ComBo: a Novel Functional Bootstrapping Method for Efficient Evaluation of Nonlinear Functions in the Encrypted Domain

Pierre-Emmanuel Clet, Aymen Boudguiga, Renaud Sirdey, Martin Zuber

ComBo: a Novel Functional Bootstrapping Method for Efficient Evaluation of Nonlinear Functions in the Encrypted Domain*

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

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

Abstract. The application of Fully Homomorphic Encryption (FHE) to privacy issues arising in inference or training of neural networks has been actively researched over the last few years. Yet, although practical performances have been demonstrated on certain classes of neural networks, the inherent high computational cost of FHE operators has prevented the scaling capabilities of FHE-based encrypted domain inference to the large and deep networks used to deliver advanced classification functions such as image interpretation tasks. To achieve this goal, a new hope is coming from TFHE functional bootstrapping which, rather than being just used for refreshing ciphertexts (i.e., reducing their noise level), can be used to evaluate operators which are difficult to express as low complexity arithmetic circuits, at no additional cost. In this work, we first propose ComBo (Composition of Bootstrappings) 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 on previous approaches: like them, we allow for evaluating any functions, but with error rates reduced by a factor of up to 2^{80}. This claim is supported by a theoretical analysis of the error rate of other functional bootstrapping methods from the literature. The paper is concluded by extensive experimental results demonstrating that our method achieves better performances in terms of both time and precision, in particular for the Rectified Linear Unit (ReLU) function, a nonlinear activation function commonly used in neural networks. As such, this work provides a fundamental building-block towards scaling the homomorphic evaluation of neural networks over encrypted data.

Keywords: FHE; TFHE; functional bootstrapping; ReLU; ComBo

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 [3, 12, 11, 7, 33, 25, 34]. Even training machine learning models with privacy guarantees on the training data has been investigated in the centralized [26, 14, 30, 29] and collaborative [31, 1] settings. In practice, machine learning

* This work was supported by the France 2030 ANR Project ANR-22-PECY-003 SecureCompute.

algorithms and especially neural networks require the computation of non-linear activation functions such as the sign, ReLU or sigmoid functions. Still, computing non-linear functions homomorphically remains challenging. For levelled homomorphic schemes such as BFV [9, 23] or CKKS [13], non-linear functions have to be approximated by polynomials. However, the precision of these approximations 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 [15, 18] or FHEW [22], can be tweaked to encode functions via look-up table (LUT) evaluations within their bootstrapping procedure. Hence, rather than being just used for refreshing ciphertexts (i.e., reducing their noise level), the bootstrapping becomes *functional* [8] or *programmable* [19] by allowing the evaluation of arbitrary functions as a bonus. These capabilities result in promising new approaches for improving the overall performances of homomorphic calculations, making the FHE "API" better suited to the evaluation of mathematical operators which are difficult to express as low complexity arithmetic circuits. 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 [5, 33]. 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 inner 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 – The main contribution of this paper is a novel functional bootstrapping algorithm[1]. It is a *full domain* functional bootstrapping algorithm in the sense that it does not require to add a bit of padding to the encoding of the messages (as described clearly in [19]). There are several other such methods in the literature. We show that ours is the best option to date for single-digit operations on the full torus (where a message is encoded into a single ciphertext).
There are several other contributions in this paper. We present them succinctly here:

- Our *novel functional bootstrapping algorithm* (ComBo) is built by composing several bootstrapping operations. It is based on the idea to separate any function in a even and odd part and then compute both in parallel. We present several versions to increase its efficiency and show that our method is the most accurate among state-of-the art full domain bootstrapping algorithms.
- We *implement and test* our algorithms by evaluating several functions homomorphically. Among them, the Rectified Linear Unit (ReLU) function is of particular interest for private neural network applications. This allows us to compare the computational overhead of our algorithm with other existing methods.
- In order to compare the error rate of the different existing methods (which this work aims to reduce), *we develop an error analysis methodology* and describe it in detail. This shows that *our algorithm improves on previous approaches*, most of the time

[1] This paper is an updated version of the eprint [21]

by a significant margin. This methodology, we argue, is the most appropriate way to compare similar algorithms and can be reused for further research on the subject to improve comparability.
– As a bonus, in order to compare our algorithm fairly to other previous solutions from the community, *we introduce consistent notations for describing all existing solutions and their error probabilities in a unified way.* We also fully implemented and tested all of them. We consider that this strengthens the present paper and can be considered, in and of itself, a worthy contribution to the development of the field.

