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

Pierre-Emmanuel Clet(✉)[1], Aymen Boudguiga[1], and Renaud Sirdey[1]

Université Paris-Saclay, CEA LIST, 91120, Palaiseau, France
name.surname@cea.fr

Abstract. The TFHE cryptosystem only supports small plaintext space, up to 5 bits with usual parameters. However, one solution to circumvent this limitation is to decompose input messages into a basis B over multiple ciphertexts. In this work, we introduce B-gates, an extension of logic gates to non binary bases, to compute base B logic circuit. The flexibility introduced by our approach improves the speed performance over previous approaches such as the so called tree-based method which requires an exponential amount of operations in the number of inputs. We provide experimental results using sorting as a benchmark application and, additionally, we obtain a speed-up of $\times 3$ in latency compared to state of the art BGV techniques for this application. As an additional result, we introduce a keyswitching key specific to packing TLWE ciphertexts into TRLWE ciphertexts with redundancy, which is of interest in many functional bootstrapping scenarios.

keywords: FHE; TFHE; functional bootstrapping

1 Introduction

Homomorphic encryption schemes having an efficient bootstrapping, such as TFHE [9], can be tweaked to evaluate look-up tables within their bootstrapping procedure. Hence, rather than being just used for refreshing ciphertexts (i.e., reducing their noise level), the bootstrapping becomes *functional* [5] or *programmable* [11] by allowing the evaluation of arbitrary functions as a bonus. These capabilities result in promising new approaches for improving the overall performances of homomorphic calculations, making the FHE "API" better suited to the evaluation of mathematical operators which are difficult to express as low complexity arithmetic circuits.

TFHE made a first breakthrough by proposing an efficient bootstrapping for homomorphic gate computation. Then, Bourse et al., [4] used the same bootstrapping algorithm for extracting the (encrypted) sign of an encrypted input. It was later used by Izabachene et al., [20] to evaluate a Hopfield network in the encrypted domain. Boura et al., [3] showed in 2019 that TFHE bootstrapping naturally allows to encode function evaluation via their representation as look-up tables (LUTs). Recently, different approaches have been investigated for functional bootstrapping improvement. Guimarães et al., [18] extended the ideas in Bourse et al., [5] to support the evaluation of functions over multiple inputs via LUTs. In this work, we build on the work of Guimarães et al., and extend it to provide a new evaluation technique for *arbitrary* circuits.

Related works – After Bourse et al., [4] described how the functional bootstrapping of TFHE computes a sign function, researchers investigated the implementation of $\mathrm{LUT}(f)$ for any function f with domain either half of the torus [11] or the entire torus [21, 23, 12, 15]. Encoding plaintext values only on $[0, \frac{1}{2}[$ (i.e., half of the torus) avoids the restriction of managing negacyclic functions during the bootstrapping. However, it reduces the size of the plaintext space as it is encoded on a smaller portion of the torus $\mathbb{T}$. Meanwhile, other methods [21, 23, 12, 15] support $\mathbb{T}$ as a plaintext space at the cost of adding more bootstrappings. Subsequently, Guimarães et al., have proposed the tree-based and chaining-based methods to evaluate functions of multiple ciphertexts by means of several functional bootstrapping. However, the efficiency of their tree-based method is limited by its exponential complexity relatively to the number of inputs. Meanwhile, the chaining-method is only well suited for computing specific functions, notably functions with a carry-like logic such as additions, and functions with a test-like logic such as the sign function [18].

Contributions – In this work, we first revisit the noise variances and success probabilities of the tree-based and chain-based method of Guimarães et al [18]. We also describe and compare in detail multiple methods to compute B-gates which are logic gates extended to bases B greater than 2, and serve as building blocks for the computation of base logic circuits. We show that the evaluation of circuits with our novel building block favorably compares to the tree-based method in terms of time performance. As an application, we show that our technique can be implemented efficiently in practice by taking as example a sorting algorithm[1].

Paper organization – The remainder of this paper is organized as follows. Section 2 reviews TFHE building blocks. Section 3 summarizes our framework to estimate the time complexity, the noise variance, and the success probability of algorithms build as a succession of functional bootstrapping operations. Section 4 describes techniques from the literature to evaluate look up tables with multiple inputs. Section 5 describes our method to compute multi-inputs functions using functional bootstrapping. Finally, Section 7 highlights the benefits of our method through the evaluation of a sorting algorithm.

[1] Papers [22] and [2] are applications of the Chocobo method to more intricate use cases.

2 Background

2.1 Notations

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

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

Given $x \xleftarrow{\mathcal{N}(\mu,\sigma^2)} \mathbb{R}$, the probability $P(a \leq x \leq b)$ is equal to $\frac{1}{2}(\mathrm{erf}(\frac{b-\mu}{\sqrt{2}\sigma}) - \mathrm{erf}(\frac{a-\mu}{\sqrt{2}\sigma}))$, where erf is the Gauss error function: $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}$. The same result apply to $x \xleftarrow{\mathcal{N}(0,\sigma^2)} \mathbb{T}$ as long as the distribution is concentrated as described in [9].

2.2 TFHE Structures

The TFHE encryption scheme was proposed in 2016 [8] and updated in [9]. It introduces the TLWE problem as an adaptation of the LWE problem to $\mathbb{T}$. TFHE relies on three structures to encrypt plaintexts defined over $\mathbb{T}$, $\mathbb{T}_N[X]$ or $\mathcal{R}$:

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

  Let $\mathcal{M} \subset \mathbb{T}_N[X]$ (or $\mathcal{M} \subset \mathbb{T}$) be the discrete message space[2]. To encrypt a message $m \in \mathcal{M}$, we add $(\mathbf{0}, m)$ to a fresh T(R)LWE sample. In the following, we refer to an encryption of m with the secret key $\boldsymbol{s}$ as a T(R)LWE ciphertext noted $\boldsymbol{c} \in \mathrm{T(R)LWE}_{\boldsymbol{s}}(m)$.

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

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

- **TRGSW Sample:** a TRGSW sample is a vector of TRLWE samples. To encrypt a message $m \in \mathcal{R}$, we add $m \cdot H$ to a TRGSW sample, where H is a gadget matrix[3] using an integer B_g as a basis for its decomposition. Chilotti et al., [9] defines an external product between a TRGSW sample A encrypting $m_a \in \mathcal{R}$ and a TRLWE sample $\boldsymbol{b}$ encrypting $m_b \in \mathbb{T}_N[X]$. This external product consists in multiplying A by the approximate decomposition of $\boldsymbol{b}$ with respect to H (Definition 3.12 in [9]). It yields an encryption of $m_a \cdot m_b$ i.e., a TRLWE sample $\boldsymbol{c} \in \text{TRLWE}_{\boldsymbol{s}}(m_a \cdot m_b)$. Otherwise, the external product allows also to compute a controlled MUX gate (CMUX) where the selector is $C_b \in \text{TRGSW}_{\boldsymbol{s}}(b)$, $b \in \{0, 1\}$, and the inputs are $\boldsymbol{c}_0 \in \text{TRLWE}_{\boldsymbol{s}}(m_0)$ and $\boldsymbol{c}_1 \in \text{TRLWE}_{\boldsymbol{s}}(m_1)$.

