Putting up the swiss army knife of homomorphic calculations by means of TFHE functional bootstrapping

Pierre-Emmanuel Clet,
Martin Zuber, Aymen Boudguiga, Renaud Sirdey and Cédric Gouy-Pailler

pierre-emmanuel.clet@cea.fr

CEA-LIST, 91190 Gif-sur-Yvette

Abstract. In this work, we first propose a new full domain functional bootstrapping method with TFHE for evaluating any function of domain and codomain the real torus $\mathbb{T}$ by using a small number of bootstrappings. This result improves some aspects of previous approaches: like them, we allow for evaluating any functions, but with better precision. In addition, we develop efficient multiplication and addition over ciphertexts building on the digit-decomposition approach of [GBA21]. As a practical application, our results lead to an efficient implementation of ReLU, one of the most used activation functions in deep learning. The paper is concluded by extensive experimental results comparing each building block as well as their practical relevance and trade-offs.

Keywords: FHE · TFHE · functional bootstrapping

1 Introduction

Machine learning application to the analysis of private data, such as health or genomic data, has encouraged the use of homomorphic encryption for private inference or prediction with classification or regression algorithms where the ML models and/or their inputs are encrypted homomorphically [Xie+14; Cha+17; Cha+19; Bou+18; ZCS20b; ISZ19; ZS21]. Even training machine learning models with privacy guarantees on the training data has been investigated in the centralized [JA18; CKP19; Nan+19; Lou+20] and collaborative [Séb+21; Mad+21] settings. In practice, machine learning algorithms and especially neural networks require the computation of non-linear activation functions such as the sign, ReLU or sigmoid functions. Computing non-linear functions homomorphically remains challenging. For levelled homomorphic schemes such as BFV [Bra12; FV12] or CKKS [Che+17], non-linear functions have to be approximated by polynomials. However, the precision of this approximation differs with respect to the considered plaintext space (i.e., input range), approximation polynomial degree and its coefficients size, and has a direct impact on the multiplicative depth and parameters of the cryptosystem. The more precise is the approximation, the larger are the cryptosystem parameters and the slower is the computation. On the other hand, homomorphic encryption schemes having an efficient bootstrapping, such as TFHE [Chi+16; Chi+19] or FHEW [DM15], can be tweaked to encode functions via look-up table evaluations within their bootstrapping procedure. Hence, rather than being just used for refreshing ciphertexts (i.e., reducing their noise level), the bootstrapping becomes *functional* [BST19] or *programmable* [CJP21] by allowing the evaluation of arbitrary functions as a bonus. These capabilities results 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. It is also important to note that FHE cryptosystems can be hybridized, for example BFV ciphertexts can be efficiently (and homomorphically) turned into TFHE ones [Bou+20; ZCS20a]. As such, the building blocks discussed in this paper are of relevance also in the setting where the desired encrypted-domain calculation can be split into a preprocessing step more efficiently done using BFV (e.g. several dot product or distance computations) followed by a nonlinear postprocessing step (such as an activation function or an argmin) which can then be more conveniently performed by exploiting TFHE functional bootstrapping. In this work, we thus systematize and further investigate the capabilities of TFHE functional bootstrapping.

Contributions – In this paper, we review, unify and extend the capabilities of TFHE functional bootstrapping. We strive to present the main existing methods as well as new variants. We compare their relative accuracy and performance as well as discuss their main pros and cons. Indeed, on top of the extensions that we present, we aim for this paper to be a complete reference for anyone looking to get a view of the state of functional bootstrapping. As such, several methods for LUTs evaluation using functional bootstrapping are presented: the usual method using one bit of padding (described clearly in [CJP21]), two methods coming from recent papers that work without padding [KS21; Yan+21], one novel approach also working without padding, and a method using digit decomposition of the inputs in order to get an arbitrary large plaintext space (presented initially by Bourse et al., [BST19] and generalized later by Guimarães et al. [GBA21]). The first method encodes the plaintext space in $[0, \frac{1}{2}[$, i.e., the segment of the real torus $\mathbb{T}$ corresponding to the positive numbers. Meanwhile, the other methods use the full torus for encoding the plaintext space and propose various solutions to cope with the negacyclicity of TFHE bootstrapping when used for evaluating LUTs. A novel way we present to achieve this is to use several bootstrappings one after the other to cancel the negacyclicity of a single bootstrapping. In addition, we show how to reduce the noise resulting from the technique in [KS21]. Finally, the decomposition method allows working with larger plaintext spaces. Its main idea is to decompose each plaintext into small digits which allows keeping TFHE parameters small enough to lead to performance improvements. We generalize the chaining method of [GBA21] in order to compute *any* function with *any* chosen precision.

Related works – In 2016, the TFHE paper made a breakthrough by proposing an efficient bootstrapping for homomorphic gate computation. Then, Bourse et al., [Bou+18] and Izabachene et al., [ISZ19] used the same bootstrapping algorithm for extracting the (encrypted) sign of an encrypted input. Boura et al., [Bou+19] showed later that TFHE bootstrapping could be extended to support a wider class of functionalities. Indeed, 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. In particular, Kluczniak and Schild [KS21] and Yang et al., [Yan+21] proposed two methods that take into consideration the negacyclicity of the cyclotomic polynomial used within the bootstrapping, for encoding look-up tables over the full real torus $\mathbb{T}$. Meanwhile, Guimarães et al., [GBA21] extended the ideas in Bourse et al., [BST19] to support the evaluation of certain activation functions such as the sigmoid. One last method, presented in Chillotti et al., [Chi+21] achieves a functional bootstrapping over the full torus using a BFV type multiplication.

Paper organization – The remainder of this paper is organized as follows. Section 2 reviews TFHE building blocks. Section 3 describes the functional bootstrapping idea coming from the TFHE gate bootstrapping. Sections 4 and 6 detail several methods, including ours, for the intricate Look-Up Tables (LUTs) encoding via the functional bootstrapping. Indeed, section 4 describes methods for LUTs evaluation when having a unique ciphertext as input. Meanwhile, section 6 considers the case where LUTs are evaluated over several ciphertexts encrypting

separately the digits of a large plaintext. The Section 5 details the formulas relative to the noise variance and the probability of error for each bootstrapping. Finally, section 7 gives unitary results comparing these methods for LUTs evaluation over encrypted data.

2 TFHE

2.1 Notations

In the upcoming sections, we denote vectors by bold letters and so, each vector $\boldsymbol{x}$ of n elements is described as: $\boldsymbol{x}=(x_1,...,x_n)$. $\langle \boldsymbol{x},\boldsymbol{y}\rangle$ 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 $\mathcal{M}_{m,n}(\mathbb{K})$.

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. That is, multiplication by scalars from $\mathbb{Z}$ is well-defined over $\mathbb{T}$. $\mathbb{T}_N[X]$ 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 $\mathbb{B}_N[X]=\mathbb{B}[X]/(X^N+1)$ ($\mathbb{B}=\{0,1\}$). 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 $\mathcal{R}$.

$x \xleftarrow{\$} \mathbb{K}$ 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 $\frac{1}{2}(erf(\frac{b-\mu}{\sqrt{2}\sigma}) - erf(\frac{a-\mu}{\sqrt{2}\sigma}))$, where erf is Gauss error function; $erf(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}$. If $\mu=0$, we will denote $P(-a\leq x\leq a)=erf(\frac{a}{\sqrt{2}\sigma})$ by (a,σ^2). The same result and notation apply for $x \xleftarrow{\mathcal{N}(0,\sigma^2)} \mathbb{T}$ as long as the distribution is concentrated as described in [Chi+19].

Given a function $f:\mathbb{T}\to\mathbb{T}$ and an integer k, we define $\mathrm{LUT}_k(f)$ to be the Look-Up Table defined by the set of k pairs $\left(i, f\left(\frac{i}{k}\right)\right)$ for $i\in[\![0,k-1]\!]$. We will write $\mathrm{LUT}(f)$ when the value of k is tacit.

Given a function $f:\mathbb{T}\to\mathbb{T}$ and an integer $k\leq N$, we define a polynomial $P_{f,k}\in\mathbb{T}_N[X]$ of degree N as: $P_{f,k}=\sum_{i=0}^{N-1} f\left(\frac{\lfloor \frac{k\cdot i}{2N}\rfloor}{k}\right)\cdot X^i$. If k is a divisor of $2N$, $P_{f,k}$ can be written as $P_{f,k}=\sum_{i=0}^{\frac{k}{2}-1}\sum_{j=0}^{\frac{2N}{k}-1} f(\frac{i}{k})\cdot X^{\frac{2N}{k}\cdot i+j}$. For simplicity sake, we will write P_f instead of $P_{f,k}$ when the value k is tacit.

2.2 TFHE Structures

The TFHE encryption scheme was proposed in 2016 [Chi+16]. It improves the FHEW cryptosystem [DM15] and introduces the TLWE problem as an adaptation of the LWE problem to $\mathbb{T}$. It was updated later in [Chi+17] and both works were recently unified in [Chi+19]. The TFHE scheme is implemented as the TFHE library [Chi+]. 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 $e\xleftarrow{\mathcal{N}(0,\sigma^2)}\mathbb{T}$. Then, $(\boldsymbol{a},b)$ is a fresh encryption of 0.
- **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. In this case, $(\boldsymbol{a},b)$ is a fresh encryption of 0.

The TRLWE decision problem consists of distinguishing TRLWE samples from random samples in $\mathbb{T}_N[X]^k \times \mathbb{T}_N[X]$. Meanwhile, the TRLWE search problem consists in finding the private polynomial $\boldsymbol{s}$ given arbitrarily many TRLWE samples. When $N=1$ and k is large, the TRLWE decision and search problems become the TLWE decision and search problems, respectively.

Let $\mathcal{M} \subset \mathbb{T}_N[X]$ (or $\mathcal{M} \subset \mathbb{T}$) be the discrete message space[1]. To encrypt a message $m \in \mathcal{M} \subset \mathbb{T}_N[X]$, we add $(\mathbf{0},m) \in \mathbb{T}_N[X]^k \times \mathbb{T}_N[X]$ to a TRLWE sample encrypting 0 (or to a TLWE sample of 0 if $\mathcal{M} \subset \mathbb{T}$). In the following, we refer to an encryption of m with the secret key $\boldsymbol{s}$ as a T(R)LWE ciphertext noted $\boldsymbol{c} \in \mathrm{T(R)LWE}_{\boldsymbol{s}}(m)$.

To decrypt a sample $\boldsymbol{c} \in \mathrm{T(R)LWE}_{\boldsymbol{s}}(m)$, 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 (yet high enough to ensure security), the decryption will be accurate.

- **TRGSW Sample:** is a vector of l TRLWE samples encrypting 0. To encrypt a message $m \in \mathcal{R}$, we add $m \cdot H$ to a TRGSW sample of 0, where H is a gadget matrix[2] using an integer B_g as a base for its decomposition. Chilotti et al., [Chi+19] 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 [Chi+19]). It yields an encryption of $m_a \cdot m_b$ i.e., a TRLWE sample $\boldsymbol{c} \in \mathrm{TRLWE}_{\boldsymbol{s}}(m_a \cdot m_b)$. Otherwise, the external product allows also to compute a controlled MUX gate (CMUX) where the selector is $C_b \in \mathrm{TRGSW}_{\boldsymbol{s}}(b), b \in \{0,1\}$, and the inputs are $\boldsymbol{c}_0 \in \mathrm{TRLWE}_{\boldsymbol{s}}(m_0)$ and $\boldsymbol{c}_1 \in \mathrm{TRLWE}_{\boldsymbol{s}}(m_1)$.

2.3 TFHE Bootstrapping

TFHE bootstrapping relies mainly on three building blocks:

- **Blind Rotate:** rotates a plaintext polynomial encrypted as a TRLWE ciphertext by an encrypted position. It takes as inputs: a TRLWE ciphertext $\boldsymbol{c} \in \mathrm{TRLWE}_{\boldsymbol{k}}(m)$, a vector $(a_1,...,a_n,a_{n+1}=b)$ where $\forall i,\ a_i \in \mathbb{Z}_{2N}$, and n TRGSW ciphertexts encrypting $(s_1,...,s_n)$ where $\forall i,\ s_i \in \mathbb{B}$. It returns a TRLWE ciphertext $\boldsymbol{c'} \in \mathrm{TRLWE}_{\boldsymbol{k}}(X^{\langle \boldsymbol{a},\boldsymbol{s} \rangle - b} \cdot m)$. In this paper, we will refer to this algorithm by BlindRotate. With respect to independence heuristic[3] stated in [Chi+19], the variance $\mathcal{V}_{BR}$ of the resulting noise after a BlindRotate satisfies the formula:

$$\mathcal{V}_{BR} < V_c + n((k+1)\ell N(\frac{B_g}{2})^2 \vartheta_{BK} + \frac{(1+kN)}{4 \cdot B_g^{2l}}),$$

where V_c is the variance of the noise of the input ciphertext c, and ϑ_{BK} is the the variance of the error of the bootstrapping key. In the following, we define:

$$\mathcal{E}_{BS} = n((k+1)\ell N(\frac{B_g}{2})^2 \vartheta_{BK} + \frac{(1+kN)}{4 \cdot B_g^{2l}})$$

- **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

[1] 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 $\{0,0.25,0.5,0.75\}$.