Related works – In 2016, the TFHE paper made a breakthrough by proposing an efficient bootstrapping for homomorphic gate computation. Then, Bourse et al., [7] and Izabachene et al., [25] used the same bootstrapping algorithm for extracting the (encrypted) sign of an encrypted input. Boura et al., [6] 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 [27], Liu et al., [28] and Yang et al., [32] proposed 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., [24] extended the ideas in Bourse et al., [8] to support the evaluation of certain activation functions such as the sigmoid. One last method (WoP-PBS), presented in Chillotti et al., [20] achieves a functional bootstrapping over the full torus using a BFV type multiplication, which was designed for and only applicable to parameter sets much larger than standard TFHE parameters. Besides, since the probabilistic behavior of decryption also appears during the bootstrapping procedure, the error rate analysis of homomorphic computation are becoming of interest when using TFHE as shown in [24] and [2].

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 presents our new functional bootstrapping method ComBo in full detail. It also describes, under a unified formalism, the other available methods for single digit functional bootstrapping. Finally, Section 6 provides experimental results for ComBo and compares it to the other methods which we also implemented. These results are provided for both generic LUT evaluations over encrypted data as well as the ReLU neural network activation function.

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 $\mathbb{R}/\mathbb{Z}$ as $\mathbb{T}$. $\mathbb{T}_N[X]$ 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 $\mathcal{P}(a, \sigma^2)$. 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 [18].

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 [15]. It improves the FHEW cryptosystem [22] and introduces the TLWE problem as an adaptation of the LWE problem to $\mathbb{T}$. It was updated later in [16] and both works were recently unified in [18]. The TFHE scheme is implemented in the TFHE library [17]. 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 TLWE 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 TRLWE 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[2]. To encrypt a message $m \in \mathcal{M}$, we add $(\mathbf{0}, m) \in \{0\}^k \times \mathcal{M}$ to a TRLWE sample (or to a TLWE sample 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 $\boldsymbol{c} \in \text{T(R)LWE}_{\boldsymbol{s}}(m)$.

To decrypt a ciphertext $\boldsymbol{c} \in \text{T(R)LWE}_{\boldsymbol{s}}(m)$, we compute its *phase* $\phi(\boldsymbol{c}) = b - \langle \boldsymbol{a}, \boldsymbol{s} \rangle = m + e$. Then, we round 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:** a valid TRGSW sample is a vector of TRLWE samples. To encrypt a message $m \in \mathcal{R}$, we add $m \cdot H$ to a TRGSW sample, where H is a gadget matrix[3] using an integer B_g as a base for its decomposition. Chilotti et al., [18] 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 [18]). It yields an encryption of $m_a \cdot m_b$ i.e., a TRLWE sample $\boldsymbol{c} \in \text{TRLWE}_{\boldsymbol{s}}(m_a \cdot m_b)$. Otherwise, the external product allows also to compute a controlled MUX gate (CMUX) where the selector is $C_b \in \text{TRGSW}_{\boldsymbol{s}}(b), b \in \{0,1\}$, and the inputs are $\boldsymbol{c}_0 \in \text{TRLWE}_{\boldsymbol{s}}(m_0)$ and $\boldsymbol{c}_1 \in \text{TRLWE}_{\boldsymbol{s}}(m_1)$.

2.3 TFHE Bootstrapping

TFHE bootstrapping relies mainly on three building blocks:

- **Blind Rotate:** rotates a plaintext polynomial encrypted as a TRLWE ciphertext by an encrypted position. It takes as inputs: a TRLWE ciphertext $\boldsymbol{c} \in \text{TRLWE}_{\boldsymbol{k}}(m)$, a vector $(a_1, \dots, a_n, a_{n+1} = b)$ where $\forall i,\ a_i \in \mathbb{Z}_{2N}$, and n TRGSW ciphertexts encrypting $(s_1, \dots, s_n)$ where $\forall i,\ s_i \in \mathbb{B}$. It returns a TRLWE ciphertext $\boldsymbol{c'} \in \text{TRLWE}_{\boldsymbol{k}}(X^{\langle \boldsymbol{a}, \boldsymbol{s} \rangle - b} \cdot m)$. In this paper, we will refer to this algorithm as BlindRotate. With respect to independence heuristic[4] stated in [18], the variance $\mathcal{V}_{BR}$ of the resulting noise after a BlindRotate satisfies the formula:
$$\mathcal{V}_{BR} < V_c + n\left((k+1)\ell N \left(\frac{B_g}{2}\right)^2 \vartheta_{BK} + \frac{(1+kN)}{4 \cdot B_g^{2l}}\right)$$
where V_c is the variance of the noise of the input ciphertext c, and ϑ_{BK} is the variance of the error of the bootstrapping key. In the following, we define:
$$\mathcal{E}_{BR} = n\left((k+1)\ell N \left(\frac{B_g}{2}\right)^2 \vartheta_{BK} + \frac{(1+kN)}{4 \cdot B_g^{2l}}\right)$$
- **TLWE Sample Extract:** takes as inputs both a ciphertext $\boldsymbol{c} \in \text{TRLWE}_{\boldsymbol{k}}(m)$ and a position $p \in [\![0, N[\![$, and returns a TLWE ciphertext $\boldsymbol{c'} \in \text{TLWE}_{\boldsymbol{k}}(m_p)$ where m_p is

[2] In practice, we discretize the Torus with respect to our plaintext modulus. For example, the usual encryption of a message $m \in \mathbb{Z}_4 = \{0,1,2,3\}$ would be one of the following value $\{0, 0.25, 0.5, 0.75\}$.

[3] Refer to Definition 3.6 and Lemma 3.7 in TFHE paper [18] for more information about the gadget matrix H.

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

the p^{th} coefficient of the polynomial m. In this paper, we will refer to this algorithm as 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 $\mathbb{T}_{\overline{N}}[X]$. 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 as KeySwitch. As stated in [18, 24], the variance $\mathcal{V}_{KS}$ of the resulting noise after KeySwitch follows the formula[5]:

$$\mathcal{V}_{KS} < R^2 \cdot V_c + n\overline{N}\left(t\vartheta_{KS} + \frac{B_{KS}^{-2t}}{4}\right)$$

where V_c is the variance of the noise of the input ciphertext c, R is the Lipschitz constant of f and ϑ_{KS} the variance of the error of the keyswitching key. Note that n is a parameter of the input ciphertext, while $\overline{N}$ is a parameter of the output ciphertext. Thus, $\overline{N}=1$ if the output is a TLWE ciphertext. In this paper and in most cases, $R=1$. In the following, we define:

$$\mathcal{E}_{KS}^{n,\overline{N}} = n\overline{N}\left(t\vartheta_{KS} + \frac{B_{KS}^{-2t}}{4}\right)$$

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. Note that $\{0,1\}$ is encoded as $\{0,\frac{1}{2}\}$. Step 1 consists in selecting a value $\mu \in \mathbb{T}$ which will serve later for setting the coefficients of the test polynomial *testv* (in step 3). Step 2 rescales the components of the input ciphertext $\boldsymbol{c}$ as elements of $\mathbb{Z}_{2N}$. Step 3 defines the test polynomial *testv*. Note that for all $p \in [\![0,2N[\![$, the constant term of $testv \cdot X^p$ is μ if $p \in]\!]\frac{N}{2}, \frac{3N}{2}[\![$ and $-\mu$ otherwise. 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 $-\mu$ if $\boldsymbol{c}$ encrypts 0, or μ if $\boldsymbol{c}$ encrypts $\frac{1}{2}$ as long as the noise of the ciphertext is small enough. Then, step 5 creates a new ciphertext $\overline{\boldsymbol{c}}$ by extracting the constant term of ACC and adding to it $(\mathbf{0},\mu)$. That is, $\overline{\boldsymbol{c}}$ either encrypts 0 if $\boldsymbol{c}$ encrypts 0, or m if $\boldsymbol{c}$ encrypts $\frac{1}{2}$ (By choosing $m=\frac{1}{2}$, we get a fresh encryption of $\boldsymbol{c}$). 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: $\mathcal{V}_{BS} < \mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1}$.

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

[5] Note that there is a discrepancy in the original TFHE papers [15, 16, 18] between the theorem and the proof.

