Improved Programmable Bootstrapping with Larger Precision and Efficient Arithmetic Circuits for TFHE

Ilaria Chillotti[1], Damien Ligier[1], Jean-Baptiste Orfila[1], and Samuel Tap[1]

Zama, Paris, France - https://zama.ai/
{ilaria.chillotti,damien.ligier,jb.orfila,samuel.tap}@zama.ai

Abstract. *Fully Homomorphic Encryption* (FHE) schemes enable to compute over encrypted data. Among them, TFHE [8] has the great advantage of offering an efficient method for *bootstrapping noisy ciphertexts*, i.e., reduce the noise. Indeed, homomorphic computation increases the noise in ciphertexts and might compromise the encrypted message. TFHE bootstrapping, in addition to reducing the noise, also evaluates (for free) *univariate functions* expressed as look-up tables. It however requires to have the most significant bit of the plaintext to be known *a priori*, resulting in the loss of one bit of space to store messages. Furthermore it represents a non negligible overhead in terms of computation in many use cases.
In this paper, we propose a solution to overcome this limitation, that we call Programmable Bootstrapping Without Padding (**WoP-PBS**). This approach relies on two building blocks. The first one is the multiplication *à la* BFV [13] that we incorporate into TFHE. This is possible thanks to a thorough noise analysis showing that correct multiplications can be computed using practical TFHE parameters. The second building block is the generalization of TFHE bootstrapping introduced in this paper. It offers the flexibility to select any chunk of bits in an encrypted plaintext during a bootstrap. It also enables to evaluate many LUTs at the same time when working with small enough precision. All these improvements are particularly helpful in some applications such as the evaluation of Boolean circuits (where a bootstrap is no longer required in each evaluated gate) and, more generally, in the efficient evaluation of arithmetic circuits even with large integers. Those results improve TFHE circuit bootstrapping as well. Moreover, we show that bootstrapping large precision integers is now possible using much smaller parameters than those obtained by scaling TFHE ones.

Keywords: FHE · TFHE · Bootstrapping.

1 Introduction

Fully Homomorphic Encryption (FHE) is a family of encryption schemes allowing to perform computation over encrypted data. FHE schemes use noisy ciphertexts for security reasons, i.e., ciphertexts containing some randomness.

This noise grows after every performed homomorphic operation, and, if not controlled, can compromise the message and prevent the user from decrypting correctly. A technique called *bootstrapping* and introduced by Gentry [14] allows to reduce the noise, by mean of a public key called *bootstrapping key*. By using bootstrapping frequently, thus reducing the noise when needed, one can perform as many homomorphic operations as she wants, but it remains an expensive technique, both in terms of execution time and memory usage.

Nowadays, the most practical FHE schemes are based on the hardness assumption called *Learning With Errors* (LWE), introduced by Regev in 2005 [20], and on its *ring* variant (RLWE) [22,19]. Even if bootstrapping is possible for all these schemes, some of them (such as BGV [3], BFV [2,13] and CKKS [6]) actually avoid it because the technique remains a bottleneck. These schemes make use of RLWE ciphertexts exclusively and adopt a *leveled approach*, which consists in choosing parameters that are large enough to tolerate all the noise produced during the computation. These schemes take advantage of *SIMD encoding* [21] to pack many messages in a single ciphertext and perform the homomorphic evaluations in parallel on all of these messages at the same time, and they naturally perform homomorphic multiplications between RLWE ciphertexts by doing a *(tensor) product* followed by a *relinearization/key switching*.

TFHE [7,8,9] is also an (R)LWE-based FHE scheme which differentiates from the other (R)LWE-based cryptosystems because it supports a *very efficient bootstrapping* technique. TFHE was originally proposed as an improvement of *FHEW* [12], a GSW [15] based scheme with a fast bootstrapping for the evaluation of homomorphic Boolean gates. Apart from improving FHEW bootstrapping, TFHE also introduces new techniques in order to support more functionalities than the ones proposed by FHEW and to improve homomorphic evaluation of complex circuits. TFHE efficiency comes in part from the choice of a small ciphertext modulus which allows to use CPU native types to represent a ciphertext both in the standard domain and in Fourier domain. This is what we call the *TFHE context*.

TFHE encrypts messages in the most significant bits, meaning a message $m \in \mathbb{Z}$ is rescaled by a factor $\Delta \in \mathbb{Z}$ before being reduced modulo q. The small noise $e \in \mathbb{Z}$ is added in the least significant bit, so a noisy plaintext looks like $\Delta \cdot m + e \mod q$. In this paper, when we refer to *bits of precision*, we mean the quantity $p = \log_2(\frac{q}{\Delta})$. We illustrate this in Figure 1. Note that if $m > 2^p$ some of the information in m will be lost because of the modulo q.

TFHE *bootstrapping* is very efficient, but also *programmable*, meaning that a univariate function can be evaluated at the same time as the noise is being reduced. It is often called *programmable bootstrapping* [10,11] and noted PBS. The function to be evaluated is represented as a *look-up table* (LUT) and the bootstrapping rotates this table (stored in an encrypted polynomial) in order to output the correct element in the table. The LUT has to have redundancy (each coefficient is repeated a certain amount of time consecutively) in order to remove the input ciphertext noise during the PBS.

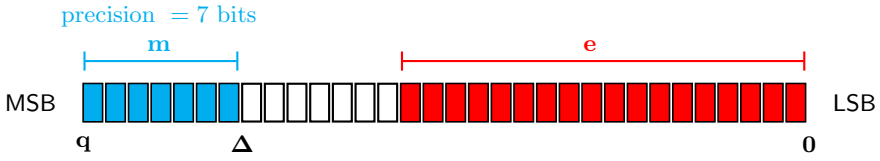

Fig. 1. In TFHE, messages are encoded in the most significant bits (MSB), and so it is rescaled by a scaling factor Δ, while the error appears in the least significant bits (LSB). The precision is $\log_2(\frac{q}{\Delta})$, i.e 7 bits in the figure.

A multi-output version of the PBS is described in [4] allowing the evaluation of multiple (negacyclic) functions $\{f_i\}_i$ over one encrypted input. Each function f_i is encoded as a LUT in a polynomial P_i. One can find a shared polynomial Q such that we can decompose each P_i as $Q \cdot P'_i$ and compute $\mathsf{CT}_{\mathsf{out}} \leftarrow \mathbf{PBS}(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, Q)$. Then, one needs to multiply $\mathsf{CT}_{\mathsf{out}}$ by each of P'_i and sample extract the resulting ciphertexts. One would have obtained the evaluation of each function. One drawback of this method is that the noise inside the i-th output ciphertexts depends on P'_i.

A recent paper revisits the TFHE bootstrapping [16]. It gives two algorithms and a few optimizations to compute programmable bootstrapping on large precision ciphertexts encrypting one message decomposed in a certain base. Those algorithms could be used to homomorphically compute multivariate functions if we call them with the right lookup tables.

The BGV/BFV/CKKS leveled approach is very convenient when the circuit that has to be homomorphically evaluated is small in terms of multiplicative depth, but also known in advance. When multiple inputs have to be evaluated with the same circuit at once, this approach is also very good in terms of amortized computation time. However, when the circuit is deep and unknown *a priori*, the TFHE approach is more convenient.

A recent work by Boura et al., called Chimera [1], tries to take advantage of both approaches, by building bridges between FHE schemes (TFHE, BFV and CKKS), in order to switch between them depending on which functionality is needed.

TFHE and its fast PBS are very powerful, but have some *limitations*:

A In general, to correctly bootstrap a ciphertext, its encrypted plaintext needs to have its *first Most Significant Bit (MSB) set to zero* (or at least known). The only exception is when the univariate function evaluated is negacyclic.
B One cannot bootstrap efficiently a message with a *large precision* (e.g., more than 6 bits). The number of bits of the message we bootstrap is strictly related to the dimension N of the ring chosen for the PBS. This means that the more we increase the precision, the more we have to increase the parameter N, and the *slower* the computation is.
C The PBS algorithm is *not multi-thread friendly*. Indeed, it is a loop working on an accumulator.

D There exists *no native multiplication* between two LWE ciphertexts. There are two approaches to multiply LWE ciphertexts: (i) use two programmable bootstrappings to evaluate the function $x \mapsto \frac{x^2}{4}$ so we can build the multiplication $x \cdot y = \frac{(x+y)^2}{4} - \frac{(x-y)^2}{4}$; (ii) use 1 or more TFHE circuit bootstrappings [8, Alg. 6] in order to convert one of the inputs into a GGSW (if not given as input) and then performing an external product. Since both techniques use PBS, they both suffer from limitations A and B.

E Because of limitations A and B it is not possible, in an efficient manner, to homomorphically *split a message* contained in a single ciphertext into several ciphertexts containing smaller chunks of the original message.

F The PBS can evaluate only a *single function per call*. Using the [4] trick, we can evaluate multiple Look-Up Tables at the same time, but the output will have an additional amount of noise which depends on the function evaluated.

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

H TFHE circuit bootstrapping requires ℓ PBS followed by many key switchings which is quite time consuming.

Contributions. In this paper we overcome the above-mentioned TFHE limitations. First, we *generalize TFHE PBS* so it can evaluate *several functions at once* without additional computation or noise. This approach is possible when the message to bootstrap is small enough. It overcomes limitation F and enables to compute a single generalized PBS when computing a circuit bootstrapping instead of ℓ PBS, overcoming limitation H. Circuit bootstrapping is particularly interesting in the leveled evaluation of Look-Up Tables, as shown in [8].

Furthermore, we thoroughly study the noise growth when computing a tensor product followed by a relinearization (i.e., the BFV-like multiplication) and found *parameters compatible with the TFHE context* representing a new way of computing LWE multiplications in TFHE. This multiplication is efficient and does not require a PBS which overcomes limitation D. We also propose a packed use of this algorithm to compute several LWE products at once or a sum of several LWE products at once. Our noise analysis is also valid for BFV-like schemes and can help estimate the noise growth there.

From this multiplication, we define a *new PBS procedure* that does not require the MSB to be set to zero, overcoming limitation A. This new procedure is composed of few generalized PBS that can be computed in parallel which makes it more multi-thread compatible (limitation C). Observe that, differently from Chimera, which builds bridges to move between different schemes, we add the support for a BFV-like multiplication into TFHE, in order to remove some of the TFHE limitations. In this way, we don't need to switch between schemes, and we can remain all the time in the TFHE context.

From this new PBS we are able to *homomorphically decompose* a plaintext from a single ciphertext into several ciphertexts encrypting blocks of the input

plaintext, overcoming limitation E, and also relax the need for PBS at every gate in the gate bootstrapping and its generalization, overcoming limitation G.

From this new decomposition algorithm and the Tree-PBS algorithm [16], we are able to create a *fast PBS for larger input messages*, overcoming limitation B. We can also in an even faster manner refresh the noise (bootstrap, not PBS) in a ciphertext from this new decomposition algorithm.

2 Background and Notations

The parameter q is a positive integer and represents the modulo for the integers we are working with. We note $\mathbb{Z}_q$ the ring $\mathbb{Z}/q\mathbb{Z}$. The parameter N is a power of 2 and represents the *size of polynomials* we are working with. We note $\mathfrak{R}_q$ the ring $\mathbb{Z}_q[X]/(X^N+1)$. A *Gaussian distribution* with a mean set to zero and a standard deviation set to σ is written χ_σ. We use the symbol $||$ for concatenation. When ι is an integer, we note by $[\cdot]_\iota$ the reduction modulo ι and by $\lfloor\cdot\rceil_\iota$ the rounding then the reduction modulo ι. We refer to the *most* (resp. *least*) *significant bits* of an integer as MSB (resp. LSB). We alse refer to *look-up tables* as LUT. The (computational) complexity of an algorithm Alg, potentially dependent on some parameters $p_1, \cdots, p_n$, is denoted $\mathbb{C}_{\mathsf{Alg}}^{p_1,\cdots,p_n}$.

Remark 1. Observe that in this paper we use different notations compared to TFHE [7,8,9]. In TFHE, the message and ciphertext spaces are expressed by using the real torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$. On a computer, they implemented $\mathbb{T}$ by using native arithmetic modulo 2^{32} or 2^{64}, which means that they work on $\mathbb{Z}_q$ (with $q = 2^{32}$ or $q = 2^{64}$). This is why we prefer to use $\mathbb{Z}_q$ instead of $\mathbb{T}$, as already adopted in [10]. It is made possible because there is an isomorphism between $\mathbb{Z}_q$ and $\frac{1}{q}\mathbb{Z}/\mathbb{Z}$ as explained in [1, Section 1].

LWE, RLWE & GLWE Ciphertexts. A GLWE ciphertext of a message $M \in \mathfrak{R}_q$ with the scaling factor $\Delta \in \mathbb{Z}_q$ under the secret key $\mathbf{S} \in \mathfrak{R}_q^k$ is defined as follows:

$$\mathsf{CT} = (A_1, \cdots, A_k, B = \sum_{i=1}^{k} A_i \cdot S_i + \lfloor M \cdot \Delta \rceil_q + E) = \mathsf{GLWE}_{\mathbf{S}}(M \cdot \Delta) \in \mathfrak{R}_q^{k+1}$$

such that $\mathbf{S} = (S_1, \cdots, S_k) \in \mathfrak{R}_q^k$ is the secret key with coefficients either sampled from a uniform binary, uniform ternary or Gaussian distribution, $\{A_i\}_{i=1}^k$ are polynomials in $\mathfrak{R}_q$ with coefficients sampled from the uniform distribution in $\mathbb{Z}_q$, E is an noise (error) polynomial in $\mathfrak{R}_q$ such that its coefficients are sampled from a Gaussian distributions χ_σ. The parameter k is a positive integer and represents the number of polynomials in the GLWE secret key. To simplify notations, we sometimes define S_{k+1} as -1.

A GLWE ciphertext with $N = 1$ is an *LWE ciphertext* and in this case we consider the parameter $n = k$ for the size of the LWE secret key and we note both the ciphertext and the secret with a lower case e.g. ct and $\mathbf{s}$. A GLWE ciphertext with $k = 1$ and $N > 1$ is an *RLWE ciphertext*.

Lev, RLev & GLev Ciphertexts. A GLev ciphertext with the base $\mathfrak{B} \in \mathbb{N}^*$ and $\ell \in \mathbb{N}^*$ levels, of a message $M \in \mathfrak{R}_q$ under the GLWE secret key $\mathbf{S} \in \mathfrak{R}_q^k$ is defined as the following vector of GLWE ciphertexts:

$$\overline{\mathsf{CT}} = (\mathsf{CT}_1, \cdots, \mathsf{CT}_\ell) = \mathsf{GLev}_{\mathbf{S}}^{\mathfrak{B},\ell}(M) \in \mathfrak{R}_q^{\ell \times (k+1)}$$

where $\mathsf{CT}_i = \mathsf{GLWE}_{\mathbf{S}}(M \cdot \frac{q}{\mathfrak{B}^i})$ is a GLWE ciphertext.

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

Decomposition Algorithms. The decomposition algorithm in the integer base $\mathfrak{B} \in \mathbb{N}^*$ with $\ell \in \mathbb{N}^*$ levels is written $\mathsf{dec}^{(\mathfrak{B},\ell)}$ and takes as input an integer $x \in \mathbb{Z}_q$ and output a decomposition vector of integers $(x_1, \cdots, x_\ell) \in \mathbb{Z}_q^\ell$ such that:

$$\left\langle \mathsf{dec}^{(\mathfrak{B},\ell)}(x), \left(\frac{q}{\mathfrak{B}^1}, \cdots, \frac{q}{\mathfrak{B}^\ell}\right) \right\rangle = \left\lfloor x \cdot \frac{\mathfrak{B}^\ell}{q} \right\rceil \cdot \frac{q}{\mathfrak{B}^\ell} \in \mathbb{Z}_q$$

Note that this decomposition starts from the MSB. When we apply this decomposition on a vector of integers, we end up with a vector of decomposition vector of integers.