[2] Refer to Definition 3.6 and Lemma 3.7 in TFHE paper [Chi+19] for more information about the gadget matrix H.

[3] 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.

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 $\boldsymbol{c'} \in \mathrm{T(R)LWE}_{\boldsymbol{s}}(f(m_1,...,m_p))$, where $f()$ is a public linear morphism from $\mathbb{T}^p$ to $\mathbb{T}_N[X]$. This algorithm uses 2 specific parameters, namely B_{KS} which is used as a base to decompose some coefficients, and t which gives the precision of the decomposition. Note that functional keyswitching serves at changing encryption keys and parameters. In this paper, we will refer to this algorithm by KeySwitch. As stated in [Chi+19; GBA21], the variance $\mathcal{V}_{KS}$ of the resulting noise after KeySwitch follows the formula:

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

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 case, $R=1$. In the following, we define:

$$\mathcal{E}_{KS} = n(tN\vartheta_{KS} + \frac{B_{KS}^{-2t}}{12})$$

TFHE comes with two bootstrapping algorithms. The first one is the gate bootstrapping. It aims at reducing 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. The binary nature of inputs/outputs of this algorithm is not due to inherent limitations of the TFHE scheme but rather to the fact that the authors of the paper were building a bitwise set of operators for which this bootstrapping operation was perfectly fitted.

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[4]. 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 can be summarized as a BlindRotate over a noiseless TRLWE followed by a Keyswitch, the bootstrapping noise ($\mathcal{V}_{BS}$) satisfies: $\mathcal{V}_{BS} < \mathcal{E}_{BS} + \mathcal{E}_{KS}$.

Algorithm 1 TFHE gate bootstrapping [Chi+19]

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: $ACC \leftarrow \mathsf{BlindRotate}((\boldsymbol{0}, testv), (\bar{a}_1, ..., \bar{a}_n, \bar{b}), (BK_1, ..., BK_n))$
5: $\overline{\boldsymbol{c}} = (\boldsymbol{0}, \hat{m}) + \mathsf{SampleExtract}(ACC)$
6: return $\mathsf{KeySwitch}_{\boldsymbol{s'} \to \boldsymbol{s}}(\overline{\boldsymbol{c}})$

[4] Further details on the proper bound of the noise are given in Section 5.

TFHE specifies a second type of bootstrapping called *circuit bootstrapping*. It converts TLWE samples into TRGSW samples, and serves mainly for TFHE use in a levelled manner. This type of bootstrapping will not be discussed further in this paper.

3 TFHE Functional Bootstrapping

3.1 Encoding and Decoding

Our goal is to build an homomorphic LUT for any function $f : \mathbb{Z}_p \to \mathbb{Z}_p$ for any integer p. As we are using TFHE, every message from $\mathbb{Z}_p$ has to be encoded in $\mathbb{T}$. To that end, we use the encoding function:

$$\iota : \begin{array}{ccc} \mathbb{Z}_p & \to & \mathbb{T} \\ k & \mapsto & \frac{k}{p} + \frac{1}{2p} \end{array}$$

and its corresponding decoding function:

$$\omega : \begin{array}{ccc} \mathbb{T} & \to & \mathbb{Z}_p \\ x & \mapsto & \lfloor x \cdot p \rfloor \end{array}$$

The choice of such an encoding function is further detailed in Section A.

Finally, we specify a torus-to-torus function $f_{\mathbb{T}}$ to get $f = D_p \circ f_{\mathbb{T}} \circ E_p$.

$$\begin{array}{ccc} \mathbb{Z}_p & \xrightarrow{f=\omega \circ f_{\mathbb{T}} \circ \iota} & \mathbb{Z}_p \\ \iota \downarrow & & \uparrow \omega \\ \mathbb{T} & \underset{f_{\mathbb{T}}}{\longrightarrow} & \mathbb{T} \end{array}$$

Since the function $f_{\mathbb{T}} = E_p \circ f \circ D_p$ makes the diagram commutative, we discuss in the following sections several ways for building any Look-Up Table (LUT) for such function $f_{\mathbb{T}}$ for $p \leq N$.

We will use $m^{(p)}$ to refer to a message in $\mathbb{Z}_p$, and m to refer to $E_p(m^{(p)})$. Then, m is the representation of $m^{(p)}$ in $\mathbb{T}$ after discretization.

3.2 Functional Bootstrapping Idea

The original bootstrapping algorithm from [Chi+16] had already all the tools to implement a LUT of any negacyclic function[5]. In particular, TFHE is well-suited for $\frac{1}{2}$-antiperiodic function, 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 $(X^N + 1)$. Note that $X^{\alpha+N} \equiv -X^{\alpha} mod[X^N + 1]$ and it allows encoding negacylic functions as explained in the upcoming sections.

Boura et al., [Bou+19] were the first to use the term *functional bootstrapping* for TFHE. They describe how TFHE bootstrapping computes a sign function. In addition, they state that bootstrapping can be used to build a Rectified Linear Unit (ReLU). However, they do not delve into the details of how to implement the ReLU in practice[6].

Algorithm 2 Sign extraction with bootstrapping

Input: a constant $\mu \in \mathbb{T}$, a TLWE sample $\boldsymbol{c} = (\boldsymbol{a}, b) \in \text{TLWE}_{\boldsymbol{s}}(m)$ with $m \in \mathbb{T}$, a bootstrapping key $BK_{\boldsymbol{s}\to\boldsymbol{s'}} = (BK_i \in \text{TRGSW}_{S'}(s_i))_{i\in[\![1,n]\!]}$ where S' is the TRLWE interpretation of a secret key $\boldsymbol{s'}$

Output: a TLWE sample $\overline{\boldsymbol{c}} \in \text{TLWE}_{\boldsymbol{s}}(\mu.sign(m))$

1: Let $\bar{b} = \lfloor 2Nb \rceil$ and $\bar{a}_i = \lfloor 2Na_i \rceil \in \mathbb{Z}, \forall i \in [\![1,n]\!]$
2: Let $testv := (1 + X + \cdots + X^{N-1}) \cdot \mu \in \mathbb{T}_N[X]$
3: $ACC \leftarrow \mathsf{BlindRotate}((\mathbf{0}, testv), (\bar{a}_1, ..., \bar{a}_n, \bar{b}), (BK_1, ..., BK_n))$
4: $\overline{\boldsymbol{c}} = \mathsf{SampleExtract}(ACC)$
5: return $\mathsf{KeySwitch}_{\boldsymbol{s'}\to\boldsymbol{s}}(\overline{\boldsymbol{c}})$

Algorithm 2 describes a sign computation with the TFHE bootstrapping. It returns μ if m is positive (i.e., $m \in [0, \frac{1}{2}[$), and $-\mu$ if m is negative.

When we look at the building blocks of Algorithm 2, we notice that there is some leeway to build more complex functions just by changing the coefficients of the test polynomial $testv$.

Indeed, if we consider $t = \sum_{i=0}^{N-1} t_i \cdot X^i$ where $t_i \in \mathbb{T}$ and $t^*(x)$ is the function:

$$t^* : \begin{array}{ccc} [\![-N, N-1]\!] & \to & \mathbb{T} \\ i & \mapsto & \begin{cases} t_i & \text{if } i \in [\![0, N[\![\\ -t_{i+N} & \text{if } i \in [\![-N, 0[\![\end{cases} \end{array}$$

then, the output of the bootstrapping of a TLWE ciphertext $[x]_{\mathbb{T}} = (\boldsymbol{a}, b)$ with the test polynomial $testv = t$ is $[t^*(\phi(\bar{\boldsymbol{a}}, \bar{b}))]_{\mathbb{T}}$, where $(\bar{\boldsymbol{a}}, \bar{b})$ is the rescaled version of $(\boldsymbol{a}, b)$ in $\mathbb{Z}_{2N}$ (line 1 of Algorithm 2).

To prove this result, we first remind that for any positive integer i s.t. $0 \leq i < N$, we have:

$$testv.X^{-i} = t_i + \cdots - t_0 X^{N-i} - \cdots - t_{i-1} X^{N-1} \mod [X^N + 1] \tag{1}$$

Then, we notice that $\mathsf{BlindRotate}$ (line 3 of Algorithm 2) computes $testv \cdot X^{-\phi(\bar{\boldsymbol{a}}, \bar{b})}$. Therefore, we obtain the following results using equation (1):

- if $\phi(\bar{\boldsymbol{a}}, \bar{b}) \in [\![0, N[\![$, the constant term of $testv \cdot X^{-\phi(\bar{\boldsymbol{a}}, \bar{b})}$ is $t_{\phi(\bar{\boldsymbol{a}}, \bar{b})}$.
- if $\phi(\bar{\boldsymbol{a}}, \bar{b}) \in [\![-N, 0[\![$, we have:

  $testv \cdot X^{-\phi(\bar{\boldsymbol{a}}, \bar{b})} = -testv \cdot X^{-\phi(\bar{\boldsymbol{a}}, \bar{b}) - N} mod [X^N + 1]$

  with $(\phi(\bar{\boldsymbol{a}}, \bar{b}) + N) \in [\![0, N[\![$. So, the constant term of $testv \cdot X^{-\phi(\bar{\boldsymbol{a}}, \bar{b})}$ is $-t_{\phi(\bar{\boldsymbol{a}}, \bar{b}) + N}$.

All that remains for the bootstrapping algorithm is extracting the previous constant term (in line 4) and keyswitching (in line 5) to get the TLWE sample $[t^*(\phi(\bar{\boldsymbol{a}}, \bar{b}))]_{\mathbb{T}}$.

We can use this to build a discretized function evaluation as follows. Let $f : \mathbb{Z}_p \to \mathbb{Z}_p$ be any negacyclic function over $\mathbb{Z}_p$ and $f_{\mathbb{T}} = E_p \circ f \circ D_p$. We call $\tilde{f}$ the well-defined function $f_{\mathbb{T}} \circ E_{2N}$ that satisfies:

$$\tilde{f} : \begin{array}{ccc} [\![-N, N-1]\!] & \to & \mathbb{T} \\ x & \mapsto & \begin{cases} f_{\mathbb{T}}(\frac{x}{2N}) & \text{if } x \in [\![0, N[\![\\ -f_{\mathbb{T}}(\frac{x+N}{2N}) & \text{if } x \in [\![-N, 0[\![\end{cases} \end{array} \tag{2}$$

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

[6] The article does only mention that the function $2\times\mathsf{ReLU}$ can be built from an absolute value function but does not explain how to divide by two to get the ReLU result.

Let P_f be the polynomial $P_f = \sum_{i=0}^{N-1} \tilde{f}(i) \cdot X^i$. Now, if we apply the bootstrapping Algorithm 2 to a TLWE ciphertext $[m]_{\mathbb{T}} = (\boldsymbol{a}, b)$ with $m^{(p)} \in \mathbb{Z}_p$ and $testv = P_f$, it outputs $[\tilde{f}(\phi(\bar{\boldsymbol{a}}, \bar{b}))]_{\mathbb{T}}$. That is, Algorithm 2 allows the encoding of the function f as long as $\frac{\phi(\bar{\boldsymbol{a}}, \bar{b})}{2N} = m + e'$, for some e' small enough. Further details on the variance of e' and the probability of the bootstrapping error are given in Section 5.

4 Look-Up-Tables over a Single Ciphertext

In Section 3.2, we demonstrated that functional bootstrapping allows for the computation of $\mathrm{LUT}(f)$ for any negacyclic function f. In this section, we describe 4 different ways to build homomorphic LUTs for *any* function (i.e., not necessarily negacyclic ones). We present 3 solutions from the state of the art [CJP21; KS21; Yan+21] in Sections 4.1, 4.2 and 4.3, and one that is novel to our work in Section 4.4. In addition, we describe a method for reducing the noise of the functional bootstrapping presented in [KS21].

As in Section 3.1, we call $f_{\mathbb{T}} : \mathbb{T} \to \mathbb{T}$ the function used to build our homomorphic LUT, and $f : \mathbb{Z}_p \to \mathbb{Z}_p$ its corresponding function over the input and output space $\mathbb{Z}_p$. Considering that the LUTs are actually polynomials, $p \leq 2N$.