2.3 TFHE Bootstrapping

TFHE bootstrapping relies mainly on three building blocks:

- **Blind Rotate:** rotates a plaintext polynomial encrypted as $\boldsymbol{c} \in \text{TRLWE}_{\boldsymbol{k}}(m)$ by a position encrypted as $\boldsymbol{c}_p \in \text{TLWE}_{\boldsymbol{s}}(p)$. It takes as inputs: the TRLWE ciphertext $\boldsymbol{c} \in \text{TRLWE}_{\boldsymbol{k}}(m)$, a rescaled and rounded vector of $\boldsymbol{c}_p$ represented by $(a_1, \ldots, a_n, a_{n+1} = b)$ where $\forall i,\ a_i \in \mathbb{Z}_{2N}$, and n TRGSW ciphertexts encrypting $(s_1, \ldots, s_n)$ where $\forall i,\ s_i \in \mathbb{B}$. It returns a TRLWE ciphertext $\boldsymbol{c'} \in \text{TRLWE}_{\boldsymbol{k}}(X^{\langle \boldsymbol{a}, \boldsymbol{s} \rangle - b} \cdot m)$. In this paper, we will refer to this algorithm by $\mathsf{BlindRotate}$. With respect to the independence heuristic[4] stated in [9], the variance $\mathcal{V}_{BR}$ of the resulting noise after a $\mathsf{BlindRotate}$ satisfies the formula[5]:

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

V_c is the variance of the noise of the input ciphertext $\boldsymbol{c}$, and ϑ_{BK} is the the variance of the error of the bootstrapping key. l and Bg are the parameters defining the gadget matrix as in [9]. Note that the noise of the $\mathsf{BlindRotate}$ is independent from the noise of the encrypted position $\boldsymbol{c}_p$.

[3] Refer to Definition 3.6 and Lemma 3.7 in TFHE paper [9] for more information about the gadget matrix H.

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

[5] In [8], the formula holds a 4 instead of a 12. We believe that they did not take into account the standard deviation of a sample $x \xleftarrow{\$} [-\frac{1}{2}, \frac{1}{2}]$ when adapting the worst case noise formula to the average case.

- **TLWE Sample Extract:** takes as inputs both a ciphertext $\boldsymbol{c} \in \mathrm{TRLWE}_{\boldsymbol{k}}(m)$ and a position $p \in [\![0, N[\![$, and returns a TLWE ciphertext $\boldsymbol{c'} \in \mathrm{TLWE}_{\boldsymbol{k}}(m_p)$ where m_p is the p^{th} coefficient of the polynomial m. In this paper, we will refer to this algorithm by SampleExtract. This algorithm does not add any noise to the ciphertext.
- **Public Functional Keyswitching:** transforms a set of p ciphertexts $\boldsymbol{c}_i \in \mathrm{TLWE}_{\boldsymbol{k}}(m_i)$ into the resulting ciphertext $\boldsymbol{c'} \in \mathrm{T(R)LWE}_{\boldsymbol{s}}(f(m_1, \ldots, m_p))$, where $f()$ is a public linear morphism from $\mathbb{T}^p$ to $\mathbb{T}_N[X]$. Note that $N = 1$ when keyswitching to a TLWE ciphertext. This algorithm requires 2 parameters: the decomposition basis B_{KS} and the precision of the decomposition t. In this paper, we will refer to this algorithm by KeySwitch. As stated in [9], the variance $\mathcal{V}_{KS}$ of the resulting noise after a KeySwitch with $B_{KS} = 2$ follows the formula[6]:

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

  where V_c is the variance of the noise of the input ciphertext c, R is the Lipschitz constant of f and ϑ_{KS} the variance of the error of the keyswitching key. In this paper and in most cases, $R = 1$. When using a decomposition base B_{KS} and the tighter analysis from [18], the formula becomes:

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

  We give more details on this bound in Appendix A.

TFHE specifies a gate bootstrapping to reduce the noise level of a TLWE sample that encrypts the result of a boolean gate evaluation on two ciphertexts, each of them encrypting a binary input. TFHE gate bootstrapping steps are summarized in Algorithm 1. The step 1 consists in selecting a value $\hat{m} \in \mathbb{T}$ which will serve later for setting the coefficients of the test polynomial *testv* (in step 3). The step 2 rescales the components of the input ciphertext $\boldsymbol{c}$ as elements of $\mathbb{Z}_{2N}$. The step 3 defines the test polynomial *testv*. Note that for all $p \in [\![0, 2N[\![$, the constant term of $testv \cdot X^p$ is $\hat{m}$ if $p \in]\!]\frac{N}{2}, \frac{3N}{2}]\!]$ and $-\hat{m}$ otherwise. The step 4 returns an accumulator $ACC \in \mathrm{TRLWE}_{\boldsymbol{s'}}(testv \cdot X^{\langle \bar{\boldsymbol{a}}, \boldsymbol{s}\rangle - \bar{b}})$. Indeed, the constant term of ACC is $-\hat{m}$ if $\boldsymbol{c}$ encrypts 0, or $\hat{m}$ if $\boldsymbol{c}$ encrypts 1 as long as the noise of the ciphertext is small enough[7]. Then, step 5 creates a new ciphertext $\overline{\boldsymbol{c}}$ by extracting the constant term of ACC and adding to it $(\boldsymbol{0}, \hat{m})$. That is, $\overline{\boldsymbol{c}}$ either encrypts 0 if $\boldsymbol{c}$ encrypts 0, or m if $\boldsymbol{c}$ encrypts 1 (By choosing $m = \frac{1}{2}$, we get a fresh encryption of 1).

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

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

[6] Note that as of now, the formula from [9] has a discrepancy between Theorem4.1 and its proof. The formula from the proof should be followed.

[7] Further details on the proper bound of the noise are given in Section 3.

Algorithm 1 TFHE gate bootstrapping [9]

```
Input: a constant $m \in \mathbb{T}$, a TLWE sample $\boldsymbol{c} = (\boldsymbol{a}, b) \in \mathrm{TLWE}_{\boldsymbol{s}}(x \cdot \frac{1}{2})$ with $x \in \mathbb{B}$,
    a bootstrapping key $BK_{\boldsymbol{s} \to \boldsymbol{s'}} = (BK_i \in \mathrm{TRGSW}_{S'}(s_i))_{i \in [\![1,n]\!]}$ where $S'$ is the
    TRLWE interpretation of a secret key $\boldsymbol{s'}$
Output: a TLWE sample $\overline{\boldsymbol{c}} \in \mathrm{TLWE}_{\boldsymbol{s}}(x.m)$
1: Let $\hat{m} = \frac{1}{2}m \in \mathbb{T}$ (pick one of the two possible values)
2: Let $\bar{b} = \lfloor 2Nb \rceil$ and $\bar{a}_i = \lfloor 2Na_i \rceil \in \mathbb{Z}, \forall i \in [\![1,n]\!]$
3: Let $testv := (1 + X + \cdots + X^{N-1}) \cdot X^{\frac{N}{2}} \cdot \hat{m} \in \mathbb{T}_N[X]$
4: $ACC \leftarrow \mathsf{BlindRotate}((\mathbf{0}, testv), (\bar{a}_1, \ldots, \bar{a}_n, \bar{b}), (BK_1, \ldots, BK_n))$
5: $\overline{\boldsymbol{c}} = (\mathbf{0}, \hat{m}) + \mathsf{SampleExtract}(ACC)$
6: return $\mathsf{KeySwitch}_{\mathsf{s'} \to \mathsf{s}}(\overline{\boldsymbol{c}})$
```