Algorithm 1 TFHE gate bootstrapping [18]

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 $\bar{\boldsymbol{c}} \in \mathrm{TLWE}_{\boldsymbol{s}}(x.m)$

1: Let $\mu = \frac{1}{2}m \in \mathbb{T}$ (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 $testv := (1 + X + \cdots + X^{N-1}) \cdot X^{\frac{N}{2}} \cdot \mu \in \mathbb{T}_N[X]$
4: $ACC \leftarrow \mathsf{BlindRotate}((\mathbf{0}, testv), (\bar{a}_1, ..., \bar{a}_n, \bar{b}), (BK_1, ..., BK_n))$
5: $\bar{\boldsymbol{c}} = (\mathbf{0}, \mu) + \mathsf{SampleExtract}(ACC)$
6: return $\mathsf{KeySwitch}_{\boldsymbol{s'}\to\boldsymbol{s}}(\bar{\boldsymbol{c}})$

3 TFHE Functional Bootstrapping

3.1 Encoding and Decoding

Our goal is to build a 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:

$$E_p : \begin{array}{ccc} \mathbb{Z}_p & \to & \mathbb{T} \\ k & \mapsto & \frac{k}{p} \end{array}$$

and its corresponding decoding function:

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

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

$$\begin{array}{ccc} \mathbb{Z}_p & \xrightarrow{f = D_p \circ f_{\mathbb{T}} \circ E_p} & \mathbb{Z}_p \\ E_p \downarrow & & \uparrow D_p \\ \mathbb{T} & \xrightarrow[f_{\mathbb{T}}]{} & \mathbb{T} \end{array}$$

Since the function $f_{\mathbb{T}} = E_p \circ f \circ D_p$ makes the diagram commutative, we consider this function as the encoding of f over $\mathbb{T}$.

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

3.2 Functional Bootstrapping Idea

The original bootstrapping algorithm from [15] had already all the tools to implement a LUT of any negacyclic function[6]. In particular, TFHE is well-suited for

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

$\frac{1}{2}$-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), where $\forall\alpha\in[\![0,N[\![$, $X^{\alpha+N}\equiv -X^\alpha \bmod [X^N+1]$.

Boura et al., [6] were the first to use the term *functional bootstrapping* for TFHE. They describe how TFHE bootstrapping computes a sign function. In addition, they use bootstrapping to build a Rectified Linear Unit (ReLU). However, they do not delve into the details of how to implement the ReLU in practice[7].

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.

Algorithm 2 Sign extraction with bootstrapping

Input: a constant $\mu\in\mathbb{T}$, a TLWE sample $\boldsymbol{c}=(\boldsymbol{a},b)\in \mathrm{TLWE}_{\boldsymbol{s}}(m)$ with $m\in\mathbb{T}$, 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 $\bar{\boldsymbol{c}}\in \mathrm{TLWE}_{\boldsymbol{s}}(\mu.sign(m))$

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: $\bar{\boldsymbol{c}}=\mathsf{SampleExtract}(ACC)$
5: return $\mathsf{KeySwitch}_{\boldsymbol{s'}\to\boldsymbol{s}}(\bar{\boldsymbol{c}})$

When we look at the building blocks of Algorithm 2, we notice that we can 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:

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

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).

Indeed, we first remind that for any positive integer i s.t. $0\le i<N$, we have:

$$testv.X^{-i}=t_i+\cdots-t_0X^{N-i}-\cdots-t_{i-1}X^{N-1} \bmod [X^N+1] \tag{1}$$

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

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

[7] They build the function $2\times$ReLU from an absolute value function, but do not explain how to divide by two to get the ReLU result.

– if $\phi(\bar{\boldsymbol{a}},\bar{b}) \in [\![-N,0[\![$, we have:
$testv \cdot X^{-\phi(\bar{\boldsymbol{a}},\bar{b})} = -testv \cdot X^{-\phi(\bar{\boldsymbol{a}},\bar{b})-N} \bmod [X^N+1]$
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 $[t^*(\phi(\bar{\boldsymbol{a}},\bar{b}))]_{\mathbb{T}}$.

Now, we can tweak the previous idea to evaluate discretized functions. 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:

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

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 error probability of the bootstrapping are given in Section 5.