4.1 Partial Domain Functional Bootstrapping – Half-Torus

This method avoids the negacyclic restriction of functional bootstrapping by encoding values only on $[0, \frac{1}{2}[$ (i.e., half of the torus). Let's consider the test polynomial to be P_h for a given negacyclic function h. The output of the bootstrapping operation is given by Equation 2:

$$\tilde{h} : \begin{array}{ccc} [\![-N, N-1]\!] & \to & \mathbb{T} \\ x & \mapsto & \begin{cases} h(\frac{x}{2N}) & \text{if } x \in [\![0, N[\![\\ -h(\frac{x+N}{2N}) & \text{if } x \in [\![-N, 0[\![\end{cases} \end{array}$$

If we restrict $\tilde{h}$ domain to $[\![0, N[\![$, we ensure that $\tilde{h}$ is just a LUT based on h, where h has not to be negacyclic. That is, we obtain a method to evaluate a LUT in a *single* bootstrapping. However, we have to encode the plaintext space over a smaller portion of the torus $\mathbb{T}$, therefore increasing the relative noise introduced by the TFHE encryption process. Hence, the overall result will be less accurate.

4.2 Full Domain Functional Bootstrapping – FDFB

Kluczniak and Schild [KS21] proposed this method to evaluate encrypted LUTs of domain the whole torus $\mathbb{T}$. Let's consider a TLWE ciphertext $[m]_{\mathbb{T}}$ given a message $m^{(p)} \in \mathbb{Z}_p$. We denote by g the function:

$$g : \begin{array}{ccc} \mathbb{T} & \to & \mathbb{T} \\ x & \mapsto & -f_{\mathbb{T}}(x + \frac{1}{2}) \end{array}$$

We denote by $q \in \mathbb{N}^*$ the smallest integer such that $q \cdot (P_f - P_g)$ is a polynomial with coefficients in $\mathbb{Z}$. Then, we define $P_1 = q \cdot P_f$ and $P_2 = q \cdot P_g$. Note that the coefficients of $P_f - P_g$ are multiples of $\frac{1}{p}$ in $\mathbb{T}$, where $\mathbb{T}$ corresponds to $[-\frac{1}{2}, \frac{1}{2}[$. Thus q is a divisor of p, and $P_2 - P_1$ has coefficients of norm lower or equal to $\frac{q}{2}$.

We define the Heaviside function H as:

$$H : x \mapsto \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

H can be expressed using the sign function as follows: $H(x) = \frac{\text{sign}(x)+1}{2}$.

First, we compute $[E_q(H(m))]_{\mathbb{T}}$ with only one bootstrapping (using Algorithm 2) and deduce $[E_q((1-H)(m))]_{\mathbb{T}} = (\mathbf{0}, \frac{1}{q}) - [E_q(H(m))]_{\mathbb{T}}$. Then, we make a keyswitch to transform the TLWE sample $[E_q((1-H)(m))]_{\mathbb{T}}$ into a TRLWE sample $[E_q((1-H)(m))]_{\mathbb{T}_N[X]}$. Finally, we define:

$$\boldsymbol{c}_{\text{LUT}} = (P_2 - P_1) \cdot [E_q((1-H)(m))]_{\mathbb{T}_N[X]} + (\mathbf{0}, P_f)$$

such that:

$$\boldsymbol{c}_{\text{LUT}} = \begin{cases} [P_f]_{\mathbb{T}_N[X]} & \text{if } m \geq 0 \\ [P_g]_{\mathbb{T}_N[X]} & \text{if } m < 0 \end{cases}$$

Note that depending on the sign of m, $\boldsymbol{c}_{\text{LUT}}$ is a TRLWE encryption of P_f or P_g, the test polynomials of f or g, respectively. Indeed, after a private functional bootstrapping of $[E_p(m)]_{\mathbb{T}}$ using $\boldsymbol{c}_{\text{LUT}}$ as a test polynomial, we obtain $[f_{\mathbb{T}}(m)]_{\mathbb{T}}$. This functional bootstrapping requires 2 BlindRotate during the bootstrapping: one to compute the Heaviside function and the other to apply the encrypted LUT.

The factorization idea presented in Carpov et al., [CIM19] (and described in Section B.2), allows us to reduce the noise of $\boldsymbol{c}_{\text{LUT}}$. To that end, we replace the polynomials P_f and P_g by $v_f = (1-X) \cdot P_f$ and $v_g = (1-X) \cdot P_g$, respectively. We denote by $q' \in \mathbb{N}^*$ the smallest integer such that $q' \cdot (v_f - v_g)$ is a polynomial with coefficients in $\mathbb{Z}$. Note that since $q \cdot (1-X) \cdot (P_f - P_g) = (1-X) \cdot (q \cdot (P_f - P_g))$ has coefficients in $\mathbb{Z}$, we ensure that $q' \leq q$. Then, we define $v_1 = q' \cdot v_f$ and $v_2 = q' \cdot v_g$. We get that $v_2 - v_1$ has coefficients in $\mathbb{Z}$ of norm lower or equal to q'. Finally, we compute a TRLWE encryption of $\sum_{i=0}^{N-1} X^i \cdot E_{2 \cdot q'}((1-H)(m))$ from the TLWE sample $[E_{2 \cdot q'}((1-H)(m))]_{\mathbb{T}}$, by applying a KeySwitch. Thus, we get:

$$\boldsymbol{c}_{\text{LUT}} = (v_2 - v_1) \cdot [\sum_{i=0}^{N-1} X^i \cdot E_{2 \cdot q'}((1-H)(m))]_{\mathbb{T}_N[X]} + (\mathbf{0}, P_f)$$

such that:

$$\boldsymbol{c}_{\text{LUT}} = \begin{cases} [P_f]_{\mathbb{T}_N[X]} & \text{if } m \geq 0 \\ [P_g]_{\mathbb{T}_N[X]} & \text{if } m < 0 \end{cases}$$

4.3 Full Domain Functional Bootstrapping – TOTA

Yan et al., [Yan+21] proposed this method to evaluate arbitrary functions over the torus using a functional bootstrapping. Let's consider a ciphertext $[m_1]_{\mathbb{T}} = (\boldsymbol{a}, b = <\boldsymbol{a}, \boldsymbol{s}> + m_1 + e)$. Then, by dividing each coefficient of this ciphertext by 2, we get a ciphertext $[m_2]_{\mathbb{T}} = (\frac{\boldsymbol{a}}{2}, <\frac{\boldsymbol{a}}{2}, \boldsymbol{s}> + m_2 + \frac{e}{2})$ where $m_2 = \frac{m_1}{2} + \frac{k}{2}$ with $k \in \{0,1\}$ and $\frac{m_1}{2} \in [0, \frac{1}{2}[$. Using the original bootstrapping algorithm, we compute $[\frac{\text{sign}(m_2)}{4}]_{\mathbb{T}}$ an encryption of $\frac{\text{sign}(m_2)}{4} = \begin{cases} \frac{1}{4} & \text{if } k = 0 \\ -\frac{1}{4} & \text{if } k = 1 \end{cases}$. Then, $[m_2]_{\mathbb{T}} - [\frac{\text{sign}(m_2)}{4}]_{\mathbb{T}} + (\mathbf{0}, \frac{1}{4})$ is an encryption of $\frac{m_1}{2}$.

For any function $f_{\mathbb{T}}$, let's define $f_{(2)}$ such that $f_{(2)}(x) = f_{\mathbb{T}}(2x)$. Since $\frac{m_1}{2} \in [0, \frac{1}{2}[$, we can compute $f_{(2)}(\frac{m_1}{2})$ with a single bootstrapping using the partial domain solution from 4.1, and $f_{(2)}(\frac{m_1}{2}) = f_{\mathbb{T}}(m_1)$.

Thus, this method allows computing any function with only 2 bootstrappings. Keep in mind that the torus is actually discretized, so some noise and some loss of precision are introduced after dividing by 2 due to the rounding of the coefficients.

4.4 Full Domain Functional Bootstrapping with Composition

In this section, we present a novel method to compute any function using the full (discretized) torus as plaintext space. We will consider here that p is even and fixed. Otherwise we can simply add one unused element to get back to this case.

Pseudo odd functions: We call pseudo odd function a function f that verifies $\forall x \in \mathbb{T}, f(-x-\frac{1}{p}) = -f(x)$.

We note $f_{\mathbb{T}}$ a pseudo odd function over the discretized torus, and $\lfloor x \rceil_{\mathcal{M}}$ the rounding function $: x \mapsto \frac{\lfloor p \cdot x \rfloor}{p}$ that rounds down values over the torus to a multiple of $\frac{1}{p}$. Since p is even, we have: $\forall x \in \mathbb{T}, x+\frac{1}{2} = x+\frac{1}{2} mod [1]$.

Let h_1 be the following function:

$$h_1 : \begin{array}{ccc} [0,\frac{1}{2}[& \to & \mathbb{T} \\ x & \mapsto & x+\frac{1}{2p} \end{array}$$

Then, we define a functional bootstrapping with an output function $\tilde{h}_1$ as:

$$\tilde{h}_1 : x \mapsto \begin{cases} \lfloor \frac{x}{2N} \rceil_{\mathcal{M}} + \frac{1}{2p} & \text{if } x \in [\![0,N[\![\\ -\lfloor \frac{x}{2N} \rceil_{\mathcal{M}} - \frac{1}{2} - \frac{1}{2p} & \text{if } x \in [\![-N,0[\![\end{cases}$$

From this function we define $G_{\text{odd}} : \; x \mapsto 2N(\tilde{h}_1(x) - \frac{1}{2 \cdot p})$ so that:

$$G_{\text{odd}} : x \mapsto \begin{cases} 2N \cdot \lfloor \frac{x}{2N} \rceil_{\mathcal{M}} & \text{if } x \in [\![0,N[\![\\ -2N \cdot \lfloor \frac{x}{2N} \rceil_{\mathcal{M}} - N - \frac{2N}{p} & \text{if } x \in [\![-N,0[\![\end{cases}$$

Note that $G_{\text{odd}}(0) = 0$ and $G_{\text{odd}}(N-1) = N - \frac{2N}{p}$. So, if $x \in [\![0,N[\![, G_{\text{odd}}(x) \in [\![0,N[\![$. We notice similarly that $G_{\text{odd}}(-N) = -\frac{2N}{p}$ and $G_{\text{odd}}(-1) = \frac{2N}{p} - N - \frac{2N}{p} = -N$. So, if $x \in [\![-N,0[\![, G_{\text{odd}}(x) \in [\![-N,0[\![$.

Finally, we define $\tilde{f}_{\mathbb{T}}$ as:

$$\tilde{f}_{\mathbb{T}} : x \mapsto \begin{cases} f_{\mathbb{T}}\left(\frac{x}{2N}\right) & \text{if } x \in [\![0,N[\![\\ -f_{\mathbb{T}}\left(\frac{x+N}{2N}\right) & \text{if } x \in [\![-N,0[\![\end{cases}$$

We can then compose $\tilde{f}_{\mathbb{T}}$ with G_{odd}.

$$\tilde{f}_{\mathbb{T}} \circ G_h : x \mapsto \begin{cases} f_{\mathbb{T}}\left(\lfloor \frac{x}{2N} \rceil_{\mathcal{M}}\right) & \text{if } x \in [\![0,N[\![\\ -f_{\mathbb{T}}\left(-\lfloor \frac{x}{2N} \rceil_{\mathcal{M}} - \frac{1}{p}\right) & \text{if } x \in [\![-N,0[\![\end{cases}$$

Considering that $f_{\mathbb{T}}$ is pseudo odd, we get:

$$\forall x \in [\![0,2N-1]\!], \; \tilde{f}_{\mathbb{T}} \circ G_{\text{odd}}(x) = f_{\mathbb{T}}(\lfloor \frac{x}{2N} \rceil_{\mathcal{M}})$$

Therefore $\tilde{f}_{\mathbb{T}} \circ G_{\text{odd}}$ evaluates a LUT based on $f_{\mathbb{T}}$ for the whole discretized torus.

Pseudo even functions: We call pseudo even function a function f that verifies $\forall x \in \mathbb{T}, f(-x-\frac{1}{p}) = f(x)$.

Let $f_{\mathbb{T}}$ be a pseudo even function over the discretized torus.