2.4 TFHE Functional Bootstrapping

Functional bootstrapping [11, 21, 23, 12, 15] refers to TFHE's ability of evaluating a Look-Up Table (LUT) of any single input function during the bootstrapping. In particular, TFHE is well-suited for negacyclic function[8], as the plaintext space for TFHE is $\mathbb{T}$, where $[0, \frac{1}{2}[$ corresponds to positive values and $[\frac{1}{2}, 1[$ to negative ones, and the bootstrapping step 2 of the Algorithm 1 encodes elements from $\mathbb{T}$ into powers of X modulo $(X^N + 1)$, and $X^{\alpha+N} \equiv -X^{\alpha} \mod [X^N + 1]$.

In section 4, we will discuss methods for increasing the plaintext precision during a functional bootstrapping. We will focus on Guimarães et al., [18] ideas for combining several bootstrappings with many digits as input. These digits come from the decomposition of a plaintext in a basis B.

3 Time complexity, noise variance and success probability

In this section, we first give a reminder of operations relevant to evaluating the complexity of algorithm executions over TFHE when resorting to functional bootstrapping. We also provide a unified summary of the noise variance of the output of TFHE most useful operations. We then introduce a methodology to approximate the error of an algorithm. This new approximation holds two main benefits:

- We end up with linear approximations which are easier to compute and compare from a human's perspective.
- The exact result requires the computation of the product of success rates of each operations which may be rounded to 1 due to numerical precision

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

when extremely close to 1 (2^{-52}-close to 1 when using floats on 64 bits). Our linear estimation works directly with error rates which are not rounded to 0 unless truly negligible, resulting in more precise results.

3.1 Time complexity

The algorithms we present in this paper use primarily BlindRotate and Keyswitch operations. Those 2 operations are the most time consuming FHE operations. Note that a BlindRotate is $2 \cdot 10^4$ times slower than additions and 10^3 times slower than polynomial multiplications with default parameters in TFHElib[9].

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

3.2 Noise variance

The noise variances resulting from homomorphic computations with bootstrappings are calculated from the noise variances of the input ciphertexts, and the noise bounds of BlindRotate and KeySwitch (Equations 1 and 2). We summarize in Table 1 the noise variances for basic operations such as the addition of two ciphertexts or the multiplication of a ciphertext by a plaintext. These formulas can be used as building-blocks to compute the resulting noise variance of operations in the remainder of this paper.

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

Table 1. Obtained noise variances when applying basic operations to independent inputs: $\boldsymbol{c}_i$ is a TLWE ciphertext of variance V_i, $\boldsymbol{C}_i$ is a TRLWE ciphertext of variance V_i and P is a plaintext polynomial.

3.3 Success probability

The success probability of a BlindRotate is expressed with the Gauss error function. It depends on the noise of the input ciphertext, the size of the plaintext space and its encoding over the torus [15, 18]. Indeed, Clet et al., [14] showed that for a small plaintext space, functional bootstrapping is more efficient when the plaintexts are encoded over half of the torus.

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

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

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

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

$$\mathcal{P}_B(V_{\boldsymbol{c}}) = \operatorname{erf}(\frac{1}{4B\sqrt{2(V_{\boldsymbol{c}}+V_r)}}) \tag{3}$$

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

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

We note $\mathcal{F}_B = 1 - \mathcal{P}_B$ so that $\mathcal{F}$ represents the error rate of a $\mathsf{BlindRotate}$. We get thanks to Boole's inequality the following bound:

$$1 - \prod_{i=0}^{m-1} \mathcal{P}_B(V_{\boldsymbol{c}_i}) \leq \sum_{i=0}^{m-1} \mathcal{F}_B(V_{\boldsymbol{c}_i}) \tag{4}$$

Then, if each $\mathcal{F}_B(V_{\boldsymbol{c}_i})$ is small enough, this bound is also tight. We give a proof of tightness in Appendix B. From now on, we use Boole's inequality given in Equation 4 as a tight approximation of the error rate of our computations under this assumption.

4 LUTs with Multiple Encrypted Inputs

The aforementioned functional bootstrapping methods ([11, 21, 23, 12, 15]) are univariate and have a limited plaintext precision. They evaluate look-up tables with a size bounded by the degree N of the used cyclotomic polynomial. The size of the plaintext space gets even smaller when taking noise into account [15]. In addition, these methods are not suited for computing a LUT for a multivariate function f with two or more encrypted inputs. In order to overcome these issues, Guimarães et al., [18] proposed two methods for homomorphic computation with digits: a tree-based approach and a chaining approach. In this section, we discuss these two methods, give bounds for their noise variances and error rates and consistently compare the two approaches.

4.1 Tree-based Method

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

$$g_d : \begin{array}{rcl} [\![0, B-1]\!]^d & \to & [\![0, B^d - 1]\!] \\ (m_0, \ldots, m_{d-1}) & \mapsto & \sum_{i=0}^{d-1} m_i \cdot B^i \end{array} \qquad (5)$$

or simply d uncorrelated messages.

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

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

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

Then, we apply the $\mathsf{BlindRotate}$ algorithm to $\boldsymbol{c}_{d-1}$ and $\mathsf{TRLWE}(P_i), \forall i \in [\![0, B^{d-1} - 1]\!]$. That is, we rotate each $\mathsf{TRLWE}(P_i)$ by the encrypted position in $\boldsymbol{c}_{d-1}$. Finally, we run the $\mathsf{SampleExtract}$ algorithm on each of the rotated $\mathsf{TRLWE}(P_i)$. We end up with B^{d-1} TLWE ciphertexts, each encrypting $f \circ g_d^{-1}(m_{d-1} \cdot B^{d-1} + i)$ for $i \in [\![0, B^{d-1} - 1]\!]$. Then, we apply B^{d-2} $\mathsf{KeySwitch}$ to pack these B^{d-1} TLWE ciphertexts into B^{d-2} TRLWE ciphertexts, that correspond to the LUT of h where:

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

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

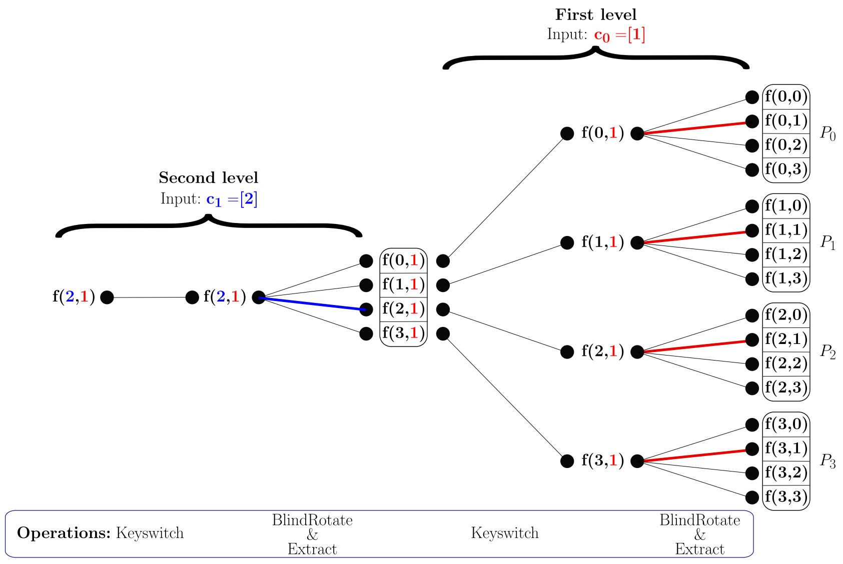

Fig. 1. Tree of depth 2 in basis 4 with inputs c_0 and c_1 encrypting 1 and 2 respectively.

We can accelerate the tree evaluation by encoding the first LUTs in plaintext polynomials rather than TRLWE ciphertexts. Then, we use the multi-value bootstrapping from [7] to compute only one BlindRotate instead of B^{d-1} at the first level of the tree. We call *selector* the result of the BlindRotate of the first layer. Thus, we now compute: $1+\sum_{i=0}^{d-2} B^i = 1 + \frac{B^{d-1}-1}{B-1}$ BlindRotate.

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

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

If we implement the multi-value bootstrapping [7] for the evaluation of the first level of the tree with the polynomials $(P_i)_{i\in[\![0,d-1]\!]}$, the noise bound increases to $(d-1+\max(||P_i||_2^2)) \cdot \mathcal{E}_{BR} + (d-1)\cdot \mathcal{E}_{KS}^{N,B} + \mathcal{E}_{KS}^{N,1}$. Note that in the worst case $\max_i(||P_i||_2^2) \leq (B+3)\cdot(B-1)^2$ which leads to a worst case noise variance of $(d-1+(B+3)\cdot(B-1)^2)\cdot \mathcal{E}_{BR} + (d-1)\cdot \mathcal{E}_{KS}^{N,B} + \mathcal{E}_{KS}^{N,1}$.

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

Each BlindRotate takes as input one of the ciphertext $\boldsymbol{c}_i$. As such, we can apply the Equation 4 from Section 3 to find the bound $\mathcal{F}_{\text{TM}} \leq \sum_{i=0}^{d-1} \mathcal{F}_B(V_{\boldsymbol{c}_i})$, where $\mathcal{F}_{\text{TM}}$ is the error rate of the tree based method. This bound holds true whether we use the multi-value bootstrapping or not.

4.2 Chaining Method

The chaining method has a much lower complexity and a lower error growth than the tree-based method. However, as stated in [18], it works only for a restricted set of functions, i.e., functions with carry-like logic.

We consider d TLWE ciphertexts $(\boldsymbol{c}_0,\dots,\boldsymbol{c}_{d-1})$ respectively encrypting the messages $(m_0,\dots,m_{d-1})$. We denote by $LC(a,b)$ *any* linear combination of a and b. Given a set of functions $(f_i)_{i\in[\![0,d-1]\!]}$ such that $f_i : [\![0,B-1]\!] \to [\![0,B-1]\!]$, we build with Algorithm 2 a function $f : [\![0,B-1]\!]^d \to [\![0,B-1]\!]$.

Algorithm 2 Chaining method

```
Input: A vector $(\boldsymbol{c}_0,\dots,\boldsymbol{c}_{d-1})$ of TLWE ciphertexts encrypting the vector of messages
    $(m_0,\dots,m_{d-1})$.
Output: A ciphertext encrypting $f(m_0,\dots,m_{d-1})$. $f$ is defined by the different linear
    combinations and the univariate functions $f_i$.
1: $\overline{\boldsymbol{c}}_0 \leftarrow f_0(\boldsymbol{c}_0)$
2: for $i \in [\![0,d-2]\!]$ do
3:     $\overline{\boldsymbol{c}}_{i+1} \leftarrow f_{i+1}(LC(\overline{\boldsymbol{c}}_i,\boldsymbol{c}_{i+1}))$
   return $\overline{\boldsymbol{c}}_{d-1}$
```

The functions $(f_i)_{i\in[\![0,d-1]\!]}$ can be implemented using any method from the state of the art for univariate functional bootstrapping. We remind that in this paper, we choose the usual encoding method where the plaintext space is restricted to half of the torus [11].

Noise variance Since Algorithm 2 ends with a functional bootstrapping, the resulting noise variance of its output ciphertext is bounded by $\mathcal{E}_{BR}+\mathcal{E}_{KS}^{N,1}$.

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

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

where LC_i is the linear combination used at the i^{th} step of Algorithm 2, c_{boot} is a freshly bootstrapped ciphertext and $\mathcal{F}_{\text{CM}}$ is the error rate of the chaining method.

4.3 Performances Comparison

Table 2 summarizes the noise variances and probabilities of success for the tree-based and chaining methods. We refer by TBM to the tree-based method without using the multi-value bootstrapping trick, by TMV to the tree-based method when using the multi-value bootstrapping, and by CM to the chaining method.

Method	**Noise variance**	**Bound on error rate**
TBM	$d \cdot \mathcal{E}_{BR} + (d-1)\mathcal{E}_{KS}^{N,B} + \mathcal{E}_{KS}^{N,1}$	$\sum_{i=0}^{d-1} \mathcal{F}_B(V_{\mathbf{c}_i})$
TMV	$(d-1+\max(\lvert\lvert P_i \rvert\rvert_2^2)) \cdot \mathcal{E}_{BR}$ $+(d-1)\cdot \mathcal{E}_{KS}^{N,B} + \mathcal{E}_{KS}^{N,1}$	$\sum_{i=0}^{d-1} \mathcal{F}_B(V_{\mathbf{c}_i})$
CM	$\mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1}$	$\mathcal{F}_B(V_{\mathbf{c}_0})$ $+\sum_{i=1}^{d-1} \mathcal{F}_B(V_{LC_i(c_{\mathsf{boot}},\mathbf{c}_i)})$

Table 2. Noise variance and success rate in basis B for d inputs

Table 3 summarizes the time complexity of both methods. As mentioned in Section 3, the time complexity is given as the number of BlindRotate and KeySwitch.