We can also decompose an integer polynomials $X \in \mathfrak{R}_q$ into a decomposition vector of polynomials $(X_1, \cdots, X_\ell) \in \mathfrak{R}_q^\ell$ such that:

$$\left\langle \mathsf{dec}^{(\mathfrak{B},\ell)}(X), \left(\frac{q}{\mathfrak{B}^1}, \cdots, \frac{q}{\mathfrak{B}^\ell}\right) \right\rangle = \left\lfloor X \cdot \frac{\mathfrak{B}^\ell}{q} \right\rceil \cdot \frac{q}{\mathfrak{B}^\ell} \in \mathfrak{R}_q$$

When we apply this decomposition on a vector of polynomials, we end up with a vector of decomposition vectors of polynomials.

Key Switching. A technique that is often used in FHE, called *key switching*, allows to change parameters and keys in the ciphertext. The key switching makes the noise grow and is performed using a so-called *key-switching key* which is a public key composed of encryptions of secret key elements.
There are different types of key switchings: we will quickly list and describe the ones that are interesting for the understanding of the paper. The LWE-to-GLWE key-switching key is noted KSK and is equal to $\mathsf{KSK} = \left\{\overline{\mathsf{CT}}_i = \mathsf{GLev}_{\mathbf{S}'}^{\mathfrak{B},\ell}(s_i)\right\}_{1 \leq i \leq n}$, where $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{Z}_q^n$ is the input LWE secret key and $\mathbf{s}' = (S'_1, \ldots, S'_k) \in \mathfrak{R}_q^k$ is the output GLWE secret key.

- $\boxed{\mathsf{CT}_{\mathsf{out}} \leftarrow \mathbf{PrivateKS}(\{\mathsf{ct}_i\}_{i \in \{1,\ldots,p\}}, \mathsf{KSK})}$: allows to apply a private linear function $f : (\mathbb{Z}/q\mathbb{Z})^p \longrightarrow \mathbb{Z}/q\mathbb{Z}[X]$ over p LWE ciphertexts $\{\mathsf{ct}_i = \mathsf{LWE}_{\mathbf{s}}(m_1)\}_{i \in \{1,\ldots,p\}}$ and creates a GLWE ciphertext $\mathsf{CT}_{\mathsf{out}} = \mathsf{GLWE}_{\mathbf{S}'}(f(m_1, \cdots, m_p))$. For more details check [8, Algorithm 2].

- $\boxed{\mathsf{CT}_{\mathsf{out}} \leftarrow \mathbf{PublicKS}(\{\mathsf{ct}_i\}_{i\in\{1,\ldots,p\}}, \mathsf{KSK}, f)}$: is a public version of the previous key switching, i.e., a key switching with a public linear function f. For more details check [8, Algorithm 1]. The *key switching used in TFHE PBS* is a public key switching, where the function f is the identity function and the output GLWE is instantiated with $k = n'$ and $N = 1$ (i.e., as an LWE instance).
- $\boxed{\mathsf{CT}_{\mathsf{out}} \leftarrow \mathbf{PackingKS}(\{\mathsf{ct}_j\}_{j=1}^{p}, \{i_j\}_{j=1}^{p}, \mathsf{KSK})}$: is a (public) key switching procedure enabling to pack several LWE ciphertexts into one GLWE. It takes as input a set of p LWE ciphertexts as well as a set of p indexes. Given the set of indexes $\{i_j\}_{j=1}^{p}$, the function f has the following shape: $f(\{m_j\}_{j=1}^{p}) \longrightarrow \sum_{j=1}^{p} m_j \cdot X^{i_j}$.

GSW, RGSW & GGSW Ciphertexts. A GGSW ciphertext with the base $\mathfrak{B} \in \mathbb{N}^*$ and $\ell \in \mathbb{N}^*$ levels, of a message $M \in \mathfrak{R}_q$ under the GLWE secret key $\mathbf{s}=(S_1,\cdots,S_k)\in \mathfrak{R}_q^k$ is defined as the following vector of GLev ciphertexts:

$$\overline{\overline{\mathsf{CT}}} = \left(\overline{\mathsf{CT}}_1, \cdots, \overline{\mathsf{CT}}_{k+1}\right) = \mathsf{GGSW}_{\mathbf{S}}^{(\mathfrak{B},\ell)}(M) \in \mathfrak{R}_q^{(k+1)\times\ell\times(k+1)}$$

where $\overline{\mathsf{CT}}_i=\mathsf{GLev}_{\mathbf{S}}^{(\mathfrak{B},\ell)}(-S_i\cdot M)$ is a GLev ciphertext. Remember that we note $S_{k+1}=-1$.

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

TFHE PBS. The bootstrapping of TFHE has a double functionality: it reduces the noise in the ciphertexts and at the same time evaluates a univariate function. We call it PBS for *programmable bootstrapping*. In order to be performed, the PBS uses a so called *bootstrapping key*, i.e., a list of GGSW encryptions of the elements of the secret key used to encrypt the input LWE (noisy) ciphertext of the PBS. The procedure is composed of three major steps:

- *Modulus Switching*: the input LWE ciphertext in $\mathbb{Z}_q^{n+1}$ is converted into a ciphertext in $\mathbb{Z}_{2N}^{n+1}$;
- *Blind Rotation*: a GLWE encryption of a *redundant LUT*[1] is rotated (by using a loop of CMux operations [9]) according to the LWE ciphertext produced in the previous step and the public bootstrapping key;
- *Sample Extraction*: the constant coefficient of the GLWE output of the previous step is extracted as a LWE ciphertext.

[1] A *redundant LUT* is a LUT corresponding to a function f, whose entries are redundantly represented inside the coefficients of a polynomial in $\mathfrak{R}_q$. In practice, the redundancy consists in a r times (with r a system parameter) repetition of the entries $f(i)$ of the LUT with a certain shift: $P_f = X^{-r/2} \cdot \sum_{i=0}^{N/r-1} X^{i\cdot r} \cdot \left(\sum_{j=0}^{r-1} f(i) \cdot X^j\right)$. The redundancy is used to perform the rounding operation during bootstrapping.

TFHE Circuit Bootstrapping. In 2017, TFHE authors propose a technique called *citcuit bootstrapping* [8, Alg. 6], to convert an LWE ciphertext into a GGSW ciphertext, and to reduce its noise at the same time. The circuit bootstrapping is composed by a series of ℓ TFHE PBS, followed by a list of $(k+1)\ell$ private key switching procedures. The goal is to build one by one all the GLWE ciphertexts composing the output GGSW.

3 Building Blocks

In this section we describe two building blocks: the LWE multiplication, that uses an existing GLWE multiplication together with some key switchings and sample extraction, and a generalized version of TFHE PBS. Both techniques are necessaries in order to build our constructions in the rest of the paper.

3.1 LWE Multiplication

We first recall the multiplication algorithm for GLWE ciphertexts in Algorithm 1. It is composed of a *tensor product* followed by a *relinearization* and is widely used in the literature [13] (we recall the GLWE [3] algorithm, instead of the more limited RLWE version). Since this algorithm is largely used in the rest of the paper, we thoroughly study its noise growth and provide a formal noise analysis where $\mathsf{Var}(S)$ is the variance of a GLWE secret key polynomial $S \in \mathfrak{R}_q$, $\mathsf{Var}(S'_{\mathsf{even}})$ (resp. $\mathsf{Var}(S'_{\mathsf{odd}})$) is the variance of even (resp. odd) coefficients in S^2 and $\mathsf{Var}(S'')$ is the variance of coefficients in $S_i \cdot S_j$ which is the product between two independent secret key polynomials $S_i, S_j \in \mathfrak{R}_q$. We provide concrete cryptographic parameters depending on the precision and the multiplicative depth in the Table 3.1.

Precision	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Max. depth	32	16	16	8	8	8	8	4	4	4	4	4	4	2	2	2	2	2	2	2	2	2	2	2
$\log_2(N)$	12	11	12	11	11	12	12	11	11	11	12	12	12	11	11	11	11	11	11	11	12	12	12	12
$\log_2(\mathfrak{B})$	8	5	8	12	10	8	8	20	17	15	17	17	8	30	30	20	20	20	20	20	20	20	20	20
ℓ	8	10	8	4	5	8	8	2	3	3	3	3	8	1	1	2	2	2	2	2	2	2	2	2

Table 1. Parameters depending on the GLWE multiplicative depth and the precision.

Theorem 1 (GLWE multiplication). *Let* $\mathsf{CT}_1 = \mathsf{GLWE}_{\mathbf{S}}(\mathsf{PT}_1) \in \mathfrak{R}_q^{k+1}$ *and* $\mathsf{CT}_2 = \mathsf{GLWE}_{\mathbf{S}}(\mathsf{PT}_2) \in \mathfrak{R}_q^{k+1}$ *be two GLWE ciphertexts, encrypting respectively* $\mathsf{PT}_1 = M_1 \Delta_1 \in \mathfrak{R}_q$ *and* $\mathsf{PT}_2 = M_2 \Delta_2 \in \mathfrak{R}_q$*, under the same secret key* $\mathbf{S} = (S_1, \ldots, S_k) \in \mathfrak{R}_q^k$*, with noise sampled respectively from* χ_{σ_1} *and* χ_{σ_2}*. Let* $\mathsf{RLK} = \left\{ \overline{\mathsf{CT}}_{i,j} = \mathsf{GLev}_{\mathbf{S}}^{(\mathfrak{B},\ell)}(S_i \cdot S_j) \in \mathfrak{R}_q^{\ell \times (k+1)} \right\}_{1 \le i \le k}^{1 \le j \le i}$ *be a relinearization key for the GLWE secret key* $\mathbf{S}$*, with noise sampled from* $\chi_{\sigma_{\mathsf{RLK}}}$*.*

Algorithm 1 computes a new GLWE ciphertext CT *encrypting the product* $\mathsf{PT}_1 \cdot \mathsf{PT}_2 / \Delta \in \mathfrak{R}_q$ *where* $\Delta = min(\Delta_1, \Delta_2)$ *(a scaling factor), under the secret key* $\mathbf{S}$*, with*

a noise variance $\mathsf{Var}_{\mathbf{GLWEMult}}$ *estimated by the following formula:*

$$\begin{aligned}\mathsf{Var}_{\mathbf{GLWEMult}} = &\tfrac{N}{\Delta^2}\left(\Delta_1^2||M_1||_\infty^2\sigma_2^2+\Delta_2^2||M_2||_\infty^2\sigma_1^2+\sigma_1^2\sigma_2^2\right)+\\
&+\tfrac{N}{\Delta^2}\left(\tfrac{q^2-1}{12}\left(1+kN\mathsf{Var}(S)+kN\mathbb{E}^2(S)\right)+\tfrac{kN}{4}\mathsf{Var}(S)+\tfrac{1}{4}(1+kN\mathbb{E}(S))^2\right)(\sigma_1^2+\sigma_2^2)+\\
&+\tfrac{1}{12}+\tfrac{kN}{12\Delta^2}\cdot\left((\Delta^2-1)\cdot\left(\mathsf{Var}(S)+\mathbb{E}^2(S)\right)+3\cdot\mathsf{Var}(S)\right)+\tfrac{k(k-1)N}{24\Delta^2}\cdot\left((\Delta^2-1)\cdot\left(\mathsf{Var}(S'')+\mathbb{E}^2(S'')\right)+3\cdot\mathsf{Var}(S'')\right)+\\
&+\tfrac{kN}{24\Delta^2}\cdot\left((\Delta^2-1)\cdot\left(\mathsf{Var}(S'_{\mathsf{odd}})+\mathsf{Var}(S'_{\mathsf{even}})+2\cdot\mathbb{E}^2(S'_{\mathsf{mean}})\right)+3\cdot\left(\mathsf{Var}(S'_{\mathsf{odd}})+\mathsf{Var}(S'_{\mathsf{even}})\right)\right)+k\ell N\sigma_{\mathsf{RLK}}^2\cdot\tfrac{(k+1)}{2}\cdot\tfrac{\mathfrak{B}^2+2}{12}+\\
&+\tfrac{kN}{2}\left(\tfrac{q^2}{12\mathfrak{B}^{2\ell}}-\tfrac{1}{12}\right)\left((k-1)\cdot(\mathsf{Var}(S'')+\mathbb{E}^2(S''_{\mathsf{mean}}))+\mathsf{Var}(S'_{\mathsf{odd}})+\mathsf{Var}(S'_{\mathsf{even}})+2\mathbb{E}^2(S'_{\mathsf{mean}})\right)+\\
&+\tfrac{kN}{8}\cdot\left((k-1)\cdot\mathsf{Var}(S'')+\mathsf{Var}(S'_{\mathsf{odd}})+\mathsf{Var}(S'_{\mathsf{even}})\right).\end{aligned} \tag{1}$$

Let $k^*=\frac{k(k+1)}{2}$ *and* $k^+=\frac{(k+1)(k+2)}{2}$*. The complexity of the algorithm is:*

$$\begin{aligned}\mathbb{C}^{(k,\ell,n,N)}_{\mathsf{GLWEMult}} &= \mathbb{C}^{(k,N)}_{\mathsf{TensorProduct}}+\mathbb{C}^{(k,\ell,N)}_{\mathsf{Relin}}, \textit{with}\\
\mathbb{C}^{(k,N)}_{\mathsf{TensorProduct}} &= 2(k+1)\mathbb{C}_{\mathsf{FFT}}+k^+\mathbb{C}_{\mathsf{iFFT}}+(k+1)^2N\mathbb{C}_{\mathsf{multFFT}}+k^*N\mathbb{C}_{\mathsf{addFFT}}, \textit{and}\\
\mathbb{C}^{(k,\ell,N)}_{\mathsf{Relin}} &= N\ell k^*\mathbb{C}_{\mathsf{dec}}+k^*\ell\mathbb{C}_{\mathsf{FFT}}+k^*\ell(k+1)N\mathbb{C}_{\mathsf{multFFT}}+(k^*\ell-1)(k+1)N\mathbb{C}_{\mathsf{addFFT}}+(k+1)\mathbb{C}_{\mathsf{iFFT}}\end{aligned} \tag{2}$$

Proof (sketch). In the proof, we compute the decryption of the resulting ciphertext, obtaining the message plus the noise so we can estimate its variance. The detailed computation leading us to the aforementioned noise formula is provided in the full version of the paper. □

The same Algorithm 1 can be adapted in order to perform a GLWE square: the square is more efficient since $R'_{i,j}$ and A'_i are computed with a single multiplication instead of two. For more details, we refer to the full version of the paper.

3.1.1 Single LWE Multiplication We now define Algorithm 2 for homomorphically *multiply two LWE ciphertexts*. It requires the *sample extraction* procedure, which is an algorithm adding no noise to the ciphertext and consisting in simply rearranging some of the coefficients of the GLWE input ciphertext to build the output LWE ciphertext encrypting one of the coefficients of the input polynomial plaintext. The sample extraction is described in [9, Section 4.2] for RLWE inputs, and can be easily extended to GLWE ones. Due to page constraint, this algorithm is described in the the full version of the paper.