We set h_2 as:

$$h_2 : \begin{array}{ccc} [0,\frac{1}{2}[& \to & \mathbb{T} \\ x & \mapsto & \lfloor x \rceil_{\mathcal{M}} + \frac{1}{4} + \frac{1}{2p} \end{array}$$

Then, we define a functional bootstrapping with an output function $\tilde{h}_2$ as:

$$\tilde{h}_2 \colon x \mapsto \begin{cases} \lfloor \frac{x}{2N} \rceil_{\mathcal{M}} + \frac{1}{4} + \frac{1}{2p} & \text{if } x \in [\![0, N[\![\\ -\lfloor \frac{x}{2N} \rceil_{\mathcal{M}} + \frac{1}{4} - \frac{1}{2p} & \text{if } x \in [\![-N, 0[\![\end{cases}$$

Finally, we compose $\tilde{f}_{\mathbb{T}}$ with $G_{\text{even}} : x \mapsto 2N(\tilde{h}_2(x) - \frac{1}{4} - \frac{1}{2 \cdot p})$ which sends every values in the positive half of the torus.

$$\tilde{f}_{\mathbb{T}} \circ G_{\text{even}} \colon x \mapsto \begin{cases} f_{\mathbb{T}}\left(\lfloor \frac{x}{2N} \rceil_{\mathcal{M}}\right) & \text{if } x \in [\![0, N[\![\\ f_{\mathbb{T}}\left(-\lfloor \frac{x}{2N} \rceil_{\mathcal{M}} - \frac{1}{p}\right) & \text{if } x \in [\![-N, 0[\![\end{cases}$$

Considering that $f_{\mathbb{T}}$ is pseudo even, we get:

$$\forall x \in \mathbb{T}, \tilde{f}_{\mathbb{T}} \circ G_{\text{even}}(x) = f_{\mathbb{T}}(\lfloor \frac{x}{2N} \rceil_{\mathcal{M}})$$

Therefore, $\tilde{f}_{\mathbb{T}} \circ G_{\text{even}}$ is a LUT based on $f_{\mathbb{T}}$ over the whole discretized torus.

Any function: Any function $f_{\mathbb{T}}$ can be written as a sum of a pseudo even function and a pseudo odd function: $f_{\mathbb{T}}(x) = \frac{f_{\mathbb{T}}(x) + f_{\mathbb{T}}(-x - \frac{1}{p})}{2} + \frac{f_{\mathbb{T}}(x) - f_{\mathbb{T}}(-x - \frac{1}{p})}{2}$. We already demonstrated that we can build a LUT for any pseudo odd or pseudo even function with at most 2 functional bootstrapping. Thus, we can build a LUT for any function with at most 4 functional bootstrapping. In addition, odd and even functions can be computed in a similar way to their pseudo equivalent with only 2 bootstrapping.

There are also a host of useful functions (sigmoid, monomial functions, trigonometric functions, identity,...) which can be computed using only 2 bootstrapping operations because they are one sum away from an odd or even function.

In practice, since both the pseudo odd and the pseudo even functions are evaluated on the same input, a multi-value functional bootstrapping (see Section B.2) can be used to reduce the maximum amount of bootstrapping operations to 3 at the cost of some noise overhead leading to greater probability of error. Furthermore, since the pseudo odd and pseudo even functions can be computed independently, it is possible to use multi-thread computation to get the time down to only 2 bootstrapping operations.

Note that the Composition method is only suitable for precise arithmetic. Indeed, because of the negacyclic nature of the bootstrapping operation, we are actually composing discontinuous functions. This can lead to unexpected behaviors if the noise of the ciphertext is too big.

In the tables of Section 5 and Section 7 we will denote by Comp the Composition method with neither the use of multi-value bootstrapping nor multi-thread computation. We call CMV the method when used with the multi-value bootstrapping, and CMT with multi-threading.

Examples: We describe how to build the functions Id, and ReLU with the Composition method.

For Id, the decomposition in pseudo even and pseudo odd function gives $\mathsf{Id}(x) = (-\frac{1}{2p}) + (x + \frac{1}{2p})$. In this case the pseudo even function $-\frac{1}{2p}$ is a constant and does not need any functional bootstrapping to be computed. We then only need to apply the technique for pseudo odd functions to $x + \frac{1}{2p}$. In this case, there will be no use for multi-threading or using the multi-value technique as we will end up with 2 consecutive $\mathsf{BlindRotate}$ anyway.

Considering ReLU, the decomposition gives $\mathsf{ReLU}(x) = f(x) + g(x)$ where:

$$f : x \mapsto \begin{cases} \frac{x}{2} & \text{if } x \geq 0 \\ -\frac{x}{2} - \frac{1}{2p} & \text{otherwise} \end{cases}$$

and

$$g : x \mapsto \begin{cases} \frac{x}{2} & \text{if } x \geq 0 \\ \frac{x}{2} + \frac{1}{2p} & \text{otherwise} \end{cases}$$

Applying directly the Composition method would result in 4 BlindRotate. But the function f can actually be computed with only 1 bootstrapping similarly to the function $\tilde{h}$ for pseudo even functions. This specific improvement is useful for Comp, as it reduces the number of consecutive BlindRotate to 3, but it does not change anything for CMT which only need 2 BlindRotate. In the case of CMV, it is relevant to check whether lowering the number of consecutive BlindRotate from 3 to 2 outweighs any potential growth of the error rate, since CMV changes the test polynomials.

In Table 5 and Table 6, we call ReLU_1 the implementation of ReLU using 4 BlindRotate, while we refer by ReLU_2 to the optimized implementation of ReLU where f is computed with only one bootstrapping.

5 Error rate and noise variance

In this section, we analyse the noise variance and error rate for the aforementioned functional bootstrapping methods.

5.1 Noise variance

The noise variance of a bootstrapped ciphertext depends on the operations applied to the input ciphertext during bootstrapping. Table 1 summarizes the theoretical variance of each operation used in functional bootstrapping techniques. These formulas are taken from [Chi+19].

Table 1: Variance for operations applied to given independent inputs. C_i are TLWE ciphertexts of variance V_i, and TV is a TRLWE ciphertext of variance V.

Operation	**Variance**
Keyswitch(c_i)	$V_i + \mathcal{E}_{KS}$
BlindRotate(c_i, TV)	$V + \mathcal{E}_{BS}$
$P \cdot C_i$	$\|\|P\|\|_2^2 \cdot V_i$
$C_i + C_j$	$V_i + V_j$

Each of the bootstrapping methods of Section 4 relies on a composition of the operations presented in Table 1. Thus, we can find their resulting variance by simply composing the formulas from Table 1. Table 2 presents the noise variance for each of the discussed functional bootstrapping method.

Table 2: Variance for each bootstrapping operation

Bootstrapping	**Variance**
Half-Torus	$\mathcal{E}_{BS} + \mathcal{E}_{KS}$
FDFB	$\|\|v_2 - v_1\|\|_2^2 \cdot (\mathcal{E}_{BS} + 2 \cdot \mathcal{E}_{KS}) + \mathcal{E}_{BS} + \mathcal{E}_{KS}$
TOTA	$\mathcal{E}_{BS} + \mathcal{E}_{KS}$
Comp	$2 \cdot (\mathcal{E}_{BS} + \mathcal{E}_{KS})$
CMV	$(\|\|v_1\|\|_2^2 + \|\|v_2\|\|_2^2) \cdot \mathcal{E}_{BS} + 2 \cdot \mathcal{E}_{KS}$

We identify from Table 2 two kinds of functional bootstrapping algorithms. On the one hand, we have functional bootstrapping algorithms that do not use any intermediary polynomial multiplication, and end-up with a similar noise growth to a classical gate bootstrapping.

On the other hand, we have functional bootstrapping algorithms that present a quadratic growth, of the resulting noise variance, with respect to the norm of the used test polynomial. For the second category, it can be useful to reduce the resulting noise by using techniques such as the factorization given in Carpov et al., [CIM19] (described in Section B.2).

5.2 Probability of Error

The original TFHE gate bootstrapping technique comes with a probability of error that depends on the noise of the input ciphertext. Since every functional bootstrapping technique is inspired from the original gate bootstrapping algorithm, we need to evaluate their respective probability of error.

We first consider a single BlindRotate operation given a message $m^{(p)} \in \mathbb{Z}_p$, a TLWE ciphertext $\boldsymbol{c}=(\boldsymbol{a},b)$ where $b=(\langle \boldsymbol{a},s\rangle+m+e)$, and a TRLWE ciphertext $(\mathbf{0},t)$, where t is the test polynomial. Following the notation from Section 3.1, we have $m=E_p(m^{(p)})$. As mentioned in Section 3.2, the BlindRotate outputs $[t^*(\phi(\bar{\boldsymbol{a}},\bar{b}))]$. Note that $\phi(\bar{\boldsymbol{a}},\bar{b})=2N\cdot(m+e+r) mod [2N]$ where r is an error introduced when scaling and rounding the coefficients of $(\boldsymbol{a},b)$ from $\mathbb{T}$ to $\mathbb{Z}_{2N}$. To be more specific, r follows a translated Irwin-Hall distribution with variance $\frac{n+1}{48\cdot N^2}$. Since this distribution is an approximation for a centered Gaussian distribution, we will consider that r follows a centered Gaussian distribution with variance $\frac{n+1}{48\cdot N^2}$. We need the equality $[t^*(\phi(\bar{\boldsymbol{a}},\bar{b}))]=[f(m)]$ to hold true for any message $m^{(p)}$ in order to compute $\mathrm{LUT}_p(f)$ for a given negacyclic function f. To that end, we consider $t=P_{f,p}$. Then, we have: $[t^*(\phi(\bar{\boldsymbol{a}},\bar{b}))]=[f\left(\frac{\lfloor p\cdot(m+e+r)\rfloor}{p}\right)]$. With no assumptions regarding f, it follows that we need to have $|e+r|<\frac{1}{2p}$. With the assumption that e and r are independent random variables, the probability of this event is $P(|e+r|<\frac{1}{2p})=(\frac{1}{2p},V_c+V_r)$ where V_c and V_r are respectively the variances of the ciphertext and r. The probability of error is then $1-(\frac{1}{2p},V_c+V_r)$.

In case of multiple BlindRotate computations during a functional bootstrapping, each of them must succeed to get the expected output. We can use the formulas of probabilities for multiple independent or correlated events to find the the overall probability of error of a functional bootstrapping.

The probability of success, of the functional bootstrapping methods from Section 4, are summarized in Table 3. We denote by:

$$V=\mathcal{E}_{BS}+\mathcal{E}_{KS}$$

the variance of a simple gate bootstrapping, and by:

$$V_i=||v_i||_2^2\cdot\mathcal{E}_{BS}+\mathcal{E}_{KS}$$

the variance of a bootstrapping using an intermediary polynomial multiplication.

Table 3: Probability of success for each bootstrapping operation with plaintext size p

Bootstrapping	Probability of success
Half-Torus	$(\frac{1}{4\cdot p},V_c+V_r)$
FDFB	$(\frac{1}{2\cdot p},V_c+V_r)$
TOTA	$(\frac{1}{4},V_c+V_r)\cdot(\frac{1}{4\cdot p},\frac{V_c}{4}+V_r+V)$
Comp	$(\frac{1}{2\cdot p},V_c+V_r)\cdot(\frac{1}{2\cdot p},V+V_r)^2$
CMV	$(\frac{1}{2\cdot p},V_c+V_r)\cdot\prod_{i=0}^{1}(\frac{1}{2\cdot p},V_i+V_r)$

Some numerical considerations need to be taken into account to compute these values in practice. When working with floats, a probability of success $>1-2^{-52}$ will be rounded

to 1. Thus we need to work directly with the error rates using implementations of the $erfc = 1 - erf$ function instead of the erf. To that end, we notice that for 2 terms $_1$ and $_2$ close to 1, we have $1 - _1 \cdot _2 = (1 - _1) + (1 - _2) - (1 - _1) \cdot (1 - _2)$ where the right size of the equality can be computed without rounding problems. This also shows that the multiplication of probabilities that appear in Table 3 do not necessarily have a big impact on the result. Indeed, for an error rate $E = 1 - _1 \cdot _2$, if each $_i$ is close to 1 we get that $E \approx (1 - _1) + (1 - _2)$. In addition, if $(1 - _1)$ and $(1 - _2)$ are not close to each other, we even get $E \approx \max(1 - _1, 1 - _2)$. For example, we expect that for TOTA the probability of error follows the approximation $1 - (\frac{1}{4}, V_c + V_r) \cdot (\frac{1}{4 \cdot p}, \frac{V_c}{4} + V_r + V) \approx 1 - (\frac{1}{4 \cdot p}, \frac{V_c}{4} + V_r + V)$ for any set of parameters where a simple bootstrapping has a high chance of success.

The variances given as inputs of the formulas and the value of p will have a high impact on the error rate. Indeed $1 - (a, V)$ gets exponentially closer to 0 as a increases or V decreases. For this reason, for a given p and V, the error rate of the Half-Torus method $(1 - (\frac{1}{4p}, V))$ should be much higher than the probability of error of FDFB $(1 - (\frac{1}{2p}, V))$, for example.

