata and weighted logics. In *Handbook of weighted automata*, pages 175–211. Springer, 2009.
17. L. Ducas and D. Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In *Eurocrypt*, pages 617–640, 2015.
18. J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. https://eprint.iacr.org/2012/144, 2012.
19. N. Gama, M. Izabachène, P. Q. Nguyen, and X. Xie. Structural lattice reduction: Generalized worst-case to average-case reductions. *(EUROCRYPT 2016) ePrint Archive, 2014/283*.
20. C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.
21. C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Crypto'13*.
22. S. Halevi and I. V. Shoup. Helib - an implementation of homomorphic encryption. https://github.com/shaih/HElib/, September 2014.
23. S. Halevi and V. Shoup. Algorithms in helib. In *Crypto'2014*, pages 554–571.
24. T. Lepoint. FV-NFLlib: Library implementing the Fan-Vercauteren homomorphic encryption scheme. https://github.com/CryptoExperts/FV-NFLlib, May 2016.
25. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, pages 1–23, 2010.
26. M. Paindavoine and B. Vialla. Minimizing the number of bootstrappings in fully homomorphic encryption. In *SAC*, pages 25–43, 2015.
27. M. A. R. Hiromasa and T. Okamoto. Packing messages and optimizing bootstrapping in gsw-fhe. In *PKC '15*, pages 699–715, 2015.
28. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.
29. SEAL. Simple encrypted arithmetic library. https://sealcrypto.codeplex.com/.
30. M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *Eurocrypt*, pages 24–43, 2010.