Theorem 2 (LWE-to-GLWE Packing Key Switch). *We start with the simplest case were we pack a single LWE ciphertext. Let* $\mathsf{ct}_{\mathsf{in}}=\mathsf{LWE}_{\mathbf{s}}(m\cdot\Delta)\in\mathbb{Z}_q^{n+1}$ *be an LWE ciphertext encrypting* $m\cdot\Delta\in\mathbb{Z}_q$*, under the LWE secret key* $\mathbf{s}=(s_1,\ldots,s_n)\in\mathbb{Z}_q^n$*, with noise sampled respectively from* χ_σ*. Let* $\mathbf{S}'$ *be a GLWE secret key such that* $\mathbf{S}'=(S'_1,\ldots,S'_k)\in\mathfrak{R}_q^{k+1}$*. Let* $\mathsf{KSK}=\left\{\overline{\mathsf{CT}}_i=\mathsf{GLev}^{\mathfrak{B},\ell}_{\mathbf{S}'}(s_i)\in\mathfrak{R}_q^{\ell\times(k+1)}\right\}_{1\le i\le n}$ *be a key switching key from* $\mathbf{s}$ *to* $\mathbf{S}'$ *with noise sampled from* $\chi_{\sigma_{\mathsf{KSK}}}$*.*

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

$$\begin{aligned}\mathsf{Var}^{(1)}_{\mathsf{fill}} &= \sigma^2+n\cdot\left(\frac{q^2}{12\mathfrak{B}^{2\ell}}-\frac{1}{12}\right)\cdot\left(\mathsf{Var}(s_i)+\mathbb{E}^2(s_i)\right)+\frac{n}{4}\cdot\mathsf{Var}(s_i)+n\cdot\ell\cdot\sigma^2_{\mathsf{KSK}}\cdot\frac{\mathfrak{B}^2+2}{12}\\
\mathsf{Var}^{(1)}_{\mathsf{emp}} &= n\cdot\ell\cdot\sigma^2_{\mathsf{KSK}}\cdot\frac{\mathfrak{B}^2+2}{12}\end{aligned} \tag{3}$$

Algorithm 1: $\mathsf{CT} \leftarrow \mathbf{GLWEMult}\left(\mathsf{CT}_1, \mathsf{CT}_2, \mathsf{RLK}\right)$

Context: $\begin{cases} \mathbf{S} = (S_1, \dots, S_k) \in \mathfrak{R}_q^k : \text{ a GLWE secret key} \\ \Delta = \min\left(\Delta_1, \Delta_2\right) \in \mathbb{Z}_q \\ \mathsf{PT}_1 = M_1 \Delta_1 \in \mathfrak{R}_q \\ \mathsf{PT}_2 = M_2 \Delta_2 \in \mathfrak{R}_q \end{cases}$

Input: $\begin{cases} \mathsf{CT}_1 = \mathsf{GLWE}_{\mathbf{S}}\left(\mathsf{PT}_1\right) = (A_{1,1}, \cdots, A_{1,k}, B_1) \in \mathfrak{R}_q^{k+1} \\ \mathsf{CT}_2 = \mathsf{GLWE}_{\mathbf{S}}\left(\mathsf{PT}_2\right) = (A_{2,1}, \cdots, A_{2,k}, B_2) \in \mathfrak{R}_q^{k+1} \\ \mathsf{RLK} = \left\{\overline{\mathsf{CT}}_{i,j} = \mathsf{GLev}_{\mathbf{S}}^{(\mathfrak{B},\ell)}\left(S_i \cdot S_j\right)\right\}_{1 \le i \le k}^{1 \le j \le i} : \text{ a relinearization key for } \mathbf{S} \end{cases}$

Output: $\mathsf{CT} = \mathsf{GLWE}_{\mathbf{S}}\left(\frac{\mathsf{PT}_1 \cdot \mathsf{PT}_2}{\Delta}\right) \in \mathfrak{R}_q^{k+1}$

begin

/* Tensor product */

for $1 \le i \le k$ **do**

$T_i' \leftarrow \left\lfloor \left\lfloor \frac{[A_{1,i} \cdot A_{2,i}]_Q}{\Delta} \right\rceil \right\rceil_q$

end

for $1 \le i \le k,\ 1 \le j < i$ **do**

$R_{i,j}' \leftarrow \left\lfloor \left\lfloor \frac{[A_{1,i} \cdot A_{2,j} + A_{1,j} \cdot A_{2,i}]_Q}{\Delta} \right\rceil \right\rceil_q$

end

for $1 \le i \le k$ **do**

$A_i' \leftarrow \left\lfloor \left\lfloor \frac{[A_{1,i} \cdot B_2 + B_1 \cdot A_{2,i}]_Q}{\Delta} \right\rceil \right\rceil_q$

end

$B' \leftarrow \left\lfloor \left\lfloor \frac{[B_1 \cdot B_2]_Q}{\Delta} \right\rceil \right\rceil_q$

/* Relinearization */

$\mathsf{CT} \leftarrow$

$(A_1', \cdots, A_k', B') + \sum_{i=1}^{k} \left\langle \overline{\mathsf{CT}}_{i,i}, \mathsf{dec}^{(\mathfrak{B},\ell)}\left(T_i'\right) \right\rangle + \sum_{1 \le i \le k}^{1 \le j < i} \left\langle \overline{\mathsf{CT}}_{i,j} \cdot \mathsf{dec}^{(\mathfrak{B},\ell)}\left(R_{i,j}'\right) \right\rangle$

end

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

$$\mathbb{C}_{\mathsf{PackingKS}}^{(\alpha,\ell,n,k,N)} = \alpha\ell n \mathbb{C}_{\mathsf{dec}} + \alpha\ell n(k+1)N\mathbb{C}_{\mathsf{mul}} + ((\alpha\ell n - 1)(k+1)N + \alpha)\mathbb{C}_{\mathsf{add}}$$

Proof (sketch). In the proof, we compute the decryption of the resulting ciphertext, obtaining the message plus the noise so we can estimate the two variances. The detailed computation leading us to the aforementioned noise formulas are provided in the full version of the paper. □

Theorem 3 (LWE Multiplication). *Let $\mathsf{ct}^{(1)} = \mathsf{LWE}_{\mathbf{s}}(m_1 \cdot \Delta_1)$ and $\mathsf{ct}^{(2)} = \mathsf{LWE}_{\mathbf{s}}(m_2 \cdot \Delta_2)$ be two LWE ciphertexts, encrypting respectively $m_1 \cdot \Delta_1$ and $m_2 \cdot \Delta_2$, both encrypted under the LWE secret key $\mathbf{s} = (s_1, \dots, s_n)$, with noise sampled respectively from χ_{σ_1} and χ_{σ_2}. Let $\mathsf{KSK} = \left\{\overline{\mathsf{CT}}_i = \mathsf{GLev}_{\mathbf{S}'}^{\mathfrak{B},\ell}(s_i)\right\}_{1 \le i \le n}$ a key switching key from $\mathbf{s}$ to $\mathbf{S}'$ where $\mathbf{S}' = (S_1', \dots, S_k')$, with noise sampled from $\chi_{\sigma_{\mathsf{KSK}}}$. Let RLK be a relinearization key for $\mathbf{S}'$, defined as in Theorem 1.*

Algorithm 2 computes a new LWE ciphertext $\mathsf{ct}_{\mathsf{out}}$, encrypting the product $m_1 \cdot m_2 \cdot \Delta_{\mathsf{out}}$, where $\Delta_{\mathsf{out}} = max(\Delta_1, \Delta_2)$, under the secret key $\mathbf{s}'$. The variance of the

Algorithm 2: $\mathsf{ct}_{\mathsf{out}} \leftarrow$ **LWEMult** $(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{RLK}, \mathsf{KSK})$

Context:
$$\begin{cases} \mathbf{s} = (s_1, \cdots, s_n) \in \mathbb{Z}_q^n : \text{ the LWE input secret key} \\ \mathbf{s}' = (s'_1, \cdots, s'_{kN}) \in \mathbb{Z}_q^{kN} : \text{ the LWE output secret key} \\ \mathbf{S}' = (S'_1, \ldots, S'_k) \in \mathfrak{R}_q^k : \text{ a GLWE secret key} \\ \forall 1 \leq i \leq k,\ S'_i = \sum_{j=0}^{N-1} s'_{(i-1)\cdot N+j+1} X^j \in \mathfrak{R}_q \\ \Delta_{\mathsf{out}} = \max(\Delta_1, \Delta_2) \in \mathbb{Z}_q \end{cases}$$

Input:
$$\begin{cases} \mathsf{ct}_1 = \mathsf{LWE}_{\mathbf{s}}(m_1 \cdot \Delta_1) \in \mathbb{Z}_q^{n+1} \\ \mathsf{ct}_2 = \mathsf{LWE}_{\mathbf{s}}(m_2 \cdot \Delta_2) \in \mathbb{Z}_q^{n+1} \\ \mathsf{RLK} : \text{ a relinearization key for } \mathbf{S}' \text{ as defined in algorithm 1} \\ \mathsf{KSK} = \left\{ \overline{\mathsf{CT}}_i = \mathsf{GLev}_{\mathbf{S}'}^{\mathfrak{B},\ell}(s_i) \right\}_{1 \leq i \leq n} : \text{ a key switching key from } \mathbf{s} \text{ to } \mathbf{S}' \end{cases}$$

Output: $\mathsf{ct}_{\mathsf{out}} = \mathsf{LWE}_{\mathbf{s}'}(m_1 \cdot m_2 \cdot \Delta_{\mathsf{out}}) \in \mathbb{Z}_q^{kN+1}$

1 **begin**
  `/* KS from LWE to GLWE */`
2 $\mathsf{CT}_1 = \mathsf{GLWE}_{\mathbf{S}'}(m_1 \cdot \Delta_1) \leftarrow$ **PackingKS**$(\{\mathsf{ct}_1\}, \{0\}, \mathsf{KSK})$;
3 $\mathsf{CT}_2 = \mathsf{GLWE}_{\mathbf{S}'}(m_2 \cdot \Delta_2) \leftarrow$ **PackingKS**$(\{\mathsf{ct}_2\}, \{0\}, \mathsf{KSK})$;
  `/* GLWE multiplication: Tensor product + Relinearization */`
4 $\mathsf{CT} = \mathsf{GLWE}_{\mathbf{S}'}(m_1 \cdot m_2 \cdot \Delta_{\mathsf{out}}) \leftarrow$ **GLWEMult**$(\mathsf{CT}_1, \mathsf{CT}_2, \mathsf{RLK})$
  `/* Sample extract the constant term */`
5 $\mathsf{ct}_{\mathsf{out}} = \mathsf{LWE}_{\mathbf{s}'}(m_1 \cdot m_2 \cdot \Delta_{\mathsf{out}}) \leftarrow$ **SampleExtract** $(\mathsf{CT}, 0)$
6 **end**

noise in $\mathsf{ct}_{\mathsf{out}}$ *can be estimated by replacing the variances* σ_1 *and* σ_2 *in the RLWE multiplication (Formula 1, Theorem 1) with the variance estimated after a packing key switch (Formula 3, Theorem 2). The complexity is:*

$$\mathbb{C}_{\mathsf{LWEMult}}^{(\ell_{\mathbf{KS}},\ell_{\mathbf{RL}},n,k,N)} = 2 \cdot \mathbb{C}_{\mathsf{PackingKS}}^{(1,\ell_{\mathbf{KS}},n,k,N)} + \mathbb{C}_{\mathsf{GLWEMult}}^{(k,\ell_{\mathbf{RL}},n,N)} + \mathbb{C}_{\mathsf{SampleExtract}}^{(N)}$$

3.1.2 Packed Products & Packed Sum of Products It is possible to use algorithm 2 to compute with a single multiplication several products, or several squares, or a sum of several products, or even a sum of several squares.

These four functionalities can be easily achieved by slightly modifying Algorithm 2. In the case of **PackedMult** and **PackedSumProducts**, the algorithm take in input two sets of LWE ciphertexts $\left\{\mathsf{ct}_i^{(1)}\right\} = \left\{\mathsf{LWE}_{\mathbf{s}}(m_i^{(1)} \cdot \Delta_1)\right\}_{0 \leq i < \alpha}$ and $\left\{\mathsf{ct}_i^{(2)}\right\} = \left\{\mathsf{LWE}_{\mathbf{s}}(m_i^{(2)} \cdot \Delta_2)\right\}_{0 \leq i < \alpha}$:

1. **PackedMult:** the goal is to compute LWE encryptions of the products $m_i^{(1)} \cdot m_i^{(2)} \cdot \Delta_{\mathsf{out}}$, where $\Delta_{\mathsf{out}} = max(\Delta_1, \Delta_2)$. The two input sets are packed with a packing key switch into two GLWE ciphertexts with indexes $\mathfrak{L}_1 = \{0,1,2,\cdots,\alpha-1\}$ and $\mathfrak{L}_2 = \{0,\alpha,2\alpha,\cdots,(\alpha-1)\alpha\}$ respectively. The resulting GLWE ciphertexts are multiplied with the GLWE multiplication (Algorithm 1) and finally all the coefficients at indexes $i \cdot (\alpha + 1)$ (for $0 \leq i < \alpha$) are extracted.
2. **PackedSumProducts:** the goal is to compute a LWE encryption of the sum of products $\sum_{i=0}^{\alpha-1} m_i^{(1)} \cdot m_i^{(2)} \cdot \Delta_{\mathsf{out}}$, where $\Delta_{\mathsf{out}} = max(\Delta_1, \Delta_2)$. The two input sets are packed with a packing key switch into two GLWE ciphertexts with indexes $\mathfrak{L}_1 = \{0,1,2,\cdots,\alpha-1\}$ and $\mathfrak{L}_2 = \{\alpha-1,\alpha-2,\alpha-3,\cdots,0\}$ respectively. The resulting GLWE ciphertexts are multiplied with the GLWE multiplication (Algorithm 1) and finally the coefficient at index $\alpha - 1$ is extracted.

Note that it is possible to compute packed squares and a packed sum of squares if the two LWE input sets are equal. It is also possible to compute squares and a sum of squares by computing a RLWE multiplication between an RLWE ciphertext and itself. In that case, a single set of LWE input in provided $\{\mathsf{ct}_i\} = \{\mathsf{LWE}_{\mathbf{s}}(m_i \cdot \Delta)\}_{0 \leq i < \alpha}$:

1. **PackedSquares:** the goal is to compute LWE encryptions of the squares $m_i^2 \cdot \Delta$. The input set is packed with a packing key switch into a GLWE ciphertext with indexes $\mathfrak{L}=\{2^0-1, 2^1-1, 2^2-1, \cdots, 2^{\alpha-1}-1\}$. The resulting GLWE ciphertext is squared by using the GLWE square algorithm and finally all the coefficients at indexes $2^{i+1}-2$ (for $0 \leq i < \alpha$) are extracted.
2. **PackedSumSquares:** the goal is to compute a LWE encryption of the sum of squares $\sum_{i=0}^{\alpha-1} m_i^2 \cdot 2\Delta$. To achieve this goal, the input set is packed with a packing key switch into a GLWE ciphertext with redundancy, using two indexes sets $\mathfrak{L}_1=\{0,1,2,\cdots,\alpha-1\}$ and $\mathfrak{L}_2=\{2\alpha-1, 2\alpha-2, 2\alpha-3, \cdots, \alpha\}$. The resulting GLWE ciphertext is squared by using the GLWE square algorithm and finally the coefficient at index $2\alpha-1$ is extracted.

Note that we could also compute packed products and a packed sum of products with a GLWE square algorithm by changing $\mathfrak{L}$, $\mathfrak{L}_1$ and $\mathfrak{L}_2$ and also extracting different coefficients. Also note that for these four algorithms, there are restrictions regarding the maximum value that α can take each time. We provide more details in the the full version of the paper.

3.2 Generalized PBS

We propose a *more versatile* algorithm for the PBS where we are able to bootstrap a precise chunk of bits, instead of only the MSB as described in TFHE, and to also apply several function evaluations at once. We describe this generalization in Algorithm 3. We introduce two *new parameters*, $\varkappa$ and ϑ, which redefine the *modulus switching* step of TFHE PBS. In particular, $\varkappa$ defines the number of MSB that are not considered in the PBS, while 2^ϑ defines the number of functions that can be evaluated at the same time in a single generalized PBS.

The two parameters $\varkappa$ and ϑ are illustrated in Figure 2, where "input" represents the plaintext (with noise) that is encrypted the input ciphertext of the modulus switching, and "output" illustrates the plaintext (with noise) that is encrypted inside the output ciphertext (after modulus switching). The first $\varkappa$ MSB will not impact the following steps of the generalized PBS and ϑ bits will be set to 0 in order to encode 2^ϑ functions in the LUT stored in P_f (see Section 4.3 for more details). Observe that the case $(\varkappa, \vartheta) = (0, 0)$ corresponds to the original TFHE PBS.