6 Look-Up-Tables over Multiple Ciphertexts

In section 4, we discussed several functional bootstrapping methods that take as input one ciphertext. These methods have a limited plaintext space and precision, and allow evaluating look-up tables with a size bounded by the degree of the used cyclotomic polynomial (N). In addition, these methods are not suited for computing a LUT for a multivariate function f that takes as inputs two or more ciphertexts. In order to overcome these issues, we describe in this section a method for computing functions using multiple ciphertexts as inputs.

Our proposed solution improves the results of Guimarães et al., [GBA21]. They, themselves, generalize the ideas of Boura et al. [BST19] and discuss two methods for homomorphic computation with digits: a tree-based approach and a chaining approach. We expand on the chaining method in order to obtain any function through its use as opposed to the subset of function previously allowed.

Subsequently, we use this method to apply a LUT to a *single* message decomposed over *multiple* ciphertexts. That is, we decompose each plaintext into several digits in a certain base B and encrypt these digits separately. Decomposition allows working with a larger plaintext space $\mathcal{I}$ while using an acceptable parameters set for an efficient computation.

In this section, we first *review* the tree-based method and then *improve* the chaining method to make it fit any function. We show how those methods can be used as building blocks in order to compute additions and multiplications of messages decomposed over multiple ciphertexts. We then show how to compute the ReLU function over a single, decomposed, plaintext. The choice of ReLU as a worthy application of our novel method was made because it is the most used activation function in modern convolutional neural networks.

6.1 Tree-based Method

We consider d TLWE ciphertexts $(\boldsymbol{c}_0, ..., \boldsymbol{c}_{d-1})$ encrypting the messages $(m_0, ..., m_{d-1})$ over half of the torus and $B \in \mathbb{N}$, such that each ciphertext $\boldsymbol{c}_i$ corresponds to an encryption of $m_i \in [\![0, B-1]\!]$. We denote by $f : [\![0, B-1]\!]^d \to [\![0, B-1]\!]$ our target function and by g the bijection:

$$g : \begin{array}{ccc} [\![0, B-1]\!]^d & \to & [\![0, B^d - 1]\!] \\ (a_0, ..., a_{d-1}) & \mapsto & \sum_{i=0}^{d-1} a_i \cdot B^i \end{array}$$

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

$$P_i(X) = \sum_{j=0}^{B-1} \sum_{k=0}^{\frac{N}{B}-1} f \circ g^{-1}(j \cdot B^{d-1} + i) \cdot X^{j \cdot \frac{N}{B} + k}$$

Then, we apply the BlindRotate algorithm to $\boldsymbol{c}_{d-1}$ and each TRLWE(P_i), and use the SampleExtract algorithm to extract the first coefficient of the result. We end up with B^{d-1} TLWE ciphertexts each encrypting a message $f \circ g^{-1}(m_{d-1} \cdot B^{d-1} + i)$ for $i \in [\![0, B^{d-1} - 1]\!]$. Thanks to TLWE to TRLWE keyswitching, we batch them into B^{d-2} TRLWE ciphertexts corresponding to the LUT of h where:

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

We iterate this operation until getting only one TLWE ciphertext encrypting $f(m_0,\ldots,m_{d-1})$. Since a function from $[\![0, B-1]\!]^d$ to $[\![0, B-1]\!]^k$ can be decomposed in k functions from $[\![0,B-1]\!]^d$ to $[\![0,B-1]\!]$, we can actually build any function between any inputs, once they are decomposed in base B then encrypted.

Note that the BlindRotate algorithm is costly and we have to call it $\sum_{i=0}^{d-1} B^i = \frac{B^d-1}{B-1}$ times. Fortunately, we can make it faster by encoding the first LUTs in plaintext polynomials rather than TRLWE ciphertexts. Then, we use the multi-value bootstrapping given in [CIM19] to compute only one bootstrapping instead of B^{d-1} in the first step of the algorithm. Thus we end-up by running $1 + \sum_{i=0}^{d-2} B^i = 1 + \frac{B^{d-1}-1}{B-1}$ BlindRotate.

Proposition 1. *Let $\overline{\boldsymbol{c}}$ be the output of the tree-based functional bootstrapping algorithm for a given input on d digits. Then, if we don't use the multi-value bootstrapping for the first level of the tree, the variance of the noise of $\overline{\boldsymbol{c}}$ will verify:*

$$Var(Err(\overline{\boldsymbol{c}})) \leq d \cdot (\mathcal{E}_{BS} + \mathcal{E}_{KS})$$

If we use the multi-value bootstrapping with polynomials P_i we get:

$$Var(Err(\overline{\boldsymbol{c}})) \leq (d - 1 + \max(||P_i||_2^2)) \cdot \mathcal{E}_{BS} + d \cdot \mathcal{E}_{KS}$$

Proof. The result comes from the composition of the formulas for multi-value functional bootstrapping, keyswitching, and private functional bootstrapping. □

Proposition 2. *Let $(\boldsymbol{c}_i)_{i \in [\![1,d]\!]}$ be d TLWE ciphertexts corresponding to d digits of a plaintext message. Suppose that we differentiate $|\mathcal{M}|$ possible input values, the probability of error of the tree-based bootstrapping algorithm with inputs $(\boldsymbol{c_i})_{i \in [\![1,d]\!]}$ verifies:*

$$P(Err((\boldsymbol{c}_i)_{i \in [\![1,d]\!]})) = 1 - \prod_{i=1}^{d} erf\left(\frac{1}{4 \cdot |\mathcal{M}| \cdot \sqrt{V_{c_i} + V_r} \cdot \sqrt{2}}\right)$$

where $V_r = \frac{n+1}{48N^2}$ is the variance of the error induced by the rounding operation in the bootstrapping algorithm.

Proof. The result comes from the fact that for each i, $\boldsymbol{c}_i$ must have a noise low enough to allow for a successful BlindRotate. □

6.2 Chaining Method

The chaining method has a much lower complexity and a lower error growth than the tree-based method but, as presented in [GBA21], works only for a more restricted set of functions.

We consider n TLWE ciphertexts $(\boldsymbol{c}_0,\dots,\boldsymbol{c}_{n-1})$ encrypting the messages $(m_0,\dots,m_{n-1})$ respectively and denote by $LC(a,b)$ *any* linear combination of a and b. Given some functions $(f_i)_{i\in[\![0,n-1]\!]}$ so that $f_i:[\![0,B-1]\!]\to[\![0,B-1]\!]$, we can build a function $f:[\![0,B-1]\!]^n\to[\![0,B-1]\!]$ following Algorithm 3. Each f_i can be implemented in the homomorphic domain using any functional bootstrapping method described in Section 4. The result of this algorithm has the same noise as a simple functional bootstrapping, thus much less than the noise output of the tree method.

Algorithm 3 Chaining method

Input: A vector $(\boldsymbol{c}_0,\dots,\boldsymbol{c}_{n-1})$ of TLWE ciphertexts encrypting the vector of messages $(m_0,\dots,m_{n-1})$.
Output: A ciphertext encrypting $f(m_0,\dots,m_{n-1})$. f is defined here by the linear combinations chosen at every step and the different single-input functions f_i.

$\overline{\boldsymbol{c}}_0 \leftarrow f_0(\boldsymbol{c}_0)$
for $i\in[\![0,n-2]\!]$ **do**
  $\overline{\boldsymbol{c}}_{i+1}\leftarrow f_{i+1}(LC(\overline{\boldsymbol{c}}_i,\boldsymbol{c}_{i+1}))$
return $\overline{\boldsymbol{c}}_{\boldsymbol{n-1}}$

Most functions cannot be computed in such a simplistic way, which greatly restricts its use even though it can be effective for functions with carry-like logic as stated in [GBA21].

Generalization. It is possible to build any function f using a similar method. We introduce the function g such that:

$$g:\begin{array}{ccc}[\![0,B-1]\!]^2 & \to & [\![0,B^2-1]\!]\\ (a_0,a_1) & \mapsto & a_0+a_1\cdot B\end{array}$$

That function is a bijection, which means that if a ciphertext can hold any message in $[\![0,B^2-1]\!]$, then we can compute any function of two ciphertexts $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ by applying one functional bootstrapping over $g(\boldsymbol{c}_1,\boldsymbol{c}_2)$.

Note that when using base 2, we can easily build any logic gate with this method. We can then build a circuit with these gates for any functions. The same idea works for any base B. However, this generalization comes at the cost of multiple bits of padding and the conception of the proper circuit.

Proposition 3. *Let $\overline{\mathbf{c}}$ be the output of the chaining functional bootstrapping algorithm for given encrypted d digits. Then, the variance of the error of $\overline{\mathbf{c}}$ follows the same formula as the last functional bootstrapping method used in the chain.*

Proof. We get the result by applying the noise formula associated to the last functional bootstrapping in the chain and by noticing that it does not depend on the noise of the input. □

The probability of error is highly dependent on the choice of: the encoded LUT in the functional bootstrapping applied to each digit, the linear combinations between the inputs and outputs of the chained bootstrappings, and the structure of the circuit corresponding to the target function. Thus, a general formula cannot be given.

6.3 Addition

We expect additions of two messages to be computed in linear time with respect to the number of digits of each message. Thus the tree-based method is ill-suited for this operation, since the tree-based method computing time grows exponentially with the number of digits used as inputs. Meanwhile, the chaining method is not exactly adapted to this operation if applied directly. Nonetheless, we show that we can still use any of the two methods to compute the addition effectively.

Let $m_1 = \sum_{i=0}^{n} m_{1,i} \cdot B^i$ and $m_2 = \sum_{i=0}^{n} m_{2,i} \cdot B^i$ be two messages expressed in base B. For each pair (i,j), let $\boldsymbol{c}_{i,j}$ be the ciphertext encrypting the message $m_{i,j}$. We define $\boldsymbol{c}_i = (\boldsymbol{c}_{i,0},...,\boldsymbol{c}_{i,n})$ as the vector of ciphertexts encrypting m_i in base B. Finally, we denote by h the half adder function, and by f the full adder one:

$$h\colon \begin{array}{ccc} [\![0,B-1]\!]^2 & \to & [\![0,B-1]\!]^2 \\ (a,b) & \mapsto & ((a+b)[B], \lfloor (a+b)/B \rfloor) \end{array}$$

$$f\colon \begin{array}{ccc} [\![0,B-1]\!]^2 \times \{0,1\} & \to & [\![0,B-1]\!]^2 \\ (a,b,c) & \mapsto & ((a+b+c)[B], \lfloor (a+b+c)/B \rfloor) \end{array}$$

These two functions are the only requirements to build the addition operation. But, in order to be able to create those two adders, we need to create the following sub-functions:

$$mod\colon \begin{array}{ccc} [\![0,2B-1]\!] & \to & [\![0,B-1]\!] \\ x & \mapsto & x[B] \end{array}$$

$$carry\colon \begin{array}{ccc} [\![0,2B-1]\!] & \to & \{0,1\} \\ x & \mapsto & \lfloor x/B \rfloor \end{array}$$

Algorithm 4 Addition

Input: Two vectors of ciphertexts $\boldsymbol{c}_1 = (\boldsymbol{c}_{1,i})_{i \in [\![0,n-1]\!]}$ and $\boldsymbol{c}_2 = (\boldsymbol{c}_{2,i})_{i \in [\![0,m-1]\!]}$ encrypting two messages m_1 and m_2 written in base B. We suppose here that $n \geq m$.
Output: An encryption of $m_1 + m_2$ in base B.

$(\overline{\boldsymbol{c}}_{1,0}, \overline{\boldsymbol{c}}_{2,0}) \leftarrow h(\boldsymbol{c}_{1,0}, \boldsymbol{c}_{2,0})$
for $i \in [\![0, m-2]\!]$ **do**
  $(\overline{\boldsymbol{c}}_{1,i+1}, \overline{\boldsymbol{c}}_{2,i+1}) \leftarrow f(\boldsymbol{c}_{1,i+1}, \boldsymbol{c}_{2,i+1}, \overline{\boldsymbol{c}}_{2,i})$
for $i \in [\![m-1, n-2]\!]$ **do**
  $(\overline{\boldsymbol{c}}_{1,i+1}, \overline{\boldsymbol{c}}_{2,i+1}) \leftarrow h(\boldsymbol{c}_{1,i+1}, \overline{\boldsymbol{c}}_{2,i})$
return $(\overline{\boldsymbol{c}}_{1,0}, ..., \overline{\boldsymbol{c}}_{1,n-1}, \overline{\boldsymbol{c}}_{2,n-1})$

We can use either the tree-based method or the chaining method to compute *mod* or *carry* functions. The chaining method needs one bit of padding to work, while the tree-based method is slower, especially for the full adder which is a three inputs function. Finally, we present Algorithm 4 for computing addition between two vectors of ciphertexts.