3.3 Example of Functional Bootstrapping in $\mathbb{Z}_4$

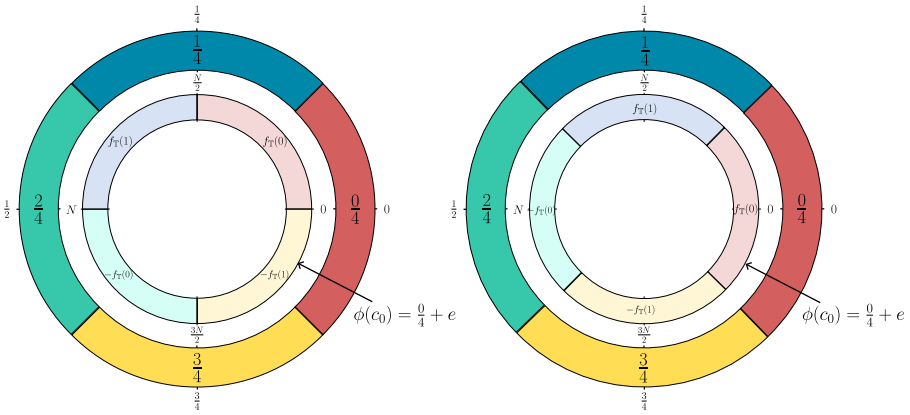

Fig. 1: Functional bootstrapping outputs with $\mathbb{Z}_4$ as plaintext space.

As an example, let us consider the plaintext space $\mathbb{Z}_4$ and a negacyclic function f. We represent $\mathbb{Z}_4$ in $\mathbb{T}$ by the set $\{\frac{0}{4},\frac{1}{4},\frac{2}{4},\frac{3}{4}\}$. We denote by $f_{\mathbb{T}}$ a function over $\mathbb{T}$ that satisfies: $f_{\mathbb{T}}(\frac{i}{4}) = \frac{f(i)}{4}$ for all $i \in [\![0,3]\!]$. We consider a ciphertext $\boldsymbol{c}_0$ encrypting the value

Algorithm 3 TFHE functional bootstrapping example

Input: a TLWE sample $\boldsymbol{c}=(\boldsymbol{a},b)\in \mathrm{TLWE}_{\boldsymbol{s}}(m)$ with $x\in\{\frac{0}{4},\frac{1}{4},\frac{2}{4},\frac{3}{4}\}$, 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 $\bar{\boldsymbol{c}}\in \mathrm{TLWE}_{\boldsymbol{s}}(f_{\mathbb{T}}(m))$

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 $testv:=P_{f_{\mathbb{T}},4}\cdot X^{-\frac{N}{4}}\in\mathbb{T}_N[X]$
3: $ACC\leftarrow \mathsf{BlindRotate}((\mathbf{0},testv),(\bar{a}_1,...,\bar{a}_n,\bar{b}),(BK_1,...,BK_n))$
4: $\bar{\boldsymbol{c}}=\mathsf{SampleExtract}(ACC)$
5: return $\mathsf{KeySwitch}_{\boldsymbol{s'}\to\boldsymbol{s}}(\bar{\boldsymbol{c}})$

0. We present in Algorithm 3 the functional bootstrapping algorithm that computes $LUT(f)$. We use the notation $P_{f,k}$ from Section 2.1.

In step 2 of Algorithm 3, we set the test polynomial $testv=P_{f_{\mathbb{T}},4}\cdot X^{-\frac{N}{4}}$, where $P_{f_{\mathbb{T}},4}$ encodes a look up table corresponding to $f_{\mathbb{T}}$, and $X^{-\frac{N}{4}}$ is an offset term.

In Figure 1, we describe the action of the offset $X^{-\frac{N}{4}}$ on $P_{f_{\mathbb{T}},4}$. We represent in the outer circle the possible phases associated to each entry from our plaintext space $\mathbb{Z}_4$. Meanwhile, we represent in the inner circle the returned coefficients after a bootstrapping. In the left part of Figure 1, we consider the result of the bootstrapping algorithm without the offset. We note that the red part of the inner and outer circles do not overlap. So, whenever the error term e in the phase is negative (even for small values of e), the considered functional bootstrapping outputs an incorrect value. In our example, the bootstrapping returns $f_{\mathbb{T}}(\frac{3}{4})=-f_{\mathbb{T}}(\frac{1}{4})$ instead of $f_{\mathbb{T}}(0)$. Meanwhile, in the right part of the Figure 1, we consider the bootstrapping algorithm with the offset. Now, the red part of the inner and outer circles overlap, and so, the functional bootstrapping returns the right value as long as the error term remains small enough.