We also define the *"plaintext modulus switching"* function written **PTModSwitch** to recover the plaintext of the encrypted output of a modulus switching algorithm. Let $m \in \mathbb{Z}_q$ be a message, $\Delta \in \mathbb{Z}_q$ its scaling factor, $\varkappa \in \mathbb{Z}$ and $\vartheta \in \mathbb{N}$ the parameters of a modulus switching. We define $q' = \frac{q}{\Delta 2^\varkappa}$. The case where $\varkappa \geq 0$ is illustrated in Figure 3. We defined $(\beta, m') \leftarrow \mathbf{PTModSwitch}_q(m, \Delta, \varkappa, \vartheta) \in \{0,1\} \times \mathbb{N}$ as follow:

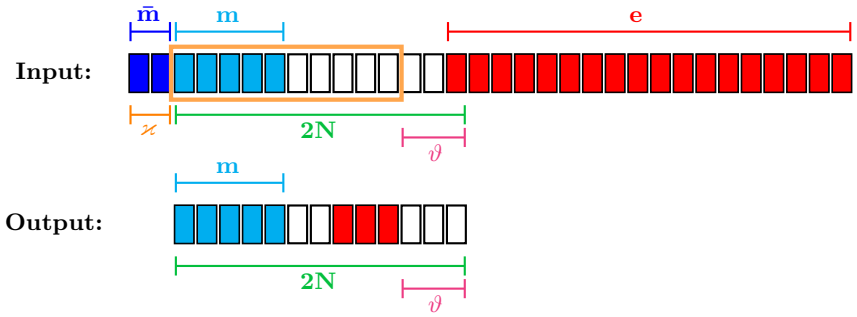

Fig. 2. Modulus switching operation in the generalized PBS (Algorithm 3): on top of the figures we illustrate the data $(\bar{m}, m, e)$, on the bottom the dimensions $(\varkappa, 2N, \vartheta)$.

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

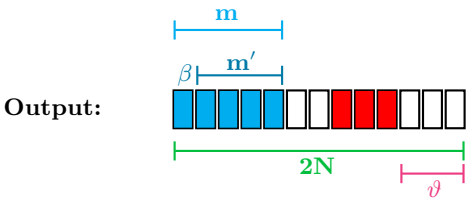

Fig. 3. Plaintext after the modulus switching from the generalized PBS (Algorithm 3) where $\varkappa \geq 0$: on top of the figure we illustrate the data(m, β, m'), on the bottom the dimensions $(2N, \vartheta)$.

Note that for simplicity purpose, we provide the generalized PBS noise formula *only for binary secret keys*. However, in the full version of the paper we provide formulas as well as proofs for more key distributions (binary, ternary and Gaussian).

Theorem 4 (Generalized PBS). *Let* $\mathbf{s}=(s_1,\cdots,s_n)\in\mathbb{Z}_q^n$ *be a binary LWE secret key. Let* $\mathbf{S}'=(S'_1,\ldots,S'_k)\in\mathfrak{R}_q^k$ *be a GLWE binary secret key such that* $S'_i=\sum_{j=0}^{N-1} s'_{(i-1)\cdot N+j+1}\cdot X^j$*, and* $\mathbf{s}' = (s'_1, \cdots, s'_{kN})$ *be the corresponding binary LWE secret key. Let* P_f *be a* r*-redundant LUT for a function* $f:\mathbb{Z}\to\mathbb{Z}$ *and* Δ_{out} *be the output scaling factor. Let* $(\varkappa, \vartheta)$ *be the two integer variables defining (along with* N*) the window size to be modulus switched, such that* $\frac{q2^{\vartheta}}{\Delta_{\mathsf{in}}2^{\varkappa}}<2N$*, and let* $(\beta,m')=\mathbf{PTModSwitch}_q(m,\Delta_{\mathsf{in}},\varkappa,\vartheta)\in\{0,1\}\times\mathbb{N}$.

Then Algorithm 3 takes as input a LWE ciphertext $\mathsf{ct}_{\mathsf{in}}=\mathsf{LWE}_{\mathbf{s}}(m\cdot\Delta_{\mathsf{in}})\in\mathbb{Z}_q^{n+1}$ *with noise distribution from* $\chi_{\sigma_{\mathsf{in}}}$*, a bootstrapping key* $\mathsf{BSK}=\left\{\overline{\overline{\mathsf{CT}}}_i=\mathsf{GGSW}_{\mathbf{S}'}^{\mathfrak{B},\ell}(s_i)\right\}_{i=1}^{n}$ *from* $\mathbf{s}$ *to* $\mathbf{S}'$ *and a (possibly trivial) GLWE encryption of* $P_f\cdot\Delta_{\mathsf{out}}$*, and returns an LWE ciphertext* $\mathsf{ct}_{\mathsf{out}}$ *under the secret key* $\mathbf{s}'$*, encrypting the message* $(-1)^{\beta}\cdot f(m')\cdot\Delta_{\mathsf{out}}$ *if and only if the input noise has variance* $\sigma_{\mathsf{in}}^2<\frac{\Delta_{\mathsf{in}}^2}{4\Gamma^2}-\frac{q'^2}{12w^2}+\frac{1}{12}-\frac{nq'^2}{24w^2}-\frac{n}{48}$*, where* Γ *is a variable depending on the probability of correctness defined as* $P=\mathsf{erf}\left(\frac{\Gamma}{\sqrt{2}}\right)$*,* $w=2N\cdot 2^{-\vartheta}$ *and* $q'=q\cdot 2^{-\varkappa}$*.*

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

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

The complexity of Algorithm 3 is the same as the complexity of TFHE bootstrapping [9], i.e.,

$$\mathbb{C}_{\mathsf{GenPBS}}^{(n,\ell,k,N)}=\mathbb{C}_{\mathsf{ModulusSwitching}}^{(n)}+n\mathbb{C}_{\mathsf{CMUX}}^{(n,\ell,k,N)}\mathbb{C}_{\mathsf{SampleExtract}}^{(N)}\quad\textit{with}$$

$$\begin{cases}\mathbb{C}_{\mathsf{ModulusSwitching}}^{(n)} & =(n+1)\mathbb{C}_{\mathsf{Scale\&Round}}\\ \mathbb{C}_{\mathsf{CMUX}}^{(n,\ell,k,N)} & =(k+1)(n+1)\mathbb{C}_{\mathsf{Rotation}}^{(N)}+2n(k+1)N\mathbb{C}_{\mathsf{Add}}+\mathbb{C}_{\mathsf{ExternalProduct}}^{(n,\ell,k,N)}\\ \mathbb{C}_{\mathsf{ExternalProduct}}^{(n,\ell,k,N)} & =n\ell(k+1)N\mathbb{C}_{\mathsf{dec}}+n\ell(k+1)\mathbb{C}_{\mathsf{FFT}}+n(k+1)\ell(k+1)N\mathbb{C}_{\mathsf{multFFT}}+\\ & \quad+n(k+1)(\ell(k+1)-1)N\mathbb{C}_{\mathsf{addFFT}}+n(k+1)\mathbb{C}_{\mathsf{iFFT}}\end{cases}$$

Proof (sketch). In the proof, we compute the decryption of the resulting ciphertext, obtaining the message plus the noise so we can estimate its variance. The detailed proof of this theorem is provided in the full version of the paper. □

4 Upgraded Bootstrapping

This section describes our main contributions, i.e., the **WoP-PBS** (PBS without a bit of padding) and the PBS evaluating multiple look-up tables at the same time (we call this algorithm **PBSmanyLUT**).

4.1 WoP-PBS first version

A big constraint with TFHE PBS is the *negacyclicity of the rotation of the LUT*. It implies a need of a *padding bit* (as mentioned in Limitation A). We propose a solution to remove that requirement, by using the aforementioned LWE multiplication (Algorithm 1) and the generalized PBS (Algorithm 3). This new bootstrapping is called the *programmable bootstrapping without padding* (**WoP-PBS**) and a first version is described in Algorithm 4.

Theorem 5 (PBS Without Padding (V1)). *Let* $\mathbf{s}=(s_1,\cdots,s_n)\in\mathbb{Z}_q^n$ *be a binary LWE secret key. Let* $\mathbf{S}'=(S'_1,\ldots,S'_k)\in\mathfrak{R}_q^k$ *be a GLWE secret key such that* $S'_i=\sum_{j=0}^{N-1}s'_{(i-1)\cdot N+j+1}X^j\in\mathfrak{R}_q$*, and* $\mathbf{s}'=(s'_1,\cdots,s'_{kN})\in\mathbb{Z}_q^{kN}$ *be the corresponding binary LWE key. Let* $P_f\in\mathfrak{R}_q$ *(resp.* $P_1\in\mathfrak{R}_q$*) be a r-redundant LUT for the function*

Algorithm 3: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathbf{GenPBS}\left(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \mathsf{CT}_f, \varkappa, \vartheta\right)$

Context:
$\begin{cases} \mathbf{s} = (s_1, \cdots, s_n) \in \mathbb{Z}_q^n : \text{ the LWE input secret key} \\ \mathbf{s}' = (s'_1, \cdots, s'_{kN}) \in \mathbb{Z}_q^{kN} : \text{ the LWE output secret key} \\ \mathbf{S}' = (S'_1, \ldots, S'_k) \in \mathfrak{R}_q^k : \text{ a GLWE secret key} \\ \forall 1 \leq i \leq k,\ S'_i = \sum_{j=0}^{N-1} s'_{(i-1)\cdot N+j+1} X^j \in \mathfrak{R}_q \\ P_f \in \mathfrak{R}_q : \text{a } r\text{-redundant LUT for } x \mapsto f(x) \\ \Delta_{\mathsf{out}} \in \mathbb{Z}_q : \text{ the output scaling factor} \\ f : \mathbb{Z} \to \mathbb{Z} : \text{a function} \\ (\beta, m') = \mathbf{PTModSwitch}_q(m, \Delta_{\mathsf{in}}, \varkappa, \vartheta) \in \{0,1\} \times \mathbb{N} \end{cases}$

Input:
$\begin{cases} \mathsf{ct}_{\mathsf{in}} = \mathsf{LWE}_{\mathbf{s}}(m \cdot \Delta_{\mathsf{in}}) = (a_1, \cdots, a_n, a_{n+1} = b) \in \mathbb{Z}_q^{n+1} \\ \mathsf{BSK} = \left\{\overline{\overline{\mathsf{CT}}}_i = \mathsf{GGSW}_{\mathbf{S}'}^{\mathfrak{B},\ell}(s_i)\right\}_{i=1}^{n} : \text{ a bootstrapping key from } \mathbf{s} \text{ to } \mathbf{S}' \\ \mathsf{CT}_f = \mathsf{GLWE}_{\mathbf{S}'}(P_f \cdot \Delta_{\mathsf{out}}) \in \mathfrak{R}_q^{k+1} \\ (\varkappa, \vartheta) \in \mathbb{Z} \times \mathbb{N} : \text{define along with } N \text{ the chunk of the plaintext to bootstrap} \end{cases}$

Output: $\mathsf{ct}_{\mathsf{out}} = \mathsf{LWE}_{\mathbf{s}'}\left((-1)^{\beta} \cdot f\left(m'\right) \cdot \Delta_{\mathsf{out}}\right)$ if we respect requirements in Theorem 4

begin
  /* modulus switching */
  for $1 \leq i \leq n+1$ **do**
    $a'_i \leftarrow \left\lfloor \left\lfloor \frac{a_i \cdot 2N \cdot 2^{\varkappa - \vartheta}}{q} \right\rceil \cdot 2^{\vartheta} \right\rfloor_{2N}$
  end
  /* blind rotate of the LUT */
  $\mathsf{CT} \leftarrow \mathbf{BlindRotate}\left(\mathsf{CT}_f, \{a'_i\}_{i=1}^{n+1}, \mathsf{BSK}\right)$;
  /* sample extract the constant term */
  $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathbf{SampleExtract}\left(\mathsf{CT}, 0\right)$
end

$f : \mathbb{Z} \mapsto \mathbb{Z}$, (resp. the constant function $x \mapsto 1$) and $\Delta_{\mathsf{out}} \in \mathbb{Z}_q$ be the output scaling factor. Let CT_f be a (possibly trivial) GLWE encryption of $P_f \cdot \Delta_{\mathsf{out}}$ and $\mathsf{CT}_{\mathbb{1}}$ be a trivial GLWE encryption of $P_{\mathbb{1}} \cdot \Delta_{\mathsf{out}}$. Let $(\varkappa, \vartheta) \in \mathbb{Z} \times \mathbb{N}$ be the two integer variables defining (along with N) the chunk of the plaintext that is going to be bootstrapped, such that $\frac{q 2^{\vartheta}}{\Delta_{\mathsf{in}} 2^{\varkappa}} < 2N$, and let $(\beta, m') = \mathbf{PTModSwitch}_q(m, \Delta_{\mathsf{in}}, \varkappa, \vartheta) \in \{0,1\} \times \mathbb{N}$.

Let $\mathsf{KSK} = \left\{\overline{\mathsf{CT}}_i = \mathsf{GLev}_{\mathbf{S}'}^{(\mathfrak{B},\ell)}\left(s'_i\right)\right\}_{1 \leq i \leq n}$ be a key switching key from $\mathbf{s}'$ to $\mathbf{S}'$, with noise sampled respectively from $\chi_{\sigma^{(1)}}$ and $\chi_{\sigma^{(2)}}$. Let $\mathsf{RLK} = \left\{\overline{\mathsf{CT}}_{i,j} = \mathsf{GLev}_{\mathbf{S}'}^{(\mathfrak{B},\ell)}\left(S'_i \cdot S'_j\right)\right\}_{1 \leq i \leq k}^{1 \leq j \leq i}$ be a relinearization key for $\mathbf{S}'$, defined as in Theorem 1. Let $\mathsf{BSK} = \left\{\overline{\overline{\mathsf{CT}}}_i = \mathsf{GGSW}_{\mathbf{S}'}^{\mathfrak{B},\ell}(s_i)\right\}_{i=1}^{n}$ be a bootstrapping key from $\mathbf{s}$ to $\mathbf{S}'$.

Then the Algorithm 4 takes in input a LWE ciphertext $\mathsf{ct}_{\mathsf{in}} = \mathsf{LWE}_{\mathbf{s}}(m \cdot \Delta_{\mathsf{in}}) \in \mathbb{Z}_q^{n+1}$ where $\mathsf{ct}_{\mathsf{in}} = (a_1, \cdots, a_n, a_{n+1} = b)$, with noise sampled from $\chi_{\sigma_{\mathsf{in}}}$, and returns an LWE ciphertext $\mathsf{ct}_{\mathsf{out}} \in \mathbb{Z}_q^{kN+1}$ under the secret key $\mathbf{s}'$ encrypting the messages $f\left(m'\right) \cdot \Delta_{\mathsf{out}}$ if and only if the input noise has variance verifying Theorem 3.

The output ciphertext noise variance verifies $\mathsf{Var}(\mathbf{WoP\text{-}PBS}_1) = \mathsf{Var}(\mathbf{LWEMult})$ with input variances for the LWE multiplication (Algorithm 2) defined as $\sigma_i = \mathsf{Var}(\mathbf{GenPBS})$, for $i \in \{1, 2\}$.