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

Table 3. Time complexity in basis B for d inputs

It is straightforward to see from Table 2 and 3 that CM leads to a lower noise variance with more efficient computation. These benefits come with limitations since CM is not a method that generalize well to every function. Besides, the error rate can be much higher with CM depending on the linear combination involved in Algorithm 2. Note that $\mathcal{F}_B(x) = \text{erfc}(\frac{1}{4B\sqrt{2\cdot(x+V_r)}})$. We use the bound $\text{erfc}(X) \lesssim \frac{e^{-X^2}}{X\sqrt{\pi}}$ which has the same order of magnitude as erfc as long as $X > 1$ to get that:

$$\mathcal{F}_B(x) \lesssim 4B \cdot e^{-\frac{1}{32B^2(x+V_r)}} \sqrt{\frac{2(x+V_r)}{\pi}} \tag{6}$$

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

5 Circuit Method

Both of the previous methods come with restrictions. On the one hand, the tree-based method requires an exponential number of BlindRotate relatively to the number of inputs. On the other hand, the chaining method can only be applied to some specific functions. Besides, both method may prove hard to efficiently integrate into FHE compilers such as Cingulata [6], Transpiler [17] or Concrete-compiler [10].

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

5.1 B-gates for logic circuits

For a given integer B, we define a B-gate as a function that takes as inputs at most two digits in $[\![0, B-1]\!]$ and outputs one digit in $[\![0, B-1]\!]$. This can be considered as an extension of logic gates in bases larger than two. Notably, when B is a power of two, these B-gates can be seen as a factorisation of multiple logic gates. As such, being able to compute B-gates is enough to compute any function thanks to their circuit representation. In addition, their circuit representation can make computation more efficient than with the tree-based method assuming an efficient way to compute B-gates.

In this section, we describe in detail how to build B-gates, which are the main building blocks of our circuits. Each B-gate can be computed either using the chaining method CM, the tree-based method TBM, or the tree-based method with multi-value bootstrapping TMV.

Using CM as a building block: We can combine two digits x and y with the bijection:

$$g_2 : \begin{array}{ccc} [\![0, B-1]\!]^2 & \to & [\![0, B^2-1]\!] \\ (x,y) & \mapsto & x + B \cdot y \end{array} \tag{7}$$

That is, we compute any B-gate with encrypted digits $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ by applying one functional bootstrapping to $g_2(\boldsymbol{c}_1, \boldsymbol{c}_2)$. To that end, we need to use a plaintext size of B^2 instead of B to encrypt each digit. We summarize in Table 4 and Table 5 the noise variance, the error rate and the time complexity of a generic gate using CM found following the formulas from Table 2. From now on, we call CM-gate gates computed with the CM method.

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

Table 4. CM-gate error rate and noise variance. $V_{\boldsymbol{c}_0}$ and $V_{\boldsymbol{c}_1}$ respectively represent the noise variance of the first and second inputs of the B-gate.

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

Table 5. CM-gate time complexity

Using TBM as a building block: B-gates can be computed as trees of depth 2. We summarize in Table 6 and Table 7 the noise variance, the error rate and the time complexity of a generic gate using TBM found following the formulas from Table 2 and Table 3. From now on, we call TBM-gate gates computed with the TBM method.

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

Table 6. TBM-gate error rate and noise variance. $V_{\boldsymbol{c}_0}$ and $V_{\boldsymbol{c}_1}$ respectively represent the noise variance of the first and second inputs of the B-gate.

Using TMV as a building block: Similarly to TBM, logic gates can be computed as trees of depth 2. Note that if multiple logic gates share an input, they can also share the "selector". This allows us to reduce the amount of BlindRotate in a circuit. We summarize in Table 8 and Table 9 the noise variance, the error

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

Table 7. TBM-gate time complexity

rate and the time complexity of a generic gate using TMV found following the formulas from Table 2 and Table 3. We consider the polynomials maximizing the error rate to bound the resulting error rate. From now on, we call TMV-gate gates computed with the TMV method.

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

Table 8. TMV-gate generic error rate and noise variance. $V_{\mathbf{c}_0}$ and $V_{\mathbf{c}_1}$ respectively represent the noise variance of the first and second inputs of the logic gate.

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

Table 9. TMV-gate generic time complexity

Example of TMV-gate Any B-gate can be computed with ease using CM and TBM. However, TMV-gates require to understand the multi-value bootstrapping technique and leads to a noticeable impact on the resulting noise variance. As an example, we describe the B-gate $\mathsf{AddGate}(x,y)=x+y[B]$. Note that we work on half of the torus, which prevents the modulus operation to be performed using a homomorphic addition.

Since additions are commutative, the same tree is used whether we want to use the first or the second input as the selector. We show in Figure 2 an example of this tree when $B=4$. The polynomials we end up with are the Q_i for $i \in [\![0,B-1]\!]$ where $Q_i=\sum_{k=0}^{\frac{N}{B}-1}\sum_{j=0}^{B-1}((i+k) \mod B)X^{k\cdot\frac{N}{B}+j}$. To apply the multi-value bootstrapping technique, we consider $P_i=(1-X)\cdot Q_i$. Then $P_0=(B-1)+\sum_{k=1}^{B-1}X^{k\cdot\frac{N}{B}}$ and for all $i>0$, $P_i=(2i-1)-B\cdot X^{(B-i)\cdot\frac{N}{B}}+\sum_{k=1}^{B-1}X^{k\cdot\frac{N}{B}}$ as polynomial for the noise formulas. We get that $\max_i(||P_i||_2^2)=(B-1)\cdot(5B-8)$. We summarize in Table 10 and Table 11 the noise variance, the error rate and the time complexity of a TMV-AddGate. The results are calculated following the formulas from Table 2 and Table 3.

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

Table 10. TMV-AddGate error rate and noise variance. $V_{\boldsymbol{c}_0}$ and $V_{\boldsymbol{c}_1}$ respectively represent the noise variance of the first and second inputs of the B-gate.

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

Table 11. TMV-AddGate time complexity. Note that the selector may be shared between multiple gates. Then, one less BlindRotate is necessary for each gate after the first.

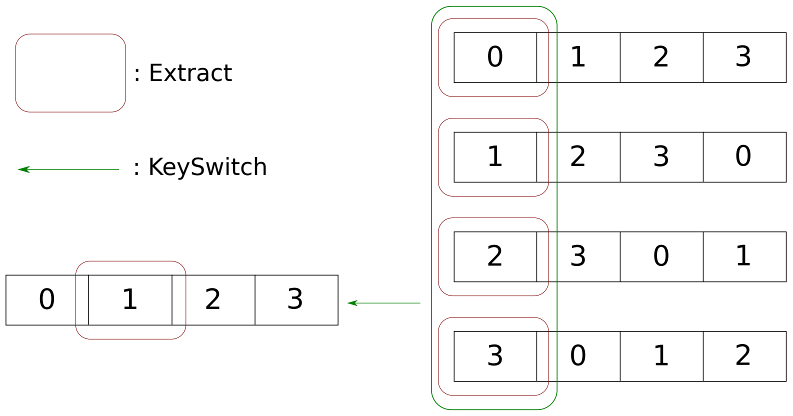

Fig. 2. Tree of AddGate

6 Empirical Performances

In this section we give empirical time results for computing generic B-gates. All computation were made on an Intel Core i5-8250U CPU @ 1.60GHz by extending the TFHE open source library[10]. We use the sets of parameters from Table 12,

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

Table 13 and Table 14 to achieve 128 bits of security and an error rate of at most 2^{-32}.