For a given plaintext space $\mathbb{Z}_p$, the offset is $X^{-\lfloor\frac{N}{p}\rceil}$. We assume from now on that p divides N to ease notations and formulas.

3.4 Multi-Value Functional Bootstrapping

Carpov et al., [10] introduced a nice method for evaluating multiple LUTs with one bootstrapping. 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}. Indeed, they notice that:

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

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: $\forall i$, $q\cdot(1-X)\cdot P_{f_i}\in\mathbb{Z}[X]$ (q is a divisor of p). We get using equation (3):

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

where:

$$v_0 = \frac{1}{2q} \cdot (1 + \cdots + X^{N-1})$$

$$v_i = q \cdot (\alpha_{i,0} + \alpha_{i,N-1} + \sum_{j=1}^{N-1} (\alpha_{i,j} - \alpha_{i,j-1}) \cdot X^j)$$

Thanks to this factorization, it becomes possible 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 $\mathrm{LUT}(f_i)$ to obtain ACC_i.

Algorithm 4 Multi-value bootstrapping

Input: a TLWE sample $\boldsymbol{c} = (\boldsymbol{a}, b) \in \mathrm{TLWE}_{\boldsymbol{s}}(m)$ with $m \in \mathbb{T}$, 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 $\boldsymbol{s'}$, k LUTs s.t. $\mathrm{LUT}(f_i) = v_0.v_i, \forall i \in [\![1,k]\!]$

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

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: $\mathrm{ACC} \leftarrow \mathsf{BlindRotate}((\mathbf{0}, testv), (\bar{a}_1, ..., \bar{a}_n, \bar{b}), (BK_1, ..., BK_n))$
4: **for** $i \leftarrow 1$ to k **do**
5: $\quad \mathrm{ACC}_i := \mathrm{ACC} \cdot v_i$
6: $\quad \bar{\boldsymbol{c}}_{\boldsymbol{i}} = \mathsf{SampleExtract}(\mathrm{ACC}_i)$
7: $\quad$ return $\mathsf{KeySwitch}_{\mathsf{s'} \to \mathsf{s}}(\bar{\boldsymbol{c}}_{\boldsymbol{i}})$

4 Look-Up-Tables over a Single Ciphertext

In Section 3.2, we demonstrated that functional bootstrapping can serve to compute $\mathrm{LUT}(f)$ for any negacyclic function f. In this section, we describe 4 different ways to specify homomorphic LUTs for *any* function (i.e., not necessarily negacyclic ones). We present 3 solutions from the state of the art [19, 27, 32] in Sections 4.1, 4.2 and 4.3, and our novel method ComBo in Section 4.4. In addition, we discuss a solution to reduce the noise of the functional bootstrapping from [27] in Section 4.2.

As in Section 3.1, we call $f_{\mathbb{T}} : \mathbb{T} \to \mathbb{T}$ the function that specifies our homomorphic LUT, and $f : \mathbb{Z}_p \to \mathbb{Z}_p$ its corresponding function over the input and output space $\mathbb{Z}_p$.

4.1 Partial Domain Functional Bootstrapping – Half-Torus

The Half-Torus method gets around 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 P_h for a given negacyclic function h. Recall Equation 2 that defines the output of the bootstrapping operation as:

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

As we restrict the encoding space to $[0,\frac{1}{2}[$, we also restrict $\tilde{h}$ domain to $[\![0,N[\![$, where h has no negacyclic property. That is, we get a method to evaluate a LUT with a *single* bootstrapping.

4.2 Full Domain Functional Bootstrapping – FDFB

Kluczniak and Schild [27] specified FDFB to evaluate encrypted LUTs of domain the full 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 $P_2 = q \cdot P_g$. We note that the coefficients of $P_f - P_g$ are multiples of $\frac{1}{p}$ in $\mathbb{T}$, where $\mathbb{T}$ corresponds to $[-\frac{1}{2},\frac{1}{2}[$. We note that 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:

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

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

In order to evaluate a LUT, we first compute $[E_q(H(m))]_{\mathbb{T}}$ with 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:

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

We 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. As such, we obtain $[f_{\mathbb{T}}(m)]_{\mathbb{T}}$ after a second bootstrapping with $[m]_{\mathbb{T}}$ as input and $\boldsymbol{c}_{\text{LUT}}$ as a test polynomial.