The complexity of Algorithm 4 is:

$$\mathbb{C}_{\mathbf{WoP\text{-}PBS}_1}^{(n,\ell_{\mathbf{PBS}},k_1,N_1,\ell_{\mathbf{KS}},\ell_{\mathbf{RL}},k_2,N_2)} = 2\mathbb{C}_{\mathsf{GenPBS}}^{(n,\ell_{\mathbf{PBS}},k_1,N_1)} + \mathbb{C}_{\mathsf{LWEMult}}^{(\ell_{\mathbf{KS}},\ell_{\mathbf{RL}},N_1,k_2,N_2)}$$

Proof (Sketch). We only provide a proof of correctness of the algorithm, considering that the noise and the complexity are directly deduced from the **GenPBS**

Algorithm 4: $\mathsf{ct}_{\mathsf{out}} \leftarrow$ **WoP-PBS**$_1(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \mathsf{RLK}, \mathsf{KSK}, P_f, \Delta_{\mathsf{out}}, \varkappa, \vartheta)$

Context:
- $\mathbf{s} = (s_1, \cdots, s_n) \in \mathbb{Z}_q^n$
- $\mathbf{s}' = (s'_1, \cdots, s'_{kN}) \in \mathbb{Z}_q^{kN}$
- $\mathbf{S}' = \left(S'^{(1)}, \ldots, S'^{(k)}\right) \in \mathfrak{R}_q^k$
- $\forall 1 \leq i \leq k,\ S'^{(i)} = \sum_{j=0}^{N-1} s'_{(i-1)\cdot N+j+1} X^j \in \mathfrak{R}_q$
- $f : \mathbb{Z} \to \mathbb{Z}$: a function
- $P_1 \in \mathfrak{R}_q$: a redundant LUT for $x \mapsto 1$
- $(\beta, m') = \mathbf{PTModSwitch}_q(m, \Delta, \varkappa, \vartheta) \in \{0,1\} \times \mathbb{N}$
- $\mathsf{CT}_f = \mathsf{GLWE}_{\mathbf{S}'}(P_f \cdot \Delta_{\mathsf{out}}) \in \mathfrak{R}_q^{k+1}$ (might be a trivial encryption)
- $\mathsf{CT}_1 \in \mathfrak{R}_q^{k+1}$: a trivial encryption of $P_1 \cdot \Delta_{\mathsf{out}}$

Input:
- $\mathsf{ct}_{\mathsf{in}} = \mathsf{LWE}_{\mathbf{s}}(m \cdot \Delta_{\mathsf{in}}) = (a_1, \cdots, a_n, a_{n+1} = b) \in \mathbb{Z}_q^{n+1}$
- $\mathsf{BSK} = \left\{\mathsf{BSK}_i = \mathsf{GGSW}_{\mathbf{S}'}^{(\mathfrak{B},\ell)}(s_i)\right\}_{1 \leq i \leq n}$: a bootstrapping key from $\mathbf{s}$ to $\mathbf{S}'$
- $\mathsf{RLK} = \left\{\overline{\mathsf{CT}}_{i,j} = \mathsf{GLev}_{\mathbf{S}'}^{(\mathfrak{B},\ell)}\left(S'_i \cdot S'_j\right)\right\}_{1 \leq i \leq k}^{1 \leq j \leq i}$: a relinearization key for $\mathbf{S}'$
- $\mathsf{KSK} = \left\{\overline{\mathsf{CT}}_i = \mathsf{GLev}_{\mathbf{S}'}^{(\mathfrak{B},\ell)}(s'_i)\right\}_{1 \leq i \leq kN}$: a key switching key from $\mathbf{s}'$ to $\mathbf{S}'$
- $P_f \in \mathfrak{R}$: a redundant LUT for $x \mapsto f(x)$
- $\Delta_{\mathsf{out}} \in \mathbb{Z}_q$: the output scaling factor
- $(\varkappa, \vartheta) \in \mathbb{Z} \times \mathbb{N}$: define along with N the window size

Output: $\mathsf{ct}_{\mathsf{out}} = \mathsf{LWE}_{\mathbf{s}'}(f(m') \cdot \Delta_{\mathsf{out}})$ if we respect requirements in Theorem 5

```
1 begin
    /* Compute two PBS in parallel:                                   */
2   ct_f = LWE_{s'}((-1)^β · f(m') · Δ_out) ← GenPBS(ct_in, BSK, CT_f, ϰ − 1, ϑ) ;
3   ct_Sign = LWE_{s'}((-1)^β · Δ_out) ← GenPBS(ct_in, BSK, CT_1, ϰ − 1, ϑ) ;
    /* Compute the multiplication                                     */
4   ct_out ← LWEMult(ct_f, ct_Sign, RLK, KSK);
5 end
```

and **LWEMult** algorithms. Both of the **GenPBS** are applied with the same parameters except for the evaluated function (P_{f} or $P_{\mathbb{1}}$). Thus, in both ciphertexts ct_{f} and $\mathsf{ct}_{\mathrm{Sign}}$ the value of β is the same. Then, $\mathsf{ct}_{\mathsf{out}} = \mathsf{LWE}_{\mathbf{s}}((-1)^{2\beta} \cdot f(m') \cdot \Delta_{\mathsf{out}}) = \mathsf{LWE}_{\mathbf{s}}(f(m') \cdot \Delta_{\mathsf{out}})$. □

Remark 2. Observe that, in Algorithm 4 we set KSK as a key switching key for $\mathbf{s}'$ to $\mathbf{S}'$ where $\mathbf{s}'$ is the LWE secret key composed of the coefficients in $\mathbf{S}'$. In practice, the key switching can be done to a key $\mathbf{S}''$, that has nothing to do with $\mathbf{s}'$. In this case, the RLK should be adapted as well to the key $\mathbf{S}''$.

It shall be noticed that in Algorithm 4:

- The two **GenPBS** have the same input ciphertext. To make the evaluation more efficient (evaluating a single bootstrapping instead of two), it is possible to use either the multi-value bootstrap described in [4], which will be faster but at the cost of a higher output noise. Another option would be to take advantage of the **PBSmanyLUT**, that we describe in detail in Algorithm 6 if the input message is small enough (*cf.* Remark 3).
- There could be only one key switching done in **LWEMult** (instead of two) if one of the two inputs is provided as a GLWE ciphertext (one **GenPBS** does not perform the final sample extraction).
- The **LWEMult** on line 4 can be replaced be a **MultSquareLWE** which is faster.

These improvements could impact both increase the noise but improve the complexity of the algorithm.

4.2 WoP-PBS second version

Another big constraint with TFHE PBS is that *the polynomial size is directly linked to the size of the message we want to bootstrap* (as mentioned in Limitation B). The smallest growth of the polynomial size slows down the computation by more than a factor 2 as TFHE PBS complexity is proportional to the FFT complexity: $N \log_2(N)$ with N the polynomial size. Keeping that in mind, we offer a different way to perform a bootstrap without padding in Algorithm 5 which can be more efficient in a multi-threaded machine. The main idea behind this Algorithm is to write a message m as $\beta||m'$ with β the most significant bit and m' the rest of the message. The function f to be computed is broken into two functions: f_0 and f_1. We want f_0 if β is equal to 0 and f_1 if $\beta = 1$. We use β as an encrypted decision bit, so we can choose between $f_0(m')$ or $f_1(m')$ thanks to the **LWEMult** algorithm.

We give the complete set of cryptographic parameters for different precisions in the full version of the paper. In a nutshell, for precisions from 1 to 5 bits, we use $\log_2(N) = 11$ and for 6 and 7 bits of precisions, we use $\log_2(N) = 12$.

Theorem 6 (PBS Without Padding (V2)). *Let f_0 and f_1 be the two functions representing f such that $f_0(x) = f(x) = f_1(x-p)$ for a certain $p \in \mathbb{N}$. Then, under the same hypothesis of Theorem 5, the Algorithm 5 takes in input a LWE ciphertext $\mathsf{ct}_{\mathsf{in}}=\mathsf{LWE}_{\mathbf{s}}(m\cdot\Delta_{\mathsf{in}})=(a_1,\cdots,a_n,a_{n+1}=b)$, with noise from $\chi_{\sigma_{\mathsf{in}}}$, and returns in output a LWE ciphertext $\mathsf{ct}_{\mathsf{out}}$ under the secret key $\mathbf{s}'$ encrypting the messages $f(m')\cdot\Delta_{\mathsf{out}}$ if and only if the input noise has variance verifying the Theorem 3.*

The output ciphertext noise variance verifies $\mathsf{Var}(\mathbf{WoP\text{-}PBS}_2)=2\cdot\mathsf{Var}(\mathbf{LWEMult})$ with input variances for the **LWEMult** *defined as $\sigma_i=\mathsf{Var}(\mathbf{GenPBS})$, for $i \in \{1,2\}$.*

The complexity of Algorithm 4 is:

$$\mathbb{C}^{(n,\ell_{\mathbf{PBS}},k_1,N_1,\ell_{\mathbf{KS}},\ell_{\mathbf{RL}},k_2,N_2)}_{\mathbf{WoP\text{-}PBS}_2} = 3\mathbb{C}^{(n,\ell_{\mathbf{PBS}},k_1,N_1)}_{\mathsf{GenPBS}} + 2\mathbb{C}^{(\ell_{\mathbf{KS}},\ell_{\mathbf{RL}},N_1,k_2,N_2)}_{\mathsf{LWEMult}} + (N_2+3)\mathbb{C}_{\mathsf{add}}$$

Proof (Sketch). We have $\mathsf{ct}_{\beta_0}=\mathsf{LWE}_{\mathbf{s}'}(\frac{\Delta_{\mathsf{out}}}{2}((-1)^{\beta}+1))$. If $\beta = 0$, then $\mathsf{ct}_{\beta_0}=\mathsf{LWE}_{\mathbf{s}'}(\Delta_{\mathsf{out}})$ else $\mathsf{ct}_{\beta_0}=\mathsf{LWE}_{\mathbf{s}'}(0)$. Then, $\mathsf{ct}_{\beta_0}=\mathsf{LWE}_{\mathbf{s}'}((1-\beta)\Delta_{\mathsf{out}})$. Similarly, we obtain $\mathsf{ct}_{\beta_1}=\mathsf{LWE}_{\mathbf{s}'}((-\beta)\Delta_{\mathsf{out}})$. The output ciphertext $\mathsf{ct}_{\mathsf{out}}$ is then equal to $\mathsf{LWE}_{\mathbf{s}'}(((-1)^{\beta}(1-\beta)\Delta_{\mathsf{out}}f_0(m')+(-1)^{\beta}(-\beta)\Delta_{\mathsf{out}}f_1(m')$. Thus, if $\beta = 0$, $\mathsf{ct}_{\mathsf{out}}=f_0(m')$ else $\mathsf{ct}_{\mathsf{out}}=f_1(m')$, as expected. □

It shall be noticed that in Algorithm 5:

- The three **GenPBS** have the same input ciphertext. As we observed for Algorithm 4, to make the evaluation more efficient by evaluating a single bootstrapping instead of three, it is possible to use either the multi-value bootstrap described in [4] or to take advantage of the **PBSmanyLUT** (Algorithm 6 and *cf.* Remark 3).
- We could remove two key switches (among four) as explained for the **WoP-PBS**$_1$.

Algorithm 5: $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathbf{WoP\text{-}PBS}_2(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \mathsf{RLK}, \mathsf{KSK}, P_f, \Delta_{\mathsf{out}}, \varkappa, \vartheta)$

Context:
- $\mathbf{s} = (s_1, \cdots, s_n) \in \mathbb{Z}_q^n$
- $\mathbf{s}' = (s'_1, \cdots, s'_{kN}) \in \mathbb{Z}_q^{kN}$
- $\mathbf{S}' = \left(S'^{(1)}, \ldots, S'^{(k)}\right) \in \mathfrak{R}_q^k$
- $\forall 1 \leq i \leq k,\ S'^{(i)} = \sum_{j=0}^{N-1} s'_{(i-1)\cdot N+j+1} X^j \in \mathfrak{R}_q$
- $f_0(x) = f(x) = f_1(x-p)$ for a certain p
- $(\beta, m') = \mathbf{PTModSwitch}_q(m, \Delta, \varkappa, \vartheta) \in \{0,1\} \times \mathbb{N}$
- $P_1 \in \mathfrak{R}_q$: as defined in Algorithm 4
- $\mathsf{CT}_{f_i} = \mathsf{GLWE}_{\mathbf{S}'}\left(P_{f_i} \cdot \Delta_{\mathsf{out}}\right) \in \mathfrak{R}_q^{k+1}$ (might be a trivial encryption)
- $\mathsf{CT}_1 \in \mathfrak{R}_q^{k+1}$: a trivial encryption of $P_1 \cdot \frac{\Delta_{\mathsf{out}}}{2}$
- $P_{f_0}, P_{f_1} \in \mathfrak{R}_q$: redundant LUTs of the two halves of P_f

Input:
- $\mathsf{ct}_{\mathsf{in}} = \mathsf{LWE}_{\mathbf{s}}(m \cdot \Delta_{\mathsf{in}}) = (a_1, \cdots, a_n, a_{n+1} = b) \in \mathbb{Z}_q^{n+1}$
- $\mathsf{BSK}, \mathsf{KSK}, \mathsf{RLK}$: as defined in Algorithm 4
- $P_f \in \mathfrak{R}_q$: a redundant LUT for $x \mapsto f(x)$
- $\Delta_{\mathsf{out}} \in \mathbb{Z}_q$: the output scaling factor
- $(\varkappa, \vartheta) \in \mathbb{Z} \times \mathbb{N}$: define along with N the window size

Output: $\mathsf{ct}_{\mathsf{out}} = \mathsf{LWE}_{\mathbf{s}'}(f(m') \cdot \Delta_{\mathsf{out}})$ if we respect requirements in Theorem 6

begin

`/* Compute in parallel 3 PBS: */`

$\mathsf{ct}_{\mathsf{f}_0} = \mathsf{LWE}_{\mathbf{s}'}((-1)^\beta \cdot \Delta_{\mathsf{out}} \cdot f_0(m')) \leftarrow \mathbf{GenPBS}(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \mathsf{CT}_{f_0}, \varkappa, \vartheta)$;

$\mathsf{ct}_{\mathsf{f}_1} = \mathsf{LWE}_{\mathbf{s}'}((-1)^\beta \cdot \Delta_{\mathsf{out}} \cdot f_1(m')) \leftarrow \mathbf{GenPBS}(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \mathsf{CT}_{f_1}, \varkappa, \vartheta)$;

$\mathsf{ct}_{\mathrm{Sign}} = \mathsf{LWE}_{\mathbf{s}'}((-1)^\beta \cdot \frac{\Delta_{\mathsf{out}}}{2}) \leftarrow \mathbf{GenPBS}(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \mathsf{CT}_1, \varkappa, \vartheta)$;

`/* Compute two sums in parallel: */`

$\mathsf{ct}_{\beta_0} = \mathsf{LWE}_{\mathbf{s}'}((1-\beta) \cdot \Delta_{\mathsf{out}}) \leftarrow \mathsf{ct}_{\mathrm{Sign}} + (\mathbf{0}, \frac{\Delta_{\mathsf{out}}}{2})$;

$\mathsf{ct}_{\beta_1} = \mathsf{LWE}_{\mathbf{s}'}(-\beta \cdot \Delta_{\mathsf{out}}) \leftarrow \mathsf{ct}_{\mathrm{Sign}} - (\mathbf{0}, \frac{\Delta_{\mathsf{out}}}{2})$;

`/* Compute two multiplications in parallel: */`

$\mathsf{ct}_{\beta \cdot \mathsf{f}_0} \leftarrow \mathbf{LWEMult}(\mathsf{ct}_{\mathsf{f}_0}, \mathsf{ct}_{\beta_0}, \mathsf{RLK}, \mathsf{KSK})$;

$\mathsf{ct}_{\beta \cdot \mathsf{f}_1} \leftarrow \mathbf{LWEMult}(\mathsf{ct}_{\mathsf{f}_1}, \mathsf{ct}_{\beta_1}, \mathsf{RLK}, \mathsf{KSK})$;

`/* Add the previous results: */`

$\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{ct}_{\beta \cdot \mathsf{f}_0} + \mathsf{ct}_{\beta \cdot \mathsf{f}_1}$;

end

- To improve both performance and noise, in practice, we can do a lazy relinearization as described in [18], i.e., the step of relinearization of the two **LWEMult** will be done after the final addition.
- The two **LWEMult** followed by the final addition can be replaced by a **PackedSumProducts** (described in the full version of the paper).

These improvements could increase the noise but also improve the complexity of the algorithm.

4.3 A multi-output PBS

We are able to extract any chunk of the encrypted plaintext with ϑ, $\varkappa$ and N. When possible, one can define a smaller chunk for the plaintext by trimming the bound in the LSB using a $\vartheta > 0$. It means that after the modulus switching there are ϑ LSB set to 0. More formally, after the modulus switching, a plaintext m^* will be of the form $m^* = m \cdot \Delta + e \cdot 2^\vartheta \in \mathbb{Z}_q$.

Thank to the ϑ LSB set to 0 in the plaintext, one can evaluate 2^ϑ functions at the cost of only one **GenPBS** without increasing the noise compared to a regular TFHE PBS. The procedure is described in Algorithm 6.