The time complexity of Algorithm 4 is linear with respect to the number of digits of the entries. The noise of each output ciphertext is the same as the noise of a simple bootstrapping if we use the chaining method for computing the sub-functions *mod* and *carry*. Meanwhile, with the tree-based method, we end-up with the noise of a simple bootstrapping followed by two BlindRotate.

6.4 Multiplication

As we expected linear computation time to be achievable for the homomorphic addition, we expect to achieve quadratic time complexity for homomorphic multiplication. Let m_1

and m_2 be two messages and $\boldsymbol{c}_1=(\boldsymbol{c}_{1,i})_{i\in[\![0,n-1]\!]}$ and $\boldsymbol{c}_2=(\boldsymbol{c}_{2,i})_{i\in[\![0,m-1]\!]}$ be their encryption in base B. In order to evaluate $m_1\cdot m_2$ in the encrypted domain, we first multiply each digit of m_1 by each digit of m_2. Then, we have just to add the obtained elements properly using half and full adders to get the final result.

Since we have already introduced homomorphic adders, we only need to describe how to multiply two digits. Given two messages a and b in $[\![0,B-1]\!]$, we need to compute $a\cdot b[B]$ and $a\cdot b/B$ in the encrypted domain. If we use the tree-base method, we can compute both functions with three LUTs since both functions will use the same selector in the first step. Otherwise, we can also use the generalized chaining method to compute both needed functions using two LUTs, but this method comes at the cost of using multiple bits of padding.

We denote by $\mathsf{MultDigits}(\boldsymbol{c}_a,\boldsymbol{c}_b)$ a method for computing $a\cdot b[B]$. In the same way, we denote by $\mathsf{CarryMult}(\boldsymbol{c}_a,\boldsymbol{c}_b)$ a method for computing $a\cdot b/B$. Then the multiplication of m_1 and m_2 can be done with Algorithm 5.

Algorithm 5 Multiplication

Input: Two vectors of ciphertexts $\boldsymbol{c}_1=(\boldsymbol{c}_{1,i})_{i\in[\![0,n-1]\!]}$ and $\boldsymbol{c}_2=(\boldsymbol{c}_{2,i})_{i\in[\![0,m-1]\!]}$ encrypting two messages m_1 and m_2 written in base B.
Output: An encryption $\overline{\boldsymbol{c}}=(\overline{\boldsymbol{c}}_i)_{i\in[\![0,n+m-1]\!]}$ of $m_1\cdot m_2$ in base B.

```
for i ∈ [[0,n+m−1]] do
    SubMul_i ← empty vector
for i ∈ [[0,n−1]] do
    for j ∈ [[0,m−1]] do
        Put MultDigits(c_{1,i}, c_{2,j}) in vector SubMul_{i+j}
        Put CarryMult(c_{1,i}, c_{2,j}) in vector SubMul_{i+j+1}
c̄_0 ← SubMul_0[0]
for i ∈ [[1,n+m−1]] do
    c̄_i ← (Σ_{j=0}^{size(SubMul_i)−1} SubMul_i[j])[B] using adders
    Put the carries in SubMul_{i+1}
return (c̄_0, ..., c̄_{n+m−1})
```

The time complexity of Algorithm 5 is quadratic with respect to the number of digits of the entries. The noise of the outputs is similar to the noise of the adder sub-functions.

6.5 ReLU

In this section, we describe how to avoid using the tree-based method, as it is, for the implementation of the ReLU activation function. Let's consider $m=\sum_{i=0}^{n}m_i\cdot B^i$ a message written using radix complement representation in base B, and $(\boldsymbol{c}_i)_{i\in[\![0,n]\!]}=(\mathrm{TLWE}_{\boldsymbol{s}}(m_i))_{i\in[\![0,n]\!]}$.

In order to use the tree-based method to evaluate intermediate functions on each encrypted digit, we use a functional bootstrapping to create a selector S from $\boldsymbol{c}_n$ that encrypts the torus element 0 if $0\leq m_n<\frac{B}{2}$ and $\frac{1}{4}$ if $\frac{B}{2}\leq m_n<B$. Note that $(0\leq m_n<\frac{B}{2})\iff(m\geq 0)$, so the value of S depends on the sign of m. Then, for each $\boldsymbol{c}_i$, we create using keyswitching a TRLWE ciphertext $\mathrm{LUT}(\boldsymbol{c}_i)$ so that for $j\in[\![0,\frac{N}{2}-1]\!]$, $\mathsf{SampleExtract}(\mathrm{LUT}(\boldsymbol{c}_i),j)$ is an encryption of m_i, and for $j\in[\![\frac{N}{2},N-1]\!]$, $\mathsf{SampleExtract}(\mathrm{LUT}(\boldsymbol{c}_i),j)$ is an encryption of 0. Then, $\mathsf{SampleExtract}(\mathsf{BlindRotate}(S,\mathrm{LUT}(\boldsymbol{c}_i),0)$ outputs:

$$\overline{c_i}=\begin{cases}\mathrm{TLWE}(0,s) & \text{if } m<0\\ \mathrm{TLWE}(m_i,s) & \text{if } m\geq 0\end{cases}$$

Thus, $(\overline{\boldsymbol{c}}_i)_{i\in[\![0,n]\!]}$ encrypts $\mathsf{ReLU}(m)$ using radix complement representation in base B.

Otherwise, we can compute the ReLU function using the chaining method. Then, each ciphertext has to encrypt a value in $[\![0,2B[\![$. First, let's compute a selector S from c_n such that:

$$S = \begin{cases} \text{TLWE}(0,s) & \text{if } m \geq 0 \\ \text{TLWE}(B,s) & \text{if } m < 0 \end{cases}$$

Then, let's define:

$$f: \begin{array}{ccc} [\![0,2B-1]\!] & \to & [\![0,2B-1]\!] \\ x & \mapsto & \begin{cases} x & \text{if } x < B \\ 0 & \text{if } x \geq B \end{cases} \end{array}$$

This function can be computed with one functional bootstrapping. For each $\boldsymbol{c}_i$, we compute $\overline{\boldsymbol{c}}_i = f(\boldsymbol{c}_i + S)$. We obtain $(\overline{\boldsymbol{c}}_i)_{i \in [\![0,n]\!]}$ an encryption using radix complement representation in base B of ReLU(m).

7 Experimental Results

In this section, we compare time and accuracy performances for each of the aforementioned functional bootstrapping methods over single ciphertexts. All experiments[7] were made on an Intel Core i5-8250U CPU @ 1.60GHz by extending TFHE official open source library[8].

7.1 Parameters

We present in Table 4 the parameters sets used for our tests. We generated these parameters by following these guidelines:

- We need the security level λ to be as low as possible to get fast operations, while ensuring that encryption parameters are secure. So, we fix it to $\lambda = 80$ or 120 bits, which are the lowest security levels usually considered as secure.
- For efficiency, we want N to be a small power of 2. We notice that for $N = 512$, the noise level required for ensuring security is too high to compute properly a functional bootstrapping. Thus, we choose $N = 1024$, which is the default value for the degree of the cyclotomic polynomial within TFHE.
- We have more leeway for selecting n as it does not need to be a power of 2. However, as the security of the bootstrapping algorithm relies on $\min(n,N)$, having n greater than N will not improve security. So, for efficiency reasons, we stick with values of n smaller or equal to N.
- For every set of λ, N, and n, we used the LWE-estimator to find the lowest value for the noise standard deviation σ_{min} while ensuring an acceptable security level.

The remaining parameters, present in Table 4, are unrelated to the security level of the cryptosystem. We choose them using the following guidelines:

- Usually, the parameters for the keyswitching algorithm are $t = 3$ and $B_{KS} = 128 = 2^7$ (so B_{KS}_bits $= 7$). Since these parameters have a low impact on the resulting noise and time of each algorithm, we keep them as is.
- For faster bootstrapping operations, we need to have l as low as possible. However, we need to have B_g^l high enough to ensure a better precision when using the gadget matrix decomposition. We find that if $(l \times B_g$_bits$)$ is higher than 18, we get a satisfying

[7] Code available at https://github.com/CEA-LIST/Cingulata/experiments/tfhe-funcbootstrap-experiments.zip

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

precision. Since l has a smaller impact on the noise than B_g_bits, having smaller l and bigger B_g_bits results in more noise. For our tests, we find that $l=3$ and B_g_bits$=7$ is usually a good choice. We increase l only if the probability of success of functional bootstrapping gets too low for a given set of parameters.

Table 4: Parameters sets

Set	n	N	l	B_g_bits	t	B_{KS}_bits	σ_{min}	λ
1	1024	1024	3	7	3	7	$7.8e^{-09}$	120
2	900	1024	4	5	3	7	$8.4e^{-08}$	120
3	800	1024	9	2	3	7	$5.9e^{-07}$	120
4	1024	1024	3	7	3	7	$1.05e^{-11}$	80
5	900	1024	3	7	3	7	$5e^{-11}$	80
6	800	1024	3	7	3	7	$3.5e^{-10}$	80
7	700	1024	3	7	3	7	$5.5e^{-09}$	80
8	600	1024	3	6	3	7	$9.4e^{-08}$	80

7.2 Error Rate

In Table 5, we compute the probability of error for the functional bootstrapping methods (of Section 4) with respect to every set of parameters described in Table 4. We will assume that the input ciphertext has the noise of a ciphertext freshly bootstrapped using the considered method. This allows to have a fair evaluation of the ability of using the same method consecutively.

Note that the error rate of each method does not depend on the function computed during the bootstrapping except for FDFB and CMV. For FDFB, we evaluate the error rate for the functions Id and ReLU as well as the worst case scenario. The latter refers to a function that maximizes the noise level. Since we use the factorization technique introduced by Carpov et al., [CIM19] (Section B.2) the test polynomial v_2-v_1 of the worst case function has $\frac{p}{2}$ non-zero values each equal to p. If we apply now the FDFB error variance formula from Table 2, we obtain the noise bound of the output ciphertext: $\frac{p^3}{2}\cdot(\mathcal{E}_{BS}+2\cdot\mathcal{E}_{KS})+\mathcal{E}_{BS}+\mathcal{E}_{KS}$.

For CMV, we consider ReLU_1 and ReLU_2 (introduced as examples in Section 4.4). Indeed, computing Id is irrelevant with CMV, as it does not use the multi-value bootstrapping. Furthermore, 2 worst case scenarios can be identified. The first one happens when following the decomposition $f_{\mathbb{T}}\circ G_{\text{odd}}+f_{\mathbb{T}}\circ G_{\text{even}}$ given in Section 4.4, and using a multi-value bootstrapping to compute G_{odd} and G_{even} at the same time. In this case, the result will be exactly the same as explained for ReLU_1. The second scenario corresponds to computing any other function (not using G_{odd} and G_{even} in the composition method) via multi-value bootstrapping while trying to reduce the total number of bootstrapping (as discussed for ReLU_2). Then, no bound can be put on the norm of the polynomials and the worst error rate would be 100%. For the CMV, we only consider the worst case when following the decomposition given in Section 4.4.

The results show that for any given set of parameters, the probability of error is identical between TOTA and the Half-Torus (HT) method, or slightly in favor of the latter. Meanwhile, Comp and CMT get much better results than any other method in every case.

We notice that FDFB and CMV do not behave in the same fashion as the other methods with respect to parameters changing. The cases where they favorably compare to the others (as in set 6 where FDFB and CMV reach respectively an error rate of 2^{-174} and 2^{-171} while HT reaches an error rate of 2^{-47}) occur when $\mathcal{E}_{KS}+\mathcal{E}_{BS}$ is small compared to V_r. In these cases, the overhead of the noise created by the intermediary polynomial multiplication is absorbed by V_r. In the opposite case, when V_r is small compared to $\mathcal{E}_{KS}+\mathcal{E}_{BS}$, the impact of the multiplication is not absorbed and FDFB and CMV unfavorably compare to the other

Table 5: $-log_2$ of error rate for $p=8$

Set		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
HT		35	25	20	37	42	47	51	17
TOTA		34	23	18	37	42	47	51	15
FDFB	**Worst**	10	2	1	91	103	114	25	1
	Id	62	13	8	135	154	172	123	6
	ReLU	71	17	10	137	155	174	135	7
Comp		123	66	48	140	159	179	191	37
CMT		123	66	48	140	159	179	191	37
CMV	$\mathbf{ReLU_1}$	58	12	8	135	153	171	117	5
	$\mathbf{ReLU_2}$	22	4	2	117	132	147	52	2

methods (as in set 8 where FDFB and CMV reach respectively an error rate of 2^{-7} and 2^{-5} while HT reaches an error rate of 2^{-17}). The specific values of the polynomial also has to be taken into account when trying to gauge whether the parameters are favorable or not towards these methods. Indeed, in simple cases such as the ReLU and Id functions, we can see a huge improvement (from 2^{-10} to 2^{-71} for set 1) compared to the worst case approximation for FDFB. Note that with our assumptions regarding the worst case of CMV and FDFB, some functions would have a better error rate with CMV than with FDFB.