We can reduce the noise of $\boldsymbol{c}_{\text{LUT}}$ by applying to P_f and P_g the factorization described in Section 3.4. First, 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. Thanks to the redundancy of the coefficients of P_f and P_g, v_f and v_g have at most $\frac{p}{2}$ non null coefficients. We denote by $q' \in \mathbb{N}^*$ the smallest integer such that $q' \cdot (v_f - v_g)$ is a polynomial with coefficients in $\mathbb{Z}$. We ensure that $q' \leq q$ as $q \cdot (1-X) \cdot (P_f - P_g) = (1-X) \cdot (q \cdot (P_f - P_g))$ has coefficients in $\mathbb{Z}$. 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. 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

Both Liu et al., [28] and Yan et al., [32] independently proposed the same approach[8] to evaluate arbitrary functions over the torus using a functional bootstrapping. As such, we refer to both methods in this paper with the name TOTA (as proposed by Yan et al.). Let's consider a ciphertext $[m_1]_{\mathbb{T}} = (\boldsymbol{a}, b = \langle \boldsymbol{a}, \boldsymbol{s} \rangle + m_1 + e)$. Then, by dividing each coefficient of this ciphertext by 2, we get a ciphertext $[m_2]_{\mathbb{T}} = (\frac{\boldsymbol{a}}{2}, \langle \frac{\boldsymbol{a}}{2}, \boldsymbol{s} \rangle + m_2 + \frac{e}{2})$ 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, we get an encryption of $\frac{m_1}{2}$ by computing: $[m_2]_{\mathbb{T}} - [\frac{\text{sign}(m_2)}{4}]_{\mathbb{T}} + (\mathbf{0}, \frac{1}{4})$.

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 approach from 4.1, and $f_{(2)}(\frac{m_1}{2}) = f_{\mathbb{T}}(m_1)$.

4.4 Full Domain Functional Bootstrapping with Composition - ComBo

In this section, we present ComBo, a novel method to compute any function using the full (discretized) torus as plaintext space. We will assume that p is even and fixed[9].

Pseudo odd functions: We call pseudo odd function a function f that satisfies: $\forall x \in \mathbb{Z}_p, f(-x-1) = -f(x)$.

Let f be a pseudo odd function over $\mathbb{Z}_p$. We define the following negacyclic functions:

$$f_{\text{neg}} : \begin{array}{ccc} [\![0, p-1]\!] & \to & \mathbb{Z}_p \\ x & \mapsto & \begin{cases} f(x) & \text{if } x \in [\![0, \frac{p}{2}-1]\!] \\ -f(x-\frac{p}{2}) & \text{if } x \in [\![\frac{p}{2}, p-1]\!] \end{cases} \end{array}$$

and

$$\text{Id}_{\text{neg}} : \begin{array}{ccc} [\![0, p-1]\!] & \to & \mathbb{R}_p \\ x & \mapsto & \begin{cases} x + \frac{1}{2} & \text{if } x \in [\![0, \frac{p}{2}-1]\!] \\ \frac{p}{2} - x - \frac{1}{2} & \text{if } x \in [\![\frac{p}{2}, p-1]\!] \end{cases} \end{array}$$

Since these 2 functions are negacyclic, they can be computed with the usual negacyclic functional bootstrapping (presented in section 3.2).

Note that $(\text{Id}_{\text{neg}} - \frac{1}{2})$ is a bijection of $\mathbb{Z}_p$ that satisfies the equality $(\text{Id}_{\text{neg}} - \frac{1}{2})(x) = x$, for all $x \in [\![0, \frac{p}{2}-1]\!]$. Otherwise, for all $x \in [\![\frac{p}{2}, p-1]\!]$, $(\text{Id}_{\text{neg}} - \frac{1}{2})(x) = \frac{p}{2} - x - 1$. In $\mathbb{Z}_p$, $\forall x \in [\![\frac{p}{2}, p-1]\!]$, we have $(\frac{p}{2} - x - 1) \in [\![\frac{p}{2}, p-1]\!]$.

[8] Although both papers use different notations, both methods rescale the message space into the first half of the torus before applying a half torus functional bootstrapping. In both cases, a sign evaluation is performed to compute that rescaling.

[9] If p is odd, we set $p := p+1$ to get back to the assumption that p is even.