Algorithm 6: $\mathsf{ct}_1, \ldots, \mathsf{ct}_{2^\vartheta} \leftarrow$ **PBSmanyLUT**$(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, P_{(f_1,\ldots f_{2^\vartheta})}, \Delta_{\mathsf{out}}, \varkappa, \vartheta)$

Context:

$$\begin{cases} \mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{Z}_q^n \\ \mathbf{s}' = (s'_1, \ldots, s'_{kN}) \in \mathbb{Z}_q^{kN} \\ \mathbf{S}' = \left(S'^{(1)}, \ldots, S'^{(k)}\right) \in \mathfrak{R}_q^k \\ \forall 1 \le i \le k,\ S'^{(i)} = \sum_{j=0}^{N-1} s'_{(i-1)\cdot N+j+1} X^j \in \mathfrak{R}_q \\ f_1, \ldots, f_{2^\vartheta} : \mathbb{Z} \to \mathbb{Z} \\ (\beta, m') = \mathbf{PTModSwitch}_q(m, \Delta, \varkappa, \vartheta) \in \{0,1\} \times \mathbb{N} \\ \mathsf{CT}_{(f_1,\ldots,f_{2^\vartheta})} = \mathsf{GLWE}_{\mathbf{S}'}\left(P_{(f_1,\ldots,f_{2^\vartheta})} \cdot \Delta_{\mathsf{out}}\right) \text{ (might be a trivial encryption)} \end{cases}$$

Input:

$$\begin{cases} \mathsf{ct}_{\mathsf{in}} = \mathsf{LWE}_{\mathbf{s}}(m \cdot \Delta_{\mathsf{in}}) = (a_1, \cdots, a_n, a_{n+1} = b) \in \mathbb{Z}_q^{n+1} \\ \mathsf{BSK} = \left\{\mathsf{BSK}_i = \mathsf{GGSW}_{\mathbf{S}'}^{(\mathfrak{B},\ell)}(s_i)\right\}_{1 \le i \le n} \\ P_{(f_1,\ldots,f_{2^\vartheta})} : \text{a redundant LUT for} : x \mapsto f_1(x)||\ldots||f_{2^\vartheta}(x) \\ (\varkappa, \vartheta) \in \mathbb{Z} \times \mathbb{N} : \text{define along with N the window size} \end{cases}$$

Output: $\mathsf{ct}_1, \ldots, \mathsf{ct}_{2^\vartheta}$ such that $\mathsf{ct}_j = \mathsf{LWE}_{\mathbf{s}'}\left((-1)^\beta \cdot f_j(m') \cdot \Delta_{\mathsf{out}}\right)$

begin
  /* modulus switching */
  for $1 \le i \le n+1$ **do**
    $a'_i \leftarrow \left\lfloor \left\lfloor \frac{a_i \cdot 2N \cdot 2^{\varkappa-\vartheta}}{q} \right\rceil \cdot 2^\vartheta \right\rfloor_{2N}$
  end
  /* blind rotate of the LUT */
  $\mathsf{CT} \leftarrow$ **BlindRotate** $\left(\mathsf{CT}_{(f_1,\cdots,f_{2^\vartheta})}, \{a'_i\}_{1 \le i \le n+1}, \mathsf{BSK}\right)$;
  /* sample extract the first 2^ϑ terms (coeffs. from 0 to $2^\vartheta - 1$) */
  for $1 \le j \le 2^\vartheta$ **do**
    $\mathsf{ct}_j \leftarrow$ **SampleExtract**$_{j-1}$ (CT)
  end
end

The form of the LUT polynomial is set accordingly to the ϑ parameter so that it contains up to 2^ϑ functions. As for TFHE bootstrapping, one needs to have redundancy in the LUT to remove the input noise. Each block of functions (i.e., the sequence of $f_i, i \in [1, 2^\vartheta]$ coefficients) is repeated all along the polynomial. The LUT can be build as follow:

$$P_{(f_1,\ldots,f_{2^\vartheta})} = X^{\frac{N}{2p}} \sum_{j=0}^{p-1} X^{j\frac{N}{p}} \sum_{k=0}^{\frac{N}{p2^\vartheta}-1} X^{k\cdot 2^\vartheta} \sum_{i=0}^{2^\vartheta-1} f_{i+1}(j) X^i, \text{ with } p = \frac{q}{\Delta_{\mathsf{in}} \cdot 2^{\varkappa+1}}$$

By doing so, one can sample extract at the end 2^ϑ coefficients which leads to 2^ϑ output ciphertexts, one for each evaluated functions. By neglecting the computational cost of the ϑ sample extractions, the complexity is the same than for a PBS evaluating only one function. The noise is also not impacted.

This method is particularly efficient when the polynomial size is constrained by the desired output noise. If the polynomial size is chosen large enough, there

will be bits set to zero between the modulus switching noise and the message. This new method allows to exploit these bits to compute different functions on the same input ciphertext.

Theorem 7 (Multi-output PBS). *Let* $\mathbf{s} = (s_1, \cdots, s_n) \in \mathbb{Z}_q^n$ *be a binary LWE secret key. Let* $\mathbf{S}'=(S'_1,\ldots,S'_k)\in\mathfrak{R}_q^k$ *be a GLWE secret key such that* $S'_i=\sum_{j=0}^{N-1} s'_{(i-1)\cdot N+j+1}X^j\in\mathfrak{R}_q$*, and* $\mathbf{s}'=(s'_1,\cdots,s'_{kN})\in\mathbb{Z}_q^{kN}$ *be the corresponding LWE key. Let* $P_{(f_1,\cdots f_{2^\vartheta})}\in\mathfrak{R}_q$ *be a* r*-redundant LUT for the functions* $x\mapsto f_1(x)||\ldots||f_{2^\vartheta}(x)$ *and* $\Delta_{\mathsf{out}} \in \mathbb{Z}_q$ *be the output scaling factor. Let* $(\varkappa, \vartheta) \in \mathbb{Z} \times \mathbb{N}$ *be the two integer variables defining (along with* N*) the window size to be modulus switched, such that* $\frac{q2^\vartheta}{\Delta_{\mathsf{in}}2^\varkappa}<2N$*, and let* $(\beta,m')=\mathbf{PTModSwitch}_q(m,\Delta_{\mathsf{in}},\varkappa,\vartheta)$.

Then, the Algorithm 6 takes in input a LWE ciphertext $\mathsf{ct}_{\mathsf{in}}=\mathsf{LWE}_{\mathbf{s}}(m\cdot\Delta_{\mathsf{in}})=(a_1,\cdots,a_n,a_{n+1}=b)$*, with noise distribution from* $\chi_{\sigma_{\mathsf{in}}}$*, a bootstrapping key* $\mathsf{BSK}=\left\{\overline{\mathsf{CT}}_i=\mathsf{GGSW}^{\mathfrak{B},\ell}_{\mathbf{S}'}(s_i)\right\}_{i=1}^{n}$ *from* $\mathbf{s}$ *to* $\mathbf{S}'$ *and a (trivial) GLWE encryption of* $P_f\cdot\Delta_{\mathsf{out}}$*, and returns in output* 2^ϑ *LWE ciphertexts* $\{\mathsf{ct}_j\}_{j\in[0,2^\vartheta]}$ *under the secret key* $\mathbf{s}'$ *encrypting the messages* $(-1)^\beta\cdot f_j(m')\cdot\Delta_{\mathsf{out}}$ *if and only if the input noise has variance verifying the Theorem 3.*

The complexity of the algorithm is:

$$\mathbb{C}^{(n,\ell,k,N,\vartheta)}_{\mathbf{PBSmanyLUT}} = \mathbb{C}^{(n,\ell,k,N)}_{\mathsf{GenPBS}} + 2^\vartheta\mathbb{C}^{(N)}_{\mathsf{SampleExtract}}$$

Proof. The proof is the mainly the same as the one from the **GenPBS** (provided in the full version of the paper). Let $p = \frac{q}{\Delta_{\mathsf{in}}\cdot 2^{\varkappa+1}}$ be the number of possible values for each $f_i, i \in [0, 2^\vartheta]$. Let $m \in [0, p-1]$ be a plaintext value. The polynomial $P_{(f_0,\cdots,f_{2^\vartheta})}$ encodes the following LUT:

$$\underbrace{\left(\ldots,\underbrace{f_1(m),\ldots,f_{2^\vartheta}(m)}_{N/p\text{ elements}},\ldots,f_1(m),\ldots,f_{2^\vartheta}(m),\underbrace{f_1(m+1),\ldots,f_{2^\vartheta}(m+1)}_{N/p\text{ elements}},\ldots,f_1(m+1),\ldots,f_{2^\vartheta}(m+1),\ldots\right)}_{p\text{ blocks}}$$

From the **GenPBS**, ϑ bits are set to 0. Then, by construction of the LUT, $\mathsf{LUT}_{(f_0,\cdots,f_{2^\vartheta})}[m^*+i]=f_{i+1}(m')$ for $i \in [0, 2^\vartheta - 1]$, so that sample extracting gives the expected result. □

Remark 3. Observe that **PBSmanyLUT** and **WoP-PBS** algorithms can be combined in two different ways:

1. Using **PBSmanyLUT** to improve **WoP-PBS**: In **WoP-PBS**$_1$, the ct_{Sign} and each ct_{f_i} resulting from distinct **GenPBS** can be evaluated at once by using a single **PBSmanyLUT**. Similarly, in **WoP-PBS**$_2$, ct_{Sign} and each $\mathsf{ct}_{f_{0,i}}$ and $\mathsf{ct}_{f_{1,i}}$ could be evaluated at once. In both cases, this variant can be applied only if the polynomial size chosen for the **WoP-PBS** is large enough to allow multiple LUT evaluations (i.e, if precision is not yet a bottleneck condition): this variant of the **WoP-PBS** will improve the complexity of the algorithm, without impacting the noise growth.
2. Using **WoP-PBS** to improve **PBSmanyLUT**: The **PBSmanyLUT** algorithm implicitly performs a **GenPBS** with a special modulus switching.

This **GenPBS** can actually be replaced by a **WoP-PBS** (with the same special modulus switching) as a **WoP-PBS** performs the same operation as **GenPBS**, without the bit of padding constraint. This technique is what we call **WoPBSmanyLUT**.

Remark 4. A technique to evaluate many LUTs at the same time by performing a single TFHE bootstrapping (plus a bunch of polynomial multiplications per LUT) has been already proposed in [4] and used in [16]. Their technique does not impose a strong constraint on the polynomial size used for the bootstrapping, however it results in a larger output noise, that strictly depends on the function that is evaluated. If the noise constraints at the output of the bootstrapping are a problem, the technique of [4] will require to increase the polynomial size.

Our new **PBSmanyLUT** is a better alternative to this technique in some situations as the output noise will be independent of the function evaluated. But this comes at the cost of having enough space for the evaluation of the different LUTs (i.e., ϑ bits on the modulus switching to evaluate 2^ϑ functions so a large enough polynomial size N must be chosen). If we already are working with large enough polynomials, there is no computation overhead nor noise growth when replacing a **GenPBS** by a **PBSmanyLUT**.

5 Applications

In this section we present some of the applications that take advantage from our new techniques. In particular, we show that:

- Using a combination of **LWEMult** and **GenPBS** improves the gate bootstrapping technique of TFHE [9], because it allows to perform leveled binary operations between bootstrappings (instead of bootstrapping every single gate).
- The improved gate bootstrapping technique can be extended in order to evaluate arithmetic circuits with larger precision, by using a combination of **LWEMult** and **WoP-PBS** (or its variants).
- Using the **PBSmanyLUT** technique allows to improve the Circuit Bootstrapping of TFHE by a factor ℓ, without affecting the noise growth.
- The **WoP-PBS** technique (and its variants) can be used to bootstrap on larger precision inputs.

5.1 Fast Arithmetic

We start by describing an improvement of FHE Boolean circuit evaluation. Then, we extend it to arithmetic circuits dealing with integers encoded in more than a single bit. Finally, we describe how to use the later to build exact computation on bigger encrypted integers.

5.1.1 Fast Boolean Arithmetic In TFHE [7], authors improve techniques proposed in FHEW [12] to perform fast homomorphic evaluation of Boolean circuits and called this feature *gate bootstrapping*. It is very easy to use, because it performs one bootstrapping for each bivariate Boolean gate evaluated: there is no need to be careful with the noise management anymore because each gate reset the noise systematically. This also makes the conversion between the cleartext Boolean circuits and the encrypted circuits quite straightforward in practice.

However, performing a bootstrapping at each bivariate Boolean gate is very expensive when we want to evaluate large circuits and seem unnecessary. One idea to make the evaluation more efficient would be to mix the bootstrapping with some leveled operations, at the cost of loosing the ease of not caring about noise growth. But this idea cannot be immediately applied when it comes to gate bootstrapping: in fact, the bootstrapping also takes care of ensuring a fixed encoding in the ciphertexts, that may not be ensured if we introduce leveled operations. Furthermore, TFHE can only evaluate linear combinations between LWE ciphertexts; non linear operations would require the use of bootstrapping or of a non native product between LWE ciphertexts (e.g., an external product which is not composable because it makes use of different input ciphertext types). This is especially problematic when we want to evaluate an **AND** gate, for instance.

To be more clear, in gate bootstrapping, messages are encoded with what we call one *"bit of padding"*: meaning that we know that the MSB of the plaintext (without noise) is set to zero. This bit is used to perform a linear combination while preserving the (plaintext) MSB of this combination so we can bootstrap it (the function is negacyclic, so do not need an additional bit of padding) and get a correct result. Roughly speaking, the initial linear combination evaluates the linear part of the gate and consumes the bit of padding, while the bootstrapping takes care of the evaluation of the non-linear part of the gate, reduces the noise and brings the bit of padding back to be able to perform a future operation.

We propose a novel approach based on the **GenPBS** and **LWEMult** which removes both the constraint of padding bits and the difficulties with the non-linear leveled evaluations. Thus, this offers the possibility of computing series of Boolean gates without the need of computing a bootstrap for every gate. A **GenPBS** should only be computed to reduce the noise when needed. In Lemma 1, we only describe some of the most common Boolean gates (i.e., **XOR**, **NOT** and **AND**), whose combination offers functional completeness. The other gates can be obtained by combining these operations.

Lemma 1. *Let* $b_i \in \{0,1\}$ *such that* $\mathsf{ct}_i = \mathsf{LWE}_{\mathbf{s}}(b_i \cdot \frac{q}{2}) \in \mathbb{Z}_q^{n+1}$, *for* $i \in \{1,2\}$. *Let* $\left(\mathbf{0}, \frac{q}{2}\right) \in \mathbb{Z}_q^{n+1}$ *be a trivial LWE ciphertext. Then, the following equalities between Boolean gates and homormorphic operators hold:*

$$\begin{aligned}
&\mathsf{ct}_1 \,\mathbf{XOR}\, \mathsf{ct}_2 = \mathsf{ct}_1 + \mathsf{ct}_2 \\
&\mathsf{ct}_1 \,\mathbf{AND}\, \mathsf{ct}_2 = \mathbf{LWEMult}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{RLK}, \mathsf{KSK}) \\
&\mathbf{NOT}\, \mathsf{ct}_1 = \mathsf{ct}_1 + \left(\mathbf{0}, \frac{q}{2}\right)
\end{aligned}$$

Proof (Sketch). A bit is naturally encoded as a 0 (resp. $\frac{q}{2}$) if its value is 0 (resp. 1). Then the Boolean gates **XOR** and **NOT** stem from that encoding. The **AND** is a direct application of the **LWEMult**. □

The noise increases after each computed gate since no bootstrap is performed. Then, after chaining many of them, a noise reduction might be required. We propose two simple processes exploiting the **GenPBS** with the (negacyclic) sign function.