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

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

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

Table 12. Parameters sets with CM (λ=128)

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

Table 13. Parameters sets with TBM (λ=128)

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

Table 14. Parameters sets with TMV (λ=128)

Given these sets of parameters, we show in Table 15 the time requirements of each method in basis 4, 8 and 16.

It is interesting to note that when using a small basis B, CM-gates are the most efficient. However, the quadratic growth of the plaintext space relatively

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

Table 15. Gate Evaluation Time in ms

to B greatly degrades its performances with larger bases. On the other side of the spectrum, TMV-gates become more and more interesting with larger bases. Besides, TBM-gates seem of limited interest since it is outclassed by CM-gates for small basis, and by TMV-gates for larger basis. This is due to the linear growth of the number of operations required to perform a TBM-gate with the size of the basis. Note that the TFHElib does not natively have a TLWE to TRLWE KeySwitch implemented and our personal implementation of this KeySwitch does not make use of parallelism. Since this KeySwitch operation takes a large part of the computation time, the performances of TBM-gates and TMV-gates can be greatly optimised. Besides, the parameter sets used for TMV-gates are meant to work for any gate. In practice, the error bound must be tailored to specific gates, which will improve the overall performance of TMV-gates. Finally, performances must be optimized depending on the amount of memory available. For instance, multiple KeySwitch techniques exist as shown in Section A which lead to different trade-offs between noise growth, memory usage and speed.

7 Experimental Results on Sorting

In this section, we compare a circuit dedicated to sorting a list of encrypted inputs in base B to the direct application of the tree-based method and the BGV implementation of sorting from [13]. Note that merging lists efficiently in the homomorphic domain is a non trivial task, even though a recent work [16] describes an elegant way to achieve it. In order to keep our example simple, we describe in Algorithm 3 the bubble sort algorithm which is possible to implement homomorphically without merging lists.

Algorithm 3 Bubble Sort

```
Input: A list $(\boldsymbol{c}_0, \boldsymbol{c}_1, ..., \boldsymbol{c}_{d-1})$ of $d$ ciphertexts encrypting the messages
    $(m_0, m_1, ..., m_{d-1})$.
Output: A list of ciphertexts $(\boldsymbol{r}_0, \boldsymbol{r}_1, ..., \boldsymbol{r}_{d-1})$ encrypting the messages in sorted order.
1: $\boldsymbol{r}_0 = \boldsymbol{c}_0$
2: for $i \in [\![1, d-1]\!]$ do
3:     $(\boldsymbol{r}_0, \boldsymbol{r}_i) = (\min(\boldsymbol{r}_0, \boldsymbol{c}_i), \max(\boldsymbol{r}_0, \boldsymbol{c}_i))$
4:     for $j \in [\![1, i-1]\!]$ do
5:         $(\boldsymbol{r}_j, \boldsymbol{r}_i) = (\min(\boldsymbol{r}_j, \boldsymbol{r}_i), \max(\boldsymbol{r}_j, \boldsymbol{r}_i))$
   return $(\boldsymbol{r}_0, \boldsymbol{r}_1, ..., \boldsymbol{r}_{d-1})$
```

The speed and noise performance of B-gates computed with CM and TBM are unrelated to the specific B-gates. Thus, we reuse the same parameter sets as in Table 12 and Table 13. Besides, the only required B-gates are $\min(x, y)$ and $\max(x, y)$ gates. Since the resulting noise using TMV depends on the function computed, we give the bound on the noise relatively to these two B-gates:

– min gate noise variance: $(1 + B \cdot (B - 1)) \cdot \mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,B} + \mathcal{E}_{KS}^{N,1}$

– max gate noise variance: $(1 + 4 \cdot (B - 1)^2) \cdot \mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,B} + \mathcal{E}_{KS}^{N,1}$

As such, both noise variances are bounded by $(1+4\cdot(B-1)^2)\cdot\mathcal{E}_{BR}+\mathcal{E}_{KS}^{N,B}+\mathcal{E}_{KS}^{N,1}$. We use this formula to find the parameters from Table 16, leading to an error rate lower than 2^{-32} per gate with TMV.

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

Table 16. Parameters sets with TMV for sorting circuit (λ=128)

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

We show in Table 17 the empirical time result we get for each method. We call CM-circuit, TBM-circuit and TMV-circuit, circuits build with CM-gates, TBM-gates and TMV-gates, respectively.

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

Table 17. Sorting Time (in s) of 4 Inputs.

As seen in Table 17, the naive tree-based method becomes prohibitively long even for small bases and low number of inputs. Our circuit method allows more flexibility in the way homomorphic computation are performed and thus reach much better performances for functions with small circuit representation such as sorting functions. Table 17 also confirms the insight from Table 15 that the CM-gates are more efficient for small bases (2 or 3 bits) while TMV-gates become the most attractive for larger plaintext spaces.

We now compare our sorting algorithm to the BGV implementation from [13] of the approach of Iliashenko & Zucca [19] which is presently the fastest available for BGV/BFV. This method has also been run on the same machine as above.

We found two sets of parameters with at least 128 bits of security fitting for the BGV sorting algorithm using messages over 4 bits. They are respectively tailored for sorting 5 and 15 inputs, and are given in Table 18.

#Inputs	**Degree**	**Plaintext Mod**	**Ciphertext Mod (bits)**	**Noise**	$\boldsymbol{\lambda}$
5	16381	181	250	179	152
15	16381	181	290	167	137

Table 18. Parameters of the BGV scheme for our experiment.

We give in Table 19 the resulting performances of both our method and the BGV implementation for messages on 4 bits.

Inputs	**TMV-circuit**	[13]	**Speed-up**
5	20.83	64.169	$\times 3.08$
15	216.26	778.74	$\times 3.60$

Table 19. Sorting Performances (in s) and Speed-Up of our Method for 5 and 15 Inputs.

As shown in Table 19, our method achieves a noticeable speed-up of at least $\times 3$ compared to the latency of the sorting algorithm from [19, 13]. Although the BGV implementation can be batched to achieve better amortized timings, our results show that TFHE approaches can be competitive with BGV ones in terms of latency on this particular application.

8 Conclusion

We introduced and compared multiple techniques to compute B-gates as efficient building blocks for the computation of base B logic circuits. Our approach allows for efficient evaluation of functions with small circuit representation, largely improving on the speed performance of the tree-based method from the literature. Furthermore, we showed that our approach achieves state of the art latency on the sorting task, with a speed up of $\times 3$ compared to the approach of [13]. As a side result, we introduced a keyswitching key specific to packing TLWE ciphertexts into TRLWE ciphertexts with redundancy, which is of separate interest.

To go further, it would be interesting to compare our method to [1] which relies on the circuit bootstrapping not implemented natively in TFHElib. It would also be interesting to optimize the implementation of B-gates by introducing parallelism to harness the full potential of this technique.