7.3 Time Performance

Our implementation results in Table 6 show that the speed of each method can be closely approximated by the speed of one simple BlindRotate multiplied by the number of consecutive BlindRotate needed. The Half-Torus method is the fastest as it only requires one BlindRotate. Then, TOTA is slightly faster than FDFB as it requires less key switching operations. It is also slightly faster than CMT as it does not induce parallelism overhead, but the difference is negligible. As far as the Comp method is concerned, the number of BlindRotate depends on the evaluated function. For a simple function such as the absolute value, its speed is identical to the Half-Torus method. Meanwhile, more complex functions need up to 4 bootstrapping, and so Comp becomes twice slower than TOTA, FDFB and CMT.

To conclude, our composition method is on par with other methods through the use of multi-threading, but gets less interesting if this possibility is denied.

7.4 Time-Error trade offs

The trade-offs between speed and error rate for each method are summarized in Figure 1a and Figure 1b.

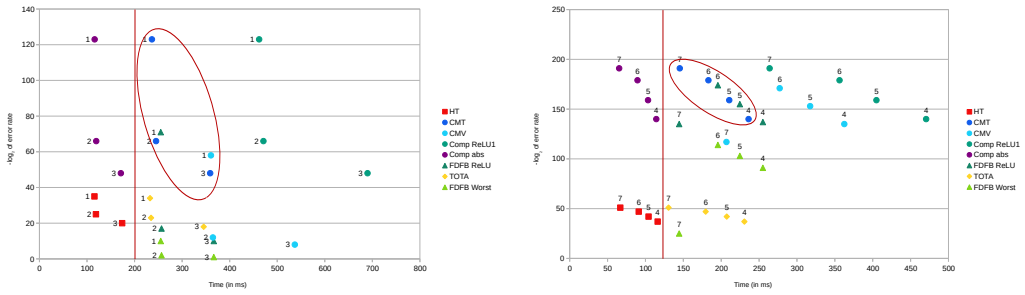
We separated the sets defined in Table 4 with sets 1 to 3 in Figure 1a and sets 4 to 7 in Figure 1b in order to have better readability of the figures. The set 8 is ignored here for aesthetic reasons, but observations are similar to the other sets regarding the relative effectiveness of each method.

For FDFB, we represent both the worst case and the ReLU which is the best case among the functions we considered. For the Comp, CMV and CMT methods, the best case is represented with the absolute value function and noted Comp abs. The Comp ReLU_1, CMV and CMT points are all relative to the ReLU_1 function. It is chosen as an instance of the worst case for all methods when applying the pseudo even and odd decomposition.

Fast operations will result in having points closer to the left. Meanwhile, a low error rate is represented with points close to the upper parts of the graphs. With those two considerations in mind, we can notice that the only methods on the left of the red line are the Half-Torus

Table 6: Time in ms

Set		**1**	**2**	**3**	**4**
HT		115.7	118.8	173.9	116.1
TOTA		232.5	234.7	345.4	230.6
FDFB		255.1	256.7	366.6	255.0
Comp	**abs**	116.1	119.4	171.2	114.3
	$\mathbf{ReLU_1}$	462.0	471.0	689.9	470.5
	$\mathbf{ReLU_2}$	347.9	357.1	518.3	343.9
CMT		236.5	245.2	359.0	236.1
CMV	**$\mathbf{ReLU_1}$**	360.6	364.7	537.0	362.5
	$\mathbf{ReLU_2}$	244.5	247.9	363.6	244.1
Set		**5**	**6**	**7**	**8**
HT		104.1	91.1	66.8	47.9
TOTA		207.3	179.5	130.5	95.3
FDFB		224.6	195.6	144.5	104.8
Comp	**abs**	103.4	89.5	65.3	47.3
	$\mathbf{ReLU_1}$	404.9	356.2	263.9	191.4
	$\mathbf{ReLU_2}$	308.9	264.5	197.2	142.4
CMT		210.6	183.1	145.3	109.0
CMV	**$\mathbf{ReLU_1}$**	317.4	277.2	206.9	149.1
	$\mathbf{ReLU_2}$	216.5	188.6	139.9	101.0



(a) Set of parameters 1 to 3

(b) Set of parameters 4 to 7

Figure 1: Time-Error trade off

and Comp in the best case scenario. In this specific scenario, the Comp method is the best in all regards. Otherwise, a compromise between speed and error rate must be made. In the red circle lies the points relative to the CMT method. We can clearly see that it is both more accurate and faster than all the other methods except for the Half-Torus one. Thus, it is the best alternative to the Half-Torus method among the suggested functional bootstrapping.

8 Conclusion

Through the use of several bootstrapping operations and - in some cases - additional operations, every full domain method (Sections 4.2, 4.3 and 4.4) adds some output noise when compared to the simpler and quicker partial domain method (Section 4.1). The question is: does a larger initial plaintext space make up for the added noise and computation time? Table 5 and Table 6 show us that the Yan et al., [Yan+21] (TOTA) method is both less accurate and twice as time-consuming than the partial domain method. Kluczniak and Schild's [KS21] (FDFB) method, gets a better accuracy for well chosen parameters but is still twice as time-consuming

as the partial domain method. Our novel composition method (Section 4.4) has a smaller error rate than any other method presented here. In addition, it comes without additional loss of speed compared to the other full domain methods as long as we use multi-threading.

Given these experimental measures, the Half-Torus bootstrapping stays the go to method to use unless one of the following points apply:

- The parameters of the cryptosystem are limited due to application constraint and the error rate of the Half-Torus is too high.
- Intermediary operations such as additions and multiplications would push messages out of the Half-Torus space.
- Modular arithmetic needs to be computed.

In these cases, our CMT method is a good alternative as it has the smallest error rate among the discussed methods and is as fast or faster than the other methods.

Besides, if a large plaintext space needs to be used, relying on a base decomposition technique is the best choice. Indeed they are the only options allowing for computations on large plaintext space using TFHE.

Furthermore, the operators presented in this paper provide key building blocks for enabling advanced deep learning functions over encrypted data.

References

[Bou+20] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. "CHIMERA: Combining Ring-LWE-based Fully Homomorphic Encryption Schemes". In: *Journal of Mathematical Cryptology* 14.1 (1Jan. 2020), pp. 316–338. DOI: https://doi.org/10.1515/jmc-2019-0026. URL: https://www.degruyter.com/view/journals/jmc/14/1/article-p316.xml.

[Bou+19] 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.

[Bou+18] F. Bourse, M. Minelli, M. Minihold, and P. Paillier. "Fast Homomorphic Evaluation of Deep Discretized Neural Networks". In: *Proceedings of CRYPTO 2018*. Springer, 2018.

[BST19] Florian Bourse, Olivier Sanders, and Jacques Traoré. *Improved Secure Integer Comparison via Homomorphic Encryption*. Cryptology ePrint Archive, Report 2019/427. https://ia.cr/2019/427. 2019.

[Bra12] Zvika Brakerski. "Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP". In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 868–886. ISBN: 978-3-642-32009-5.

[CIM19] 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.

[Cha+19] Herve Chabanne, Roch Lescuyer, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. "Recognition Over Encrypted Faces: 4th International Conference, MSPN 2018, Paris, France". In: 2019.

[Cha+17] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. *Privacy-Preserving Classification on Deep Neural Network*. Cryptology ePrint Archive, Report 2017/035. 2017.

[Che+17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. "Homomorphic Encryption for Arithmetic of Approximate Numbers". In: (2017). Ed. by Tsuyoshi Takagi and Thomas Peyrin.

[CKP19] Jung Hee Cheon, Duhyeong Kim, and Jai Hyun Park. "Towards a Practical Clustering Analysis over Encrypted Data". In: *IACR Cryptology ePrint Archive* (2019).

[Chi+16] 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.

[Chi+17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. "Faster Packed Homomorphic Operations and Efficient Circuit Bootstrapping for TFHE". In: *ASIACRYPT*. 2017.

[Chi+] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. *TFHE: Fast Fully Homomorphic Encryption Library*. URL: https://tfhe.github.io/tfhe/.

[Chi+19] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. "TFHE: Fast Fully Homomorphic Encryption Over the Torus". In: *Journal of Cryptology* 33 (Apr. 2019). DOI: 10.1007/s00145-019-09319-x.

[CJP21] Ilaria Chillotti, Marc Joye, and Pascal Paillier. "Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks". In: *Cyber Security Cryptography and Machine Learning*. 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.

[Chi+21] 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. https://ia.cr/2021/729. 2021.

[DM15] Léo Ducas and Daniele Micciancio. "FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second". In: *Advances in Cryptology – EUROCRYPT 2015*. Ed. by Elisabeth Oswald and Marc Fischlin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 617–640. ISBN: 978-3-662-46800-5.

[FV12] Junfeng Fan and Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Report 2012/144. https://ia.cr/2012/144. 2012.

[GBA21] Antonio Guimarães, Edson Borin, and Diego F. Aranha. "Revisiting the functional bootstrap in TFHE". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.2 (Feb. 2021), pp. 229–253. DOI: 10.46586/tches.v2021.i2.229-253. URL: https://tches.iacr.org/index.php/TCHES/article/view/8793.

[ISZ19] 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.

[JA18] Angela Jäschke and Frederik Armknecht. "Unsupervised Machine Learning on Encrypted Data". In: *IACR Cryptology ePrint Archive* (2018).

[KS21] 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.

[Lou+20] Qian Lou, Bo Feng, Geoffrey Charles Fox, and Lei Jiang. "Glyph: Fast and Accurately Training Deep Neural Networks on Encrypted Data". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 9193–9202. URL: https://proceedings.neurips.cc/paper/2020/file/685ac8cadc1be5ac98da9556bc1c8d9e-Paper.pdf.

[Mad+21] Abbass Madi, Oana Stan, Aurélien Mayoue, Arnaud Grivet-Sébert, Cédric Gouy-Pailler, and Renaud Sirdey. "A Secure Federated Learning framework using Homomorphic Encryption and Verifiable Computing". In: *2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS)*. 2021, pp. 1–8. DOI: 10.1109/RDAAPS48126.2021.9452005.

[Nan+19] Karthik Nandakumar, Nalini Ratha, Sharath Pankanti, and Shai Halevi. "Towards Deep Neural Network Training on Encrypted Data". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2019, pp. 40–48. DOI: 10.1109/CVPRW.2019.00011.

[Séb+21] Arnaud Grivet Sébert, Rafael Pinot, Martin Zuber, Cédric Gouy-Pailler, and Renaud Sirdey. "SPEED: secure, PrivatE, and efficient deep learning". In: *Mach. Learn.* 110.4 (2021), pp. 675–694. DOI: 10.1007/s10994-021-05970-3. URL: https://doi.org/10.1007/s10994-021-05970-3.

[Xie+14] Pengtao Xie, Misha Bilenko, Tom Finley, Ran Gilad-Bachrach, Kristin E. Lauter, and Michael Naehrig. "Crypto-Nets: Neural Networks over Encrypted Data". In: *CoRR* (2014).

[Yan+21] 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.

[ZCS20a] Martin Zuber, Sergiu Carpov, and Renaud Sirdey. "Towards Real-Time Hidden Speaker Recognition by Means of Fully Homomorphic Encryption". In: *Information and Communications Security.* Ed. by Weizhi Meng, Dieter Gollmann, Christian D. Jensen, and Jianying Zhou. Cham: Springer International Publishing, 2020, pp. 403–421. ISBN: 978-3-030-61078-4.

[ZCS20b] Martin Zuber, Sergiu Carpov, and Renaud Sirdey. "Towards real-time hidden speaker recognition by means of fully homomorphic encryption". In: *International Conference on Information and Communications Security.* Springer. 2020, pp. 403–421.

[ZS21] Martin Zuber and Renaud Sirdey. "Efficient homomorphic evaluation of k-NN classifiers". In: *Proc. Priv. Enhancing Technol.* 2021.2 (2021), pp. 111–129. DOI: 10.2478/popets-2021-0020. URL: https://doi.org/10.2478/popets-2021-0020.

A Encoding

We give here an example to highlight the use of the term $\frac{1}{2p}$ in the encoding function.