Lemma 2. *Let* $\mathsf{ct}_{\mathsf{in}}$ *be a LWE ciphertext resulting from a Boolean circuit with gates defined as in Lemma 1. Then, each of the following operators allows to bootstrap the ciphertext during the Boolean circuit evaluation:*

$$\mathsf{ct}_{\mathsf{out}} \leftarrow \mathbf{GenPBS}(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, P_1 \cdot X^{N/2}, \Delta_{\mathsf{out}} = \frac{q}{4}, \varkappa = 0, \vartheta = 0) + \left(\mathbf{0}, \frac{q}{4}\right) \tag{4}$$

$$\mathsf{ct}_{\mathsf{out}} \leftarrow \mathbf{GenPBS}(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, P_f = \sum_{i=\frac{N}{4}}^{\frac{3N}{4}-1} X^i, \Delta_{\mathsf{out}} = \frac{q}{2}, \varkappa = -1, \vartheta = 0) \tag{5}$$

Proof. The first method 4 uses **GenPBS** with the parameters $\Delta_{\mathsf{out}}=\frac{q}{4}, \varkappa=0, \vartheta=0$ and $P_f=P_1*X^{N/2}$. The output of the **GenPBS** gives $\mathsf{ct}_{\mathrm{tmp}}=\mathsf{LWE}_{\mathbf{s}}(\pm\frac{q}{4})$. Then, depending on the sign, the term $\mathsf{ct}_{\mathrm{tmp}}+(\mathbf{0},\frac{q}{4})$ is equal to $\mathsf{LWE}_{\mathbf{s}}(0)$ or $\mathsf{ct}_{\mathrm{tmp}}=\mathsf{LWE}_{\mathbf{s}}(\frac{q}{2})$.

The second approach 5 uses other parameters for the modulus switching which can be seen as shifted of one bit, i.e., $\varkappa = -1$, $\vartheta = 0$ and $\Delta_{\mathsf{out}} = \frac{q}{2}$. In this case, the sign does not impact the value of the encoded bit, since $\pm 0 = 0$ and $\pm\frac{q}{2} = \frac{q}{2}$. Then, evaluating **GenPBS** with the function $P_f=\sum_{i=\frac{N}{4}}^{\frac{3N}{4}-1} X^i$ and $\Delta_{\mathsf{out}}=\frac{q}{2}$, we obtain $\mathsf{ct}_{\mathsf{out}}=\mathsf{LWE}_{\mathbf{s}}(\pm 0)$ or $\mathsf{LWE}_{\mathbf{s}}(\pm\frac{q}{2})$. □

5.1.2 Modular Power of 2 Arithmetic We generalize the faster Boolean circuit method (described in Lemma 1) to any power of two modular integer circuits. This enables a more efficient exact arithmetic modulo 2^p for some integer p. For $i \in \{1, 2\}$, let $\mathsf{ct}_i = \mathsf{LWE}_{\mathbf{s}}(m_i \cdot \frac{q}{2^p})$ be a LWE ciphertext encrypting the message $m_i \in [\![0, 2^p[\![$ (i.e., m_i has a precision of p bits). As in the case of faster Boolean arithmetic, we define three natural homomorphic operators to mimic modular 2^p arithmetic: the addition ($\mathbf{Add}_{2^p}$) that is evaluated as an homomorphic LWE addition, the multiplication ($\mathbf{Mul}_{2^p}$) that is evaluated as a **LWEMult**, and the unary opposite ($\mathbf{Opp}_{2^p}$) that is obtained by simply negating the LWE input.

When we deal with integers encoded with more than one bit, functions we have to apply during a PBS are no longer negacyclic. It means that without a **WoP-PBS** we would have to have at least 2 bits of padding (one for a linear combination and another one for the PBS with non-negacyclic function evaluation). This results in a big N when we want to work with larger powers of two. With a **WoP-PBS**, we do not need to have bits of padding. Then, we can simply compute leveled additions and multiplications, and only use a **WoP-PBS** when we have to reset the noise to a lower level.

5.1.3 From Power of 2 Modular Arithmetic to Exact Integer Arithmetic We now present some operators allowing to extend homomorphic computation modulo a power of two modular to bigger integer arithmetic. To do so, we will use a few LWE ciphertexts to represent a single big integer. These required operations offer the possibility to compute an exact integer multiplication between two LWE ciphertexts as in 5.1.2 and keeping the LSB of the computation. However, we also need to be able to recover the MSB of additions and multiplications for carry propgation when we deal with big integers encrypted with several ciphertexts. The operators keeping the MSB of the computation between two messages $m_1, m_2 \in [\![0, 2^p[\![$ are defined as: $\mathbf{Add}_{2^p}^{\mathrm{MSB}} : (m_1, m_2) \mapsto \left\lfloor \frac{m_1+m_2}{2^p} \right\rceil \mod 2^p$ and $\mathbf{Mul}_{2^p}^{\mathrm{MSB}} : (m_1, m_2) \mapsto \left\lfloor \frac{m_1 \cdot m_2}{2^p} \right\rceil \mod 2^p$ and their implementation is described in Algorithm 7.

In Algorithm 7, to improve efficiency, we can remove both **PublicKS** and include them in the relinearization steps of the previous **WoP-PBS**. If parameters allow it, one might also replace Lines 6 and 7 of Algorithm 7 by a single **WoP-PBS** to extract the MSB directly.

Lemma 3 (MSB operations). *For $i \in \{1, 2\}$, let $\mathsf{ct}_i = \mathsf{LWE}_{\mathbf{s}}(m_i \cdot \Delta)$ be two LWE ciphertexts, encrypting $m_i \cdot \Delta$ with $0 \leq m_i < 2^p$ and $\Delta = \frac{q}{2^p}$, both encrypted under the same secret key $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{Z}_q^n$, with noise sampled in χ_{σ_i}. Let $\mathsf{BSK}, \mathsf{KSK}, \mathsf{RLK}$ be defined as in Theorem 5.*

Then, Algorithm 7 is able to compute a new LWE ciphertext $\mathsf{ct}_{\mathsf{out}}$, encrypting the MSB of the sum, i.e., the carry, $\left[\left\lfloor \frac{m_1+m_2}{2^p} \right\rceil\right]_{2^p} \cdot \Delta$ (resp. a new LWE ciphertext $\mathsf{ct}_{\mathsf{out}}$, encrypting the MSB of the product $\left[\left\lfloor \frac{m_1 \cdot m_2}{2^p} \right\rceil\right]_{2^p} \cdot \Delta$), under the secret key $\mathbf{s}'$. The variance of the noise of $\mathsf{ct}_{\mathsf{out}}$ can be estimated by composing the noise formulas of the different operations composing the algorithm.

The complexity of Algorithm 7 is:

$$\begin{aligned}
\mathbb{C}_{\mathbf{Add}_{2^p}^{\mathrm{MSB}}}^{(n,\ell_{\mathbf{PBS}},k_1,N_1,\ell_{\mathbf{KS}},\ell_{\mathbf{RL}},k_2,N_2)} &= 3\mathbb{C}_{\mathbf{WoP\text{-}PBS}}^{(n,\ell_{\mathbf{PBS}},k_1,N_1,\ell_{\mathbf{KS}},\ell_{\mathbf{RL}},k_2,N_2)} + 2\mathbb{C}_{\mathsf{PublicKS}}^{(1,\ell_{\mathbf{KS}},k_2N_2,1,n)} \\
&\quad + 2(N_2+1)\mathbb{C}_{\mathsf{add}} \\
\mathbb{C}_{\mathbf{Mul}_{2^p}^{\mathrm{MSB}}}^{(n,\ell_{\mathbf{PBS}},k_1,N_1,\ell_{\mathbf{KS}},\ell_{\mathbf{RL}},k_2,N_2)} &= 3\mathbb{C}_{\mathbf{WoP\text{-}PBS}}^{(n,\ell_{\mathbf{PBS}},k_1,N_1,\ell_{\mathbf{KS}},\ell_{\mathbf{RL}},k_2,N_2)} + 2\mathbb{C}_{\mathsf{PublicKS}}^{(1,\ell_{\mathbf{KS}},k_2N_2,1,n)} \\
&\quad + (N_2+1)\mathbb{C}_{\mathsf{add}} + \mathbb{C}_{\mathsf{LWEMult}}^{(\ell_{\mathbf{KS}},\ell_{\mathbf{RL}},k_2N_2,1,k_2N_2)}
\end{aligned} \tag{6}$$

Proof (sketch). The first two **WoP-PBS** of the algorithm send the two messages m_1 and m_2 to a lower scaling factor $\frac{q}{2^{2p}}$. This way, when the leveled addition (resp. the **LWEMult**) operation is performed, the new precision $2p$ will be able to store the entire (both MSB and LSB) exact result. The third **WoP-PBS** is used to extract only the LSB of the result, that will be subtracted from the result of the previous computation to obtain an encryption of the MSB at scaling factor $\frac{q}{2^p}$, i.e, ready to be used in the following computation. Observe that the **PublicKS** are used in order to switch the secret key in order to be compatible with the following operation. □

5.2 Faster Circuit Bootstrapping

In TFHE [8], authors present a technique called *circuit bootstrapping*, that allows to convert an LWE ciphertext into an GGSW ciphertext. The circuit bootstrap-

Algorithm 7: $\mathsf{ct}_{\mathsf{out}} \leftarrow$ $\boxed{\mathbf{Add}_{2^p}^{\mathrm{MSB}}}$ $\boxed{\mathbf{Mul}_{2^p}^{\mathrm{MSB}}}$ $(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{BSK}, \mathsf{KSK}_1, \mathsf{KSK}_2, \mathsf{RLK})$

Context:
$$\begin{cases} \mathbf{s} = (s_1, \cdots, s_n) \in \mathbb{Z}_q^n \\ \mathbf{s}' = (s'_1, \cdots, s'_{kN}) \in \mathbb{Z}_q^{kN} \\ \mathbf{S}' = \left(S'^{(1)}, \ldots, S'^{(k)}\right) \in \mathfrak{R}_q^k \\ \forall 1 \leq i \leq k,\ S'^{(i)} = \sum_{j=0}^{N-1} s'_{(i-1)\cdot N+j+1} X^j \in \mathfrak{R}_q \\ \Delta = \frac{q}{2^p} \in \mathbb{Z}_q \\ 0 \leq m_1, m_2 < 2^p \\ P_{\mathsf{id}} : \text{a redundant LUT for } x \mapsto x \text{ (identity function)} \end{cases}$$

Input:
$$\begin{cases} \mathsf{ct}_1 = \mathsf{LWE}_{\mathbf{s}}(m_1 \cdot \Delta) \in \mathbb{Z}_q^{n+1} \\ \mathsf{ct}_2 = \mathsf{LWE}_{\mathbf{s}}(m_2 \cdot \Delta) \in \mathbb{Z}_q^{n+1} \\ \mathsf{BSK} = \left\{\mathsf{BSK}_i = \mathsf{GGSW}_{\mathbf{S}'}^{(\mathfrak{B},\ell)}(s_i)\right\}_{1 \leq i \leq n} : \text{ a bootstrapping key from } \mathbf{s} \text{ to } \mathbf{S}' \\ \mathsf{KSK}_1 = \left\{\overline{\mathsf{CT}}_i = \mathsf{GLev}_{\mathbf{S}'}^{(\mathfrak{B},\ell)}(s'_i)\right\}_{1 \leq i \leq kN} : \text{ a key switching key from } \mathbf{s}' \text{ to } \mathbf{S}' \\ \mathsf{KSK}_2 = \left\{\overline{\mathsf{ct}}_i = \mathsf{Lev}_{\mathbf{s}}^{(\mathfrak{B},\ell)}(s'_i)\right\}_{1 \leq i \leq kN} : \text{ a key switching key from } \mathbf{s}' \text{ to } \mathbf{s} \\ \mathsf{RLK} = \left\{\overline{\mathsf{CT}}_{i,j} = \mathsf{GLev}_{\mathbf{S}'}^{(\mathfrak{B},\ell)}\left(S'_i \cdot S'_j\right)\right\}_{1 \leq i \leq k}^{1 \leq j \leq i} : \text{ a relinearization key for } \mathbf{S}' \end{cases}$$

Output: $\boxed{\mathsf{ct}_{\mathsf{out}} = \mathsf{LWE}_{\mathbf{s}}\left(\left[\left\lfloor \frac{m_1+m_2}{2^p} \right\rceil\right]_{2^p} \cdot \Delta\right)}$ $\boxed{\mathsf{ct}_{\mathsf{out}} = \mathsf{LWE}_{\mathbf{s}}\left(\left[\lfloor \frac{m_1 \cdot m_2}{2^p} \rceil\right]_{2^p} \cdot \Delta\right)}$

begin

/* add p bits of padding */

$\mathsf{ct}'_1 \leftarrow \mathbf{WoP\text{-}PBS}(\mathsf{ct}_1, \mathsf{BSK}, \mathsf{RLK}, \mathsf{KSK}_1, P_{\mathsf{id}}, \Delta/2^p, 0, 0)$;

$\mathsf{ct}'_2 \leftarrow \mathbf{WoP\text{-}PBS}(\mathsf{ct}_2, \mathsf{BSK}, \mathsf{RLK}, \mathsf{KSK}_1, P_{\mathsf{id}}, \Delta/2^p, 0, 0)$;

/* compute the operation */

$\boxed{\mathsf{ct}' \leftarrow \mathsf{ct}'_1 + \mathsf{ct}'_2}$ $\boxed{\mathsf{ct}' \leftarrow \mathbf{LWEMult}(\mathsf{ct}'_1, \mathsf{ct}'_2, \mathsf{RLK}, \mathsf{KSK}_1)}$;

/* key switch */

$\mathsf{ct}'' \leftarrow \mathbf{PublicKS}(\mathsf{ct}', \mathsf{KSK}_2, \mathsf{Id})$;

/* extract the LSB */

$\mathsf{ct}'_{\mathrm{LSB}} \leftarrow \mathbf{WoP\text{-}PBS}(\mathsf{ct}'', \mathsf{BSK}, \mathsf{RLK}, \mathsf{KSK}_1, P_{\mathsf{id}}, \Delta/2^p, p, 0)$;

/* subtract the LSB to only keep the MSB */

$\mathsf{ct} \leftarrow \mathsf{ct}' - \mathsf{ct}'_{\mathrm{LSB}}$;

/* key switch */

$\mathsf{ct}_{\mathsf{out}} \leftarrow \mathbf{PublicKS}(\mathsf{ct}, \mathsf{KSK}_2, \mathsf{Id})$;

end

ping is necessary for leveled evaluations using the external product: the latter's inputs are both GLWE and GGSW ciphertexts, while its output is a GLWE ciphertext. To sum up, circuit bootstrapping allows to build a new GGSW ciphertext from an LWE ciphertext so one can use it as input to an external product for instance.

The authors of [8] observe that a GGSW ciphertext, encrypting a message $\mu \in \mathbb{Z}$ (μ is binary in their application) under the secret key $\mathbf{S}=(S_1,\ldots,S_k,S_{k-1}=-1)$, is composed by $(k+1)\ell$ GLWE ciphertexts encrypting $\mu \cdot S_i \cdot \frac{q}{\mathfrak{B}^j}$, for $1 \leq i \leq k+1$ and $1 \leq j \leq \ell$. As already mentioned in Section 2, the goal of circuit bootstrapping is to build one by one all the GLWE ciphertexts composing the output GGSW. In order to do that, it performs the following two steps:

– The first step performs ℓ *independent TFHE PBS* to transform the input LWE encryption of μ into independent LWE encryptions of $\mu \cdot \frac{q}{\mathfrak{B}^j}$.

	Gate Bootstrap TFHE	**Binary arithmetic ($p = 1$) as in Sec. 5.1.1**	**Integer arithmetic ($p > 1$) generalization in Sec. 5.1.3**
$\mathbf{Opp}_{2^p}$	Negation	Addition with a constant	Negation
$\mathbf{Add}_{2^p}$	Bootstrapped XOR	Homomorphic Add	Homomorphic Add
$\mathbf{Add}_{2^p}^{\mathrm{MSB}}$	Bootstrapped AND	MultLWE	3 WoPBS + 2 Homomorphic Add + 2 public key switch
$\mathbf{Mul}_{2^p}$	Bootstrapped AND	MultLWE	MultLWE
$\mathbf{Mul}_{2^p}^{\mathrm{MSB}}$	$x \mapsto 0$	$x \mapsto 0$	3 WoPBS + MultLWE + Homomorphic Add + 2 public key switch
Noise reduction frequency	PBS at each gate	PBS when necessary	WoPBS when necessary

Table 2. Generalization of TFHE gate bootstrapping.