Our method can also be of interest to research on homomorphic compilers such as Cingulata [6], Transpiler [17] and the Concrete compiler [10] as an alternative to binary computation.

References

[1] Loris Bergerat et al. *Parameter Optimization & Larger Precision for (T)FHE.* Cryptology ePrint Archive, Paper 2022/704. 2022.

[2] Adda-Akram Bendoukha, Pierre-Emmanuel Clet, Aymen Boudguiga, and Renaud Sirdey. “Optimized Stream-Cipher-Based Transciphering by Means of Functional-Bootstrapping”. In: *Data and Applications Security and Privacy XXXVII.* Springer Nature Switzerland, 2023, pp. 91–109. ISBN: 978-3-031-37586-6.

[3] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. “Simulating Homomorphic Evaluation of Deep Learning Predictions”. In: *Cyber Security Cryptography and Machine Learning.* Ed. by Shlomi Dolev, Danny Hendler, Sachin Lodha, and Moti Yung. Cham: Springer International Publishing, 2019, pp. 212–230.

[4] F. Bourse, M. Minelli, M. Minihold, and P. Paillier. “Fast Homomorphic Evaluation of Deep Discretized Neural Networks”. In: *Proceedings of CRYPTO 2018.* Springer, 2018.

[5] Florian Bourse, Olivier Sanders, and Jacques Traoré. *Improved Secure Integer Comparison via Homomorphic Encryption.* Cryptology ePrint Archive, Report 2019/427. 2019.

[6] Sergiu Carpov, Paul Dubrulle, and Renaud Sirdey. “Armadillo: A Compilation Chain for Privacy Preserving Applications”. In: *Proceedings of the 3rd International Workshop on Security in Cloud Computing.* SCC ’15. Singapore, Republic of Singapore: Association for Computing Machinery, 2015, pp. 13–19. ISBN: 9781450334471. DOI: `10.1145/2732516.2732520`. URL: `https://doi.org/10.1145/2732516.2732520`.

[7] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. “New Techniques for Multi-value Input Homomorphic Evaluation and Applications”. In: *Topics in Cryptology – CT-RSA 2019.* Ed. by Mitsuru Matsui. Cham: Springer International Publishing, 2019, pp. 106–126. ISBN: 978-3-030-12612-4.

[8] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. “Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds”. In: *Advances in Cryptology – ASIACRYPT 2016.* Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 3–33. ISBN: 978-3-662-53887-6.

[9] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. “TFHE: Fast Fully Homomorphic Encryption Over the Torus”. In: *Journal of Cryptology* 33 (Jan. 2020). DOI: `10.1007/s00145-019-09319-x`.

[10] Ilaria Chillotti, Marc Joye, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. *CONCRETE: Concrete Operates oN Ciphertexts Rapidly by Extending TfhE.* WAHC 2020 - 8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography. HAL id:⟨hal-03926650⟩. 2020.

[11] Ilaria Chillotti, Marc Joye, and Pascal Paillier. “Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks”. In: *Cyber Security Cryptography and Machine Learning.* Ed. by Shlomi

Dolev, Oded Margalit, Benny Pinkas, and Alexander Schwarzmann. Cham: Springer International Publishing, 2021, pp. 1–19. ISBN: 978-3-030-78086-9.

[12] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. *Improved Programmable Bootstrapping with Larger Precision and Efficient Arithmetic Circuits for TFHE.* Cryptology ePrint Archive, Report 2021/729. 2021.

[13] Antoine Choffrut, Rachid Guerraoui, Rafael Pinot, Renaud Sirdey, John Stephan, and Martin Zuber. *Practical Homomorphic Aggregation for Byzantine ML.* 2023. arXiv: `2309.05395 [cs.LG]`.

[14] Pierre-Emmanuel Clet, Aymen Boudguiga, Renaud Sirdey, and Martin Zuber. "ComBo: A Novel Functional Bootstrapping Method for Efficient Evaluation of Nonlinear Functions in the Encrypted Domain". In: *Progress in Cryptology - AFRICACRYPT 2023.* Springer, 2023. DOI: `10.1007/978-3-031-37679-5\_14`.

[15] Pierre-Emmanuel Clet, Martin Zuber, Aymen Boudguiga, Renaud Sirdey, and Cédric Gouy-Pailler. *Putting up the swiss army knife of homomorphic calculations by means of TFHE functional bootstrapping.* Cryptology ePrint Archive, Paper 2022/149. `https://eprint.iacr.org/2022/149`. 2022. URL: `https://eprint.iacr.org/2022/149`.

[16] Kelong Cong, Robin Geelen, Jiayi Kang, and Jeongeun Park. *Efficient and Secure k-NN Classification from Improved Data-Oblivious Programs and Homomorphic Encryption.* Cryptology ePrint Archive, Paper 2023/852. `https://eprint.iacr.org/2023/852`. 2023. URL: `https://eprint.iacr.org/2023/852`.

[17] Shruthi Gorantala, Rob Springer, Sean Purser-Haskell, William Lam, Royce Wilson, Asra Ali, Eric P. Astor, Itai Zukerman, Sam Ruth, Christoph Dibak, Phillipp Schoppmann, Sasha Kulankhina, Alain Forget, David Marn, Cameron Tew, Rafael Misoczki, Bernat Guillen, Xinyu Ye, Dennis Kraft, Damien Desfontaines, Aishe Krishnamurthy, Miguel Guevara, Irippuge Milinda Perera, Yurii Sushko, and Bryant Gipson. *A General Purpose Transpiler for Fully Homomorphic Encryption.* Cryptology ePrint Archive, Paper 2021/811. 2021. URL: `https://eprint.iacr.org/2021/811`.

[18] Antonio Guimarães, Edson Borin, and Diego F. Aranha. "Revisiting the functional bootstrap in TFHE". In: 2021 (Feb. 2021), pp. 229–253. DOI: `10.46586/tches.v2021.i2.229-253`.

[19] Ilia Iliashenko and Vincent Zucca. "Faster homomorphic comparison operations for BGV and BFV". In: *Proceedings on Privacy Enhancing Technologies* 2021 (2021), pp. 246–264. URL: `https://api.semanticscholar.org/CorpusID:232350948`.

[20] M. Izabachène, R. Sirdey, and M. Zuber. "Practical Fully Homomorphic Encryption for Fully Masked Neural Networks". In: *Cryptology and Network Security - 18th International Conference, CANS 2019, Proceedings.* Vol. 11829. Lecture Notes in Computer Science. Springer, 2019, pp. 24–36.

[21] Kamil Kluczniak and Leonard Schild. *FDFB: Full Domain Functional Bootstrapping Towards Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Report 2021/1135. `https://ia.cr/2021/1135`. 2021.

[22] Daphné Trama, Pierre-Emmanuel Clet, Aymen Boudguiga, and Renaud Sirdey. "A Homomorphic AES Evaluation in Less than 30 Seconds by Means of TFHE". In: *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. WAHC '23. Association for Computing Machinery, 2023, pp. 79–90. DOI: `10.1145/3605759.3625260`.