The test vector that one would give to the bootstrapping algorithm in order to implement a partial domain *identity* LUT over $\mathbb{Z}_4$ is:

$$\sum_{k=0}^{\frac{N}{4}-1} \frac{0}{8} X^k + X^{\frac{N}{4}} \cdot \sum_{k=0}^{\frac{N}{4}-1} \frac{1}{8} X^k + X^{\frac{2N}{4}} \cdot \sum_{k=0}^{\frac{N}{4}-1} \frac{2}{8} X^k + X^{\frac{3N}{4}} \cdot \sum_{k=0}^{\frac{N}{4}-1} \frac{3}{8} X^k$$

Figure 2 represents the possible outputs of the bootstrapping algorithm using this test vector. It illustrates clearly the natural negacyclicity of the operation. However, the figure shows that an encrypted input $\left[\frac{1}{8}\right]$ has a non-negligible chance of producing the output $\frac{0}{8}$ due to the presence of noise. This would happen should the error introduced during the encryption be negative and larger than $\frac{1}{4N}$.

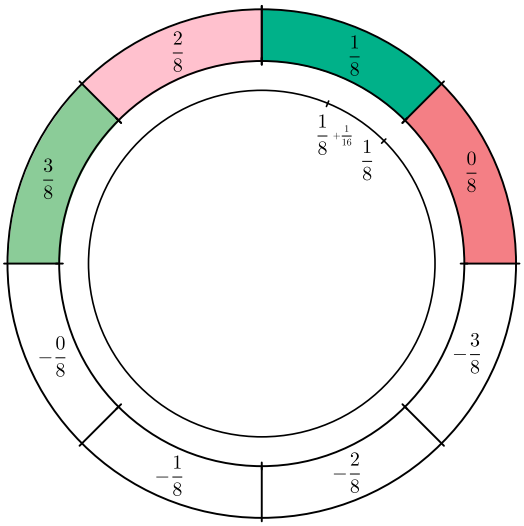


Figure 2: A representation of the possible output values of a partial domain functional bootstrapping implementation of the identity function over a message space of size 4.

Thus, we introduce an offset of $+\frac{1}{16}$, which places the input message at the center of the arc corresponding to the right output. This greatly diminishes the risk of an incorrect output value.

Figure 3 shows the encrypted polynomial at the output of the bootstrapping operation in both cases: with inputs $\left[\frac{1}{8}\right]$ and $\left[\frac{1}{8}+\frac{1}{16}\right]$. In every case the first coefficient is extracted in order to obtain the TLWE encryption of the output value. The colors match those from Figure 2. To make things clearer, the polynomials were chosen to be the ones at the output in the case where the error in the input ciphertexts is 0. But the error actually introduces another offset which leads to a slightly different rotation of the polynomial, and the extraction of a close yet different coefficient. We can see that this effect is more likely to lead to the wrong output for $[\frac{1}{8}]$ than for $[\frac{1}{8}+\frac{1}{16}]$.

We can also take the offset into account directly inside of the LUT of the bootstrapping algorithm rather than at encryption time. This way, we can work with a more natural encryption

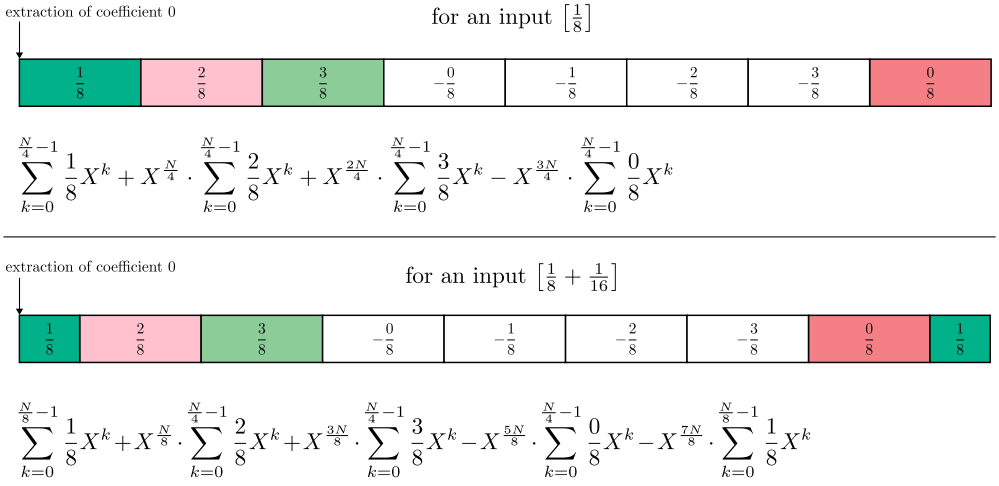


Figure 3: A representation of the output polynomial (before extraction) of the bootstrapping operation represented in Figure 2.

relatively to the homomorphic addition and multiplication of ciphertexts. However, putting the offset in the encryption allows formulas in Section 4.4 to be less confusing.

B More Functional Bootstrapping

We describe in this section two more variants of the original functional bootstrapping from the literature.

B.1 Private Functional Bootstrapping

The functional bootstrapping algorithm can be adapted to compute an encrypted negacyclic function. Indeed, given a negacyclic function $f : \mathbb{T} \to \mathbb{T}$, we create $[P_f]_{\mathbb{T}_N[X]}$, a TRLWE ciphertext whose i^{th} slot is a TLWE ciphertext encrypting $\tilde{f}(i)$. Such a ciphertext can be created using the TFHE public functional key-switching operation (see Algorithm 2 of [Chi+19]) from N TLWE ciphertexts $\left[\tilde{f}(i)\right]_{\mathbb{T}}$.

Let $\boldsymbol{c} = (\boldsymbol{a}, b) \in \text{TLWE}(m)$. Then, the Algorithm 6 outputs an encryption of $\tilde{f}(\phi(\bar{\boldsymbol{a}}, \bar{b}))$.

Algorithm 6 Encrypted LUT

Input: a TLWE sample $\boldsymbol{c} = (\boldsymbol{a}, b) \in \text{TLWE}_{\boldsymbol{s}}(m)$ with $m \in \mathbb{T}$, a bootstrapping key $BK_{\boldsymbol{s}\to\boldsymbol{s'}} = (BK_i \in \text{TRGSW}_{S'}(s_i))_{i\in[\![1,n]\!]}$ where S' is the TRLWE interpretation of a secret key $\boldsymbol{s'}$, an encryption $[P_f]_{\mathbb{T}_N[X]}$ of the polynomial P_f

Output: a TLWE sample $\overline{\boldsymbol{c}} \in \text{TLWE}_{\boldsymbol{s}}(f(\frac{\phi(\bar{\boldsymbol{a}},\bar{b})}{2N}))$

1: Let $\bar{b} = \lfloor 2Nb \rceil$ and $\bar{a}_i = \lfloor 2Na_i \rceil \in \mathbb{Z}, \forall i \in [\![1,n]\!]$
2: Let $testv := [P_f]_{\mathbb{T}_N[X]}$
3: $ACC \leftarrow \mathsf{BlindRotate}(testv, (\bar{a}_1, ..., \bar{a}_n, \bar{b}), (BK_1, ..., BK_n))$
4: $\overline{\boldsymbol{c}} = \mathsf{SampleExtract}(ACC)$
5: return $\mathsf{KeySwitch}_{\boldsymbol{s'}\to\boldsymbol{s}}(\overline{\boldsymbol{c}})$

B.2 Multi-Value Functional Bootstrapping

Carpov et al., [CIM19] introduced a nice method for evaluating multiple different negacyclic LUTs using one bootstrapping. Indeed, they factor the test polynomial P_{f_i} associated to the function f_i into a product of two polynomials v_0 and v_i, where v_0 is a common factor to all P_{f_i}. In fact, they notice that:

$$(1+X+\cdots+X^{N-1})\cdot(1-X)=2 \mod [X^N+1] \tag{3}$$

Let $P_{f_i}=\sum_{j=0}^{N-1}\alpha_{i,j}X^j$ with $\alpha_{i,j}\in\mathbb{T}$, and $q\in\mathbb{N}^*$ the smallest integer so that: $\forall i$, $q\cdot(1-X)\cdot P_{f_i}\in\mathbb{Z}[X]$. Note that q is a divisor of p. We obtain using equation (3):

$$\begin{aligned}P_{f_i}&=\frac{1}{2q}\cdot(1+\cdots+X^{N-1})\cdot(q\cdot(1-X)\cdot P_{f_i}) \mod [X^N+1]\\&=v_0\cdot v_i \mod [X^N+1]\end{aligned}$$

where:

$$\begin{aligned}v_0&=\frac{1}{2q}\cdot(1+\cdots+X^{N-1})\\v_i&=q\cdot(\alpha_{i,0}+\alpha_{i,N-1}+\sum_{j=1}^{N-1}(\alpha_{i,j}-\alpha_{i,j-1})\cdot X^j)\end{aligned}$$

Thanks to this factorization, we are able to compute many LUTs with one bootstrapping. Indeed, we just have to set the initial test polynomial to $testv=v_0$ during the bootstrapping. Then, after the BlindRotate, we multiply the obtained ACC by each v_i corresponding to LUT(f_i) to obtain ACC$_i$.

Algorithm 7 Multi-value bootstrapping

Input: a TLWE sample $\boldsymbol{c}=(\boldsymbol{a},b)\in\text{TLWE}_{\boldsymbol{s}}(m)$ with $m\in\mathbb{T}$, a bootstrapping key $BK_{\boldsymbol{s}\to\boldsymbol{s'}}=(BK_i\in\text{TRGSW}_{S'}(s_i))_{i\in[\![1,n]\!]}$ where S' is the TRLWE interpretation of a secret key $\boldsymbol{s'}$, k LUTs s.t. LUT$(f_i)=v_0.v_i,\forall i\in[\![1,k]\!]$

Output: a list of k TLWE samples $\bar{\boldsymbol{c}}_{\boldsymbol{i}}\in\text{TLWE}_{\boldsymbol{s}}(f_i(\frac{\phi(\bar{\boldsymbol{a}},\bar{b})}{2N}))$

1: Let $\bar{b}=\lfloor 2Nb\rceil$ and $\bar{a}_i=\lfloor 2Na_i\rceil\in\mathbb{Z},\forall i\in[\![1,n]\!]$
2: Let $testv:=v_0$
3: ACC$\leftarrow$ BlindRotate$((\mathbf{0},testv),(\bar{a}_1,...,\bar{a}_n,\bar{b}),(BK_1,...,BK_n))$
4: **for** $i\leftarrow 1$ to k **do**
5: ACC$_i$:=ACC$\cdot v_i$
6: $\bar{\boldsymbol{c}}_{\boldsymbol{i}}=$ SampleExtract(ACC$_i$)
7: return KeySwitch$_{\boldsymbol{s'}\to\boldsymbol{s}}(\bar{\boldsymbol{c}}_{\boldsymbol{i}})$

It would have been more intuitive to use polynomials $v_0=\frac{1}{q'}$ and $v_i'=q'\cdot P_{f_i}$ for $q'\in\mathbb{N}^*$ the smallest integer so that all v_i' have coefficients in $\mathbb{Z}$. But it would lead to an error growth of $||v_i'||_2^2.\mathcal{E}_{BS}$ higher than $||v_i||_2^2.\mathcal{E}_{BS}$ in most cases.
Indeed, if we consider a plaintext space with $p<N$ values, the coefficients of P_{f_i} will have some redundancy. As a consequence, v_i will have only $\frac{p}{2}$ non-zero coefficients. For example, consider a plaintext space with only 2 possible values. Given a negacyclic LUT(f_i), every coefficient of the polynomial P_{f_i} will then have the same value. Thus, each v_i would have at most 1 non zero coefficient.
In addition, since $q'\cdot P_{f_i}\in\mathbb{Z}[X]$, it follows that $q'\cdot(1-X)\cdot P_{f_i}$ is also in $\mathbb{Z}[X]$. Hence q is a divisor of q'. Since the integer polynomials v_i are multiplied by the torus polynomial v_0 with coefficients each equal to $\frac{1}{2q}$, we can consider each coefficient of the v_i polynomials

with modulo $2q$ and use values in $[\![-q,q-1]\!]$. The same reasoning applies to the polynomials v'_i with modulo q' and values in $[\![-\lfloor \frac{q'}{2} \rfloor, \lceil \frac{q'}{2} \rceil - 1]\!]$. Thus, we get that $||v_i||_2^2 \leq \frac{p}{2} \cdot q^2$ and $||v'_i||_2^2 \leq N \cdot (\frac{q'}{2})^2$. In practice, p is small compared to N and q is often smaller than $\frac{q'}{2}$. Then, the bound is much better for v_i than v'_i and the decomposition used here is a powerful way to mitigate the error growth of the method.