– The second step performs a list of $(k+1)\ell$ private key switchings from LWE to GLWE to multiply the messages $\mu \cdot \frac{q}{\mathfrak{B}^j}$ obtained in the first step by the elements of the secret key S_i, and so to obtain the different lines of the output GGSW.

Here, we propose a faster method based on the **PBSmanyLUT** algorithm (Algorithm 6). In a nutshell, the idea is to replace the ℓ PBS of the first step by only one **PBSmanyLUT** (that costs exactly the same as a one of the ℓ original PBS and do not increase the noise). Since the most costly part of the circuit boostrapping is due to the PBS part, the overall complexity is then roughly reduced by a factor ℓ. In [8], $\ell = 2$, so we have an improvement of a factor 2 on the PBS part, without any impact on the noise.

Lemma 4. *Let consider the circuit boostrapping algorithm as described in [8, Alg. 11]. The ℓ independent bootstrappings (line 2) could be replaced by:*

$$\begin{cases} \{\mathsf{ct}_i\}_{i\in[1,\ell]} \leftarrow \mathbf{PBSmanyLUT}(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, P \cdot X^{N/2^{\rho+1}}, 1, \varkappa = 0, \rho = \lceil \log_2(\ell) \rceil) \\ \forall i \in [1,\ell], \mathsf{ct}_i + \left(\mathbf{0}, \frac{q}{2\mathfrak{B}^i}\right) \end{cases}$$

$$\textit{with } P(X) = \sum_{i=0}^{\frac{N}{2^\rho}-1} \sum_{j=0}^{2^\rho-1} \frac{q}{2\mathfrak{B}^j} X^{2^\rho \cdot i + j}.$$

Proof. By calling **PBSmanyLUT** with $\rho = \lceil \log_2(\ell) \rceil$, we are able to compute ℓ **PBS** in parallel. The polynomial P represents the LUT:

$$\Bigg(\underbrace{\frac{q}{2\mathfrak{B}^1}, \dots, \frac{q}{2\mathfrak{B}^\ell}, 0, \dots, 0}_{2^\rho \text{ elements}}, \underbrace{\frac{q}{2\mathfrak{B}^1}, \dots, \frac{q}{2\mathfrak{B}^\ell}, 0, \dots, 0}_{2^\rho \text{ elements}}, \dots, \underbrace{\frac{q}{2\mathfrak{B}^1}, \dots, \frac{q}{2\mathfrak{B}^\ell}, 0, \dots, 0}_{2^\rho \text{ elements}} \Bigg)$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{N'=N/2^\rho \text{ elements}}$$

In the end, for $i \in [1,\ell]$, $\mathsf{ct}_i = \mathsf{LWE}_{\mathbf{S}}(\pm\frac{q}{2\mathfrak{B}^i})$, with the sign depending on the plaintext value. By adding the trivial ciphertext $(\mathbf{0}, \frac{q}{2\mathfrak{B}^i})$ to the ct_i, we either get $\mathsf{ct}_i = \mathsf{LWE}_{\mathbf{S}}(\frac{q}{\mathfrak{B}^i})$ or $\mathsf{LWE}_{\mathbf{S}}(0)$, as expected. □

5.3 Large Precision Without Padding (Programmable) Bootstrapping

We first describe a way to efficiently bootstrap an LWE ciphertext with larger precision and then show how to also compute a PBS on such ciphertexts. These algorithms do not require the input LWE ciphertext to have a bit of padding.

5.3.1 Larger Precision Without Padding Bootstrapping We introduce a new procedure in Algorithm 8 to homomorphically decompose a message encrypted inside a ciphertext in α ciphertexts each encrypting a small chunk of the original message. The key of the efficiency of this algorithm is to begin by extracting the least significant bits instead of the most significant bits. To do so, we use the previously introduced parameter $\varkappa$ to remove some of the most significant bits of the input message m and apply the bootstrapping algorithm on the remaining bits as described in subsection 3.2. The bootstrapping algorithm must be a **WoP-PBS** (Algorithm 4 or 5) as the value of most significant bit is not guaranteed to be set to zero. This procedure allows us to obtain an encryption of the least significant bits of the message. Next, by subtracting this result to the input ciphertext, we remove the least significant bits of the input message. This gives a new ciphertext encrypting only the most significant bits of the input message. From now on, this procedure is then repeated on the resulting ciphertext until we obtain α ciphertexts, each encrypting $m_i\Delta_i$ such that $m_{\mathsf{in}}\Delta_{\mathsf{in}}=\sum_{i=0}^{\alpha-1} m_i\Delta_i$. This process is somehow similar to the approach called *Digit Extraction* applied on the BGV/BFV schemes, presented in [17,5].

This entails a significantly better complexity than the solution explained in the Limitation E as each bootstrap only needs a ring dimension big enough to bootstrap correctly the number of bits of each chunk instead of having to be big enough to bootstrap correctly the total number of bits of the input ciphertext.

Efficiency might be improved within the multiplication inside each **WoP-PBS** by adding a keyswitching during the relinearization step to reduce the size of the LWE dimension. As the complexity of the **WoP-PBS** depends on this LWE dimension, this will result in a faster version of Algorithm 8.

Lemma 5. *Let* $\mathsf{ct}_{\mathsf{in}}=\mathsf{LWE}_{\mathbf{s}}(m_{\mathsf{in}}\cdot\Delta_{\mathsf{in}})\in\mathbb{Z}_q^{n+1}$ *be a LWE ciphertext, encrypting* $m_{\mathsf{in}}\cdot\Delta_{\mathsf{in}}\in\mathbb{Z}_q$. *under the LWE secret key* $\mathbf{s}=(s_1,\ldots,s_n)\in\mathbb{Z}_q^n$, *with noise sampled from* χ_σ. *Let* $\mathsf{BSK},\mathsf{KSK}$ *and* RLK *as defined in Theorem 5. Let* $\mathfrak{L}=\{d_i\}_{i\in[0,\alpha-1]}$ *with* $d_i\in\mathbb{N}^*$ *s.t.* $\Delta_{\mathsf{in}}2^{\sum_{i=0}^{\alpha-1} d_i}\leq q$ *be the list defining the bit size of each output chunk. Algorithm 8 computes* $\alpha\in\mathbb{N}^*$ *new LWE ciphertexts* $\{\mathsf{ct}_{\mathsf{out},i}\}_{i\in[0,\alpha-1]}$, *where each one of them encrypts* $m_i\cdot\Delta_i$, *where* $\Delta_i=\Delta_{\mathsf{in}}\cdot 2^{\sum_{j=1}^{i-1} d_j}$, *under the secret key* $\mathbf{s}'$. *The variances of the noise is* $\mathsf{Var}(\mathsf{ct}_{\mathsf{out},i})=\mathsf{Var}(\mathbf{WoP\text{-}PBS})$. *The complexity is:* $\mathbb{C}_{\mathsf{Decomp}}^{(n,\ell_{\mathbf{PBS}},k_1,N_1,\ell_{\mathbf{KS}},\ell_{\mathbf{RL}},\alpha)}=\alpha\mathbb{C}_{\mathbf{WoP\text{-}PBS}_1}^{(n,\ell_{\mathbf{PBS}},k_1,N_1,\ell_{\mathbf{KS}},\ell_{\mathbf{RL}},1,n)}+\alpha(n+1)\mathbb{C}_{\mathsf{add}}+(\frac{\alpha(\alpha+1)}{2})\mathbb{C}_{\mathsf{add}}$.

An immediate application of Algorithm 8 is a high precision bootstrap algorithm. By using the decomposition and then adding each $\mathsf{ct}_{\mathsf{out},i}$, one can get - with the right parameters- a noise smaller than the one of the input ciphertext.

Algorithm 8: $\mathsf{ct}_{\mathsf{out}} \leftarrow$ **Decomp**$(\mathsf{ct}_{\mathsf{in}}, \mathsf{BSK}, \mathsf{RLK}, \mathsf{KSK}, \mathscr{L})$

Context:
$$\begin{cases} \mathbf{s} = (s_1, \cdots, s_n) \in \mathbb{Z}_q^n \\ \mathbf{s}' = (s'_1, \cdots, s'_N) \in \mathbb{Z}_q^{kN} \\ \mathbf{S}' = \left(S'^{(1)}, \ldots, S'^{(k)}\right) \in \mathfrak{R}_q^k \\ \forall 1 \leq i \leq k,\ S'^{(i)} = \sum_{j=0}^{N-1} s'_{(i-1)\cdot N+j+1} X^j \in \mathfrak{R}_q \\ \{P_{f_i}\}_{i\in[0,\alpha-1]} : \text{LUTs for the functions } f_i \\ \forall i \in [1, \alpha-1], \Delta_i = \Delta_{\mathsf{in}} \cdot 2^{\sum_{j=1}^{i-1} d_j} \leq q \\ \Delta_0 = \Delta_{\mathsf{in}}, m_{\mathsf{in}}\Delta_{\mathsf{in}} = \sum_{i=0}^{\alpha-1} m_i \Delta_i \end{cases}$$

Input:
$$\begin{cases} \mathsf{ct}_{\mathsf{in}} = \mathsf{LWE}_{\mathbf{s}}(m_{\mathsf{in}} \cdot \Delta_{\mathsf{in}}) \in \mathbb{Z}_q^{n+1} \\ \mathsf{BSK}, \mathsf{KSK}, \mathsf{RLK} : \text{as defined in Algorithm 4} \\ \mathscr{L} = \{d_i\}_{i\in[0,\alpha-1]} \text{ with } d_i \in \mathbb{N}^* \end{cases}$$

Output: $\{\mathsf{ct}_{\mathsf{out},i} = \mathsf{LWE}_{\mathbf{s}'}(m_i \cdot \Delta_i)\}_{i\in[0,\alpha-1]}$

```
1 begin
2     ct ← ct_in
3     for i ∈ [0, α − 1] do
4         ϰ_i ← Σ_{j=i+1}^{α−1} d_j
5         ct_out,i ← WoP-PBS(ct, BSK, RLK, KSK, P_{f_i}, Δ_i, ϰ_i, 0)
6         ct ← ct − ct_out,i
7     end
8 end
```

5.3.2 Larger Precision WoP-PBS The **Tree-PBS** and the **ChainPBS** algorithms introduced in [16] allow to compute large precision programmable bootstrappings assuming that the input ciphertexts are already decomposed in chunks. In a nutshell, the idea behind the **Tree-PBS** is to encode a high-precision function in several LUTs. The first input ciphertext is used to select a subset among all the LUTs. This subset is then rearranged thanks to a key switching to build new encrypted LUTs. The previous steps can be repeated on the second input ciphertext, and so on. The **Tree-PBS** relies on the multi-output bootstrap from [4].

Thanks to the Algorithm 8, we are able to efficiently decompose a ciphertext. This allows to quickly switch from one representation (one ciphertext for one message) to another (e.g., several ciphertexts for one message) before calling the **Tree-PBS** or the **ChainPBS** algorithms. Moreover, we can replace the calls to PBS in both of the algorithms by a **WoP-PBS**. This relaxes the need to call **Tree-PBS** or **ChainPBS** with ciphertexts having a bit of padding. We call these two algorithms respectively the **Tree-WoP-PBS** and the **Chained-WoP-PBS**. Note that these algorithms can also be used to implement the $\mathbf{Add}_{2^p}^{\mathrm{MSB}}$ and $\mathbf{Mul}_{2^p}^{\mathrm{MSB}}$ operators.

6 Conclusion

This paper extends TFHE by exceeding some of its limitations. In particular, we present a new technique that allows to bootstrap messages without requiring a bit of padding, taking advantage of the GLWE multiplication (tensor product plus relinearization) and of our generalized version of TFHE's PBS. The latter

additionally allows to evaluate multiple LUTs in a single PBS for free when possible. These two techniques are particularly interesting when used to improve both the gate bootstrapping and the circuit bootstrapping techniques of TFHE. Thank to this new programmable bootstrapping, there is no need to compute a systematic PBS in every homomorphic Boolean gates as leveled additions and multiplications can be evaluated between when noise allows it. Additionally, the evaluation of Boolean circuits can be extended in order to support the evaluation of larger powers of 2 modular arithmetic and exact integer arithmetic. The circuit bootstrapping can be drastically improved, by replacing the evaluation of multiple PBS in the algorithm by a single **PBSmanyLUT** (that costs exactly as a PBS), without affecting the noise growth. Finally, we introduce two new efficient methods to bootstrap ciphertexts with large precision: a bootstrapping method to bring the noise down as well as a programmable bootstrapping evaluating univariate functions.

Open problems. All the new techniques proposed improve the state of the art by adding new features to TFHE and getting rid of some of its constraints. However, many enhancements could be added. In particular, one of the major bottleneck concerns the computation of the negacyclic convolutions of polynomials. The most efficient method based on the FFT inherently adds noise to ciphertext due to the use of floating points over 64 bits. When applied with larger floating point representation, the performances collapse. Thus, the study of alternative methods compatible with the TFHE parameters might improve the practical performances.

References

1. Boura, C., Gama, N., Georgieva, M., Jetchev, D.: CHIMERA: combining ring-lwe-based fully homomorphic encryption schemes. J. Math. Cryptol. **14**(1) (2020)
2. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. IACR Cryptology ePrint Archive **2012** (2012), http://eprint.iacr.org/2012/078
3. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012 (2012)
4. Carpov, S., Izabachène, M., Mollimard, V.: New techniques for multi-value input homomorphic evaluation and applications. In: Cryptographers' Track at the RSA Conference. Springer (2019)
5. Chen, H., Han, K.: Homomorphic lower digits removal and improved fhe bootstrapping. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer (2018)
6. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic encryption for arithmetic of approximate numbers. In: Advances in Cryptology - ASIACRYPT 2017 (2017)
7. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Advances in Cryptology - ASIACRYPT 2016 (2016)

8. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In: Advances in Cryptology - ASIACRYPT 2017 (2017)
9. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: fast fully homomorphic encryption over the torus. J. Cryptol. **33**(1) (2020)
10. Chillotti, I., Joye, M., Ligier, D., Orfila, J.B., Tap, S.: Concrete: Concrete operates on ciphertexts rapidly by extending tfhe. In: WAHC 2020–8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography. vol. 15 (2020)
11. Chillotti, I., Joye, M., Paillier, P.: Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In: Cyber Security Cryptography and Machine Learning - 5th International Symposium, CSCML 2021. Lecture Notes in Computer Science, vol. 12716. Springer (2021)
12. Ducas, L., Micciancio, D.: FHEW: bootstrapping homomorphic encryption in less than a second. In: Advances in Cryptology - EUROCRYPT 2015 (2015)
13. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptology ePrint Archive **2012** (2012), http://eprint.iacr.org/2012/144
14. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009 (2009)
15. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. IACR Cryptology ePrint Archive **2013** (2013), http://eprint.iacr.org/2013/340
16. Guimarães, A., Borin, E., Aranha, D.F.: Revisiting the functional bootstrap in TFHE. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(2) (2021)
17. Halevi, S., Shoup, V.: Bootstrapping for helib. In: Annual International conference on the theory and applications of cryptographic techniques. Springer (2015)
18. Lee, Y., Lee, J., Kim, Y.S., Kang, H., No, J.S.: High-precision and low-complexity approximate homomorphic encryption by error variance minimization. Cryptology ePrint Archive, Report 2020/1549 (2020), https://eprint.iacr.org/2020/1549
19. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) Advances in Cryptology - EUROCRYPT 2010. Lecture Notes in Computer Science, vol. 6110. Springer (2010)
20. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) Proceedings of the 37th Annual ACM Symposium on Theory of Computing, 2005. ACM (2005)
21. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. Des. Codes Cryptography **71**(1) (2014)
22. Stehlé, D., Steinfeld, R., Tanaka, K., Xagawa, K.: Efficient public key encryption based on ideal lattices. In: Matsui, M. (ed.) Advances in Cryptology - ASIACRYPT 2009. Lecture Notes in Computer Science, vol. 5912. Springer (2009)