[23] Zhaomin Yang, Xiang Xie, Huajie Shen, Shiying Chen, and Jun Zhou. *TOTA: Fully Homomorphic Encryption with Smaller Parameters and Stronger Security*. Cryptology ePrint Archive, Report 2021/1347. `https://ia.cr/2021/1347`. 2021.

A KeySwitch with decomposition basis greater than 2

In this section, we detail the KeySwitch operation using any decomposition basis B_{KS}, usually a power of 2. We notably discuss multiple useful variants of the KeySwitch operation and detail the resulting noise growth of these variants. Besides, we improve the base-aware KeySwitch from [18] by proposing a technique requiring smaller keys.

We give in Algorithm 4 an adaptation of the KeySwitch from [9] for a decomposition basis greater than 2. This algorithm leads to the following noise formula:

$$\mathcal{V}_{KS} < R^2 \cdot V_c + \mathcal{E}_{KS}^{n,N} \text{ where } \mathcal{E}_{KS}^{n,N} = n\left(tN\vartheta_{KS} \cdot \left(\frac{B_{KS}}{2}\right)^2 + \frac{B_{KS}^{-2t}}{12}\right) \tag{8}$$

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

Algorithm 4 general TFHE KeySwitch

```
Input: p TLWE ciphertexts $\boldsymbol{c}^{(i)} = (\boldsymbol{a}^{(i)}, b^{(i)}) \in \text{TLWE}_s(m^{(i)})$ for $i \in [\![1,p]\!]$, a public $R$-
    Lipschitz morphism $f : \mathbb{T}^p \to \mathbb{T}_N[X]$, and $\text{KS}_{i,j} \in \text{T(R)LWE}_K(\frac{s_i}{B_{KS}^j})$ for $i \in [\![1,n]\!]$,
    $j \in [\![1,t]\!]$.
Output: A T(R)LWE sample $\boldsymbol{c} \in \text{T(R)LWE}_K(f(m^{(1)}, ..., m^{(p)}))$.
1: for $i \in [\![1,n]\!]$ do
2:     Let $a_i = f(a_i^{(1)}, ..., a_i^{(p)})$
3:     Let $\tilde{a}_i = \frac{\lceil B_{KS}^t \cdot a_i \rfloor}{B_{KS}^t}$
4:     Let $\tilde{a}_{i,j} \in \mathbb{Z}_N[X]$ with coefficients in $[\![-\frac{B_{KS}}{2}, \frac{B_{KS}}{2} - 1]\!]$ so that $\sum_{j=1}^{t} \tilde{a}_{i,j} \cdot B_{KS}^{-j} =$
   $\tilde{a}_i$
   return $(0, f(b^{(1)}, ..., b^{(p)})) - \sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j} \cdot \text{KS}_{i,j}$
```

When applying a TLWE to TLWE KeySwitch the analysis from [18] regarding the variance of the rounding part of the algorithm applies. Since $N = 1$ in this case, the noise formula then drops to:

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

Algorithm 5 further improves the noise bound for KeySwitch between TLWE ciphertexts by introducing a larger key. More specifically, the size of the key grows linearly with the chosen decomposition basis. The noise formula for this algorithm is:

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

Algorithm 5 TFHE KeySwitch between TLWE ciphertexts

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

```
1: for $i \in [\![1,n]\!]$ do
2:     Let $a_i = f(a_i^{(1)}, ..., a_i^{(p)})$
3:     Let $\tilde{a}_i = \frac{\lceil B_{KS}^t \cdot a_i \rfloor}{B_{KS}^t}$
4:     Let $\tilde{a}_{i,j} \in [\![0, B_{KS}-1]\!]$ so that $\sum_{j=1}^{t} \tilde{a}_{i,j} \cdot B_{KS}^{-j} = \tilde{a}_i$
   return $(0, f(b^{(1)}, ..., b^{(p)})) - \sum_{i=1}^{n} \sum_{j=1}^{t} \mathrm{KS}_{i,j,\tilde{a}_{i,j}}$
```

Finally, we show in Algorithm 6 how to compute a packing with redundancy with less noise compared to Algorithm 4 thanks to a specific keyswitch key. Algorithm 6 improves on the base aware Keyswitch of [18] by avoiding the multiplicative increase in size of the key and correcting the algorithm. This algorithm leads to the following noise formula:

$$\mathcal{V}_{KS} < V_c + \mathcal{E}_{KS}^{n,p} \tag{9}$$

B Tightness of Boole's inequality

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

$$1 - \prod_{i=0}^{m-1} \mathcal{P}_B(V_{\boldsymbol{c}_i}) \leq \sum_{i=0}^{m-1} \mathcal{F}_B(V_{\boldsymbol{c}_i}) \tag{10}$$

Algorithm 6 TFHE packing with redundancy

```
Input: $p$ TLWE ciphertexts $\boldsymbol{c}^{(i)} = (\boldsymbol{a}^{(i)}, b^{(i)}) \in \mathrm{TLWE}_s(m^{(i)})$ for $i \in [\![1, p]\!]$, and
    $\mathrm{KS}_{i,j} \in \mathrm{TRLWE}_K(\frac{s_i}{B_{KS}^j}\sum_{k=0}^{r-1} X^k)$ for $i \in [\![1, n]\!]$, $j \in [\![1, t]\!]$.
Output: A TRLWE sample $\boldsymbol{c} \in \mathrm{TRLWE}_K\left(\sum_{i=1}^{p}\left(m^{(i)}\sum_{j=0}^{r-1} X^{r\cdot(i-1)+j}\right)\right)$.
1: for $i \in [\![1, n]\!]$ do
2:     Let $a_i = \sum_{j=1}^{p} a_i^{(j)} X^{r\cdot(j-1)}$
3:     Let $\tilde{a}_i = \frac{\lceil B_{KS}^t \cdot a_i \rfloor}{B_{KS}^t}$
4:     Let $\tilde{a}_{i,j} \in \mathbb{Z}_N[X]$ with coefficients in $[\![-\frac{B_{KS}}{2}, \frac{B_{KS}}{2} - 1]\!]$ so that $\sum_{j=1}^{t} \tilde{a}_{i,j} \cdot B_{KS}^{-j} = \tilde{a}_i$
    return $\left(0, \sum_{i=1}^{p} b^{(i)} \sum_{j=0}^{r-1} X^{r\cdot(i-1)+j}\right) - \sum_{i=1}^{n}\sum_{j=1}^{t} \tilde{a}_{i,j} \cdot \mathrm{KS}_{i,j}$
```

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

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

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

$$\begin{aligned}1 - \prod_{i=0}^{m-1} \mathcal{P}_B(V_{\boldsymbol{c}_i}) &\geq \sum_{i=0}^{m-1} \mathcal{F}_B(V_{\boldsymbol{c}_i})(1 - \alpha(\frac{m-1}{2m} + 0.22\alpha)) \\ &\geq \sum_{i=0}^{m-1} \mathcal{F}_B(V_{\boldsymbol{c}_i})(1 - \epsilon)\end{aligned} \tag{11}$$

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

We combine Equations 10 and 11 to get

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

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