Now, we compose it with f_{neg} to obtain: $f_{\text{neg}} \circ (\text{Id}_{\text{neg}} - \frac{1}{2})(x) = f_{\text{neg}}(x) = f(x)$ if $x \in [\![0, \frac{p}{2} - 1]\!]$. If $x \in [\![\frac{p}{2}, p-1]\!]$, $f_{\text{neg}} \circ (\text{Id}_{\text{neg}} - \frac{1}{2})(x) = f_{\text{neg}}(\frac{p}{2} - x - 1) = -f(-x-1)$. Since f is pseudo odd, we have: $-f(-x-1) = f(x)$.

Pseudo even functions: We call pseudo even function a function f that satisfies: $\forall x \in \mathbb{Z}_p, f(-x-1) = f(x)$.

Let f be a pseudo even function over $\mathbb{Z}_p$. We define the following negacyclic functions:

$$f_{\text{neg}} : \begin{array}{ccc} [\![0,p-1]\!] & \to & \mathbb{Z}_p \\ x & \mapsto & \begin{cases} f(x) & \text{if } x \in [\![0, \frac{p}{2} - 1]\!] \\ -f(x - \frac{p}{2}) & \text{if } x \in [\![\frac{p}{2}, p-1]\!] \end{cases} \end{array}$$

and

$$\text{abs}_{\text{neg}} : \begin{array}{ccc} [\![0,p-1]\!] & \to & \mathbb{R}_p \\ x & \mapsto & \begin{cases} x + \frac{p}{4} + \frac{1}{2} & \text{if } x \in [\![0, \frac{p}{2} - 1]\!] \\ \frac{p}{4} - x - \frac{1}{2} & \text{if } x \in [\![\frac{p}{2}, p-1]\!] \end{cases} \end{array}$$

Since these 2 functions are also negacyclic, they can similarly be computed with the usual negacyclic functional bootstrapping (presented in section 3.2).

Note that $(\text{abs}_{\text{neg}} - \frac{p}{4} - \frac{1}{2})$ satisfies the equality $(\text{abs}_{\text{neg}} - \frac{p}{4} - \frac{1}{2})(x) = x$ for all $x \in [\![0, \frac{p}{2} - 1]\!]$. However, if $x \in [\![\frac{p}{2}, p-1]\!]$, $(\text{abs}_{\text{neg}} - \frac{p}{4} - \frac{1}{2})(x) = -x - 1 \in [\![0, \frac{p}{2} - 1]\!]$. As such, we ensure that the function $(\text{abs}_{\text{neg}} - \frac{p}{4} - \frac{1}{2})$ behaves similarly to the absolute value function.

It follows that $f_{\text{neg}} \circ (\text{abs}_{\text{neg}} - \frac{p}{4} - \frac{1}{2})(x) = f_{\text{neg}}(x) = f(x)$ if $x \in [\![0, \frac{p}{2} - 1]\!]$. If $x \in [\![\frac{p}{2}, p-1]\!]$, $f_{\text{neg}} \circ (\text{abs}_{\text{neg}} - \frac{p}{4} - \frac{1}{2})(x) = f_{\text{neg}}(-x-1) = f(-x-1)$. Since f is pseudo even, we have $f(-x-1) = f(x)$.

Any function: We write any function $f \in \mathbb{Z}_p$ as a sum of a pseudo even function and a pseudo odd function: $f(x) = f_{\text{even}}(x) + f_{\text{odd}}(x)$, where $f_{\text{even}}(x) = \frac{f(x) + f(-x-1)}{2}$ and $f_{\text{odd}}(x) = \frac{f(x) - f(-x-1)}{2}$. Besides, we build any pseudo odd or pseudo even function with at most 2 bootstrappings. So, we can build any function with at most 4 bootstrappings.

We describe in Algorithm 5 the overall algorithm for running ComBo. We denote by $\mathsf{FB}[f]((\boldsymbol{a}, b))$ the application of the negacyclic functional bootstrapping procedure using the test vector $P_{\frac{f}{p}}$ (as defined in Section 2.1) and applied to a ciphertext $(\boldsymbol{a}, b)$ given a function $f : \mathbb{Z}_p \to \mathbb{R}_p$.

Correctness: If we assume that the negacyclic functional bootstrapping (FB) is correct, we obtain by Algorithm 5 a ciphertext $[\frac{f(m)}{p}]_{\mathbb{T}}$ where $m \in \mathbb{Z}_p$ is the input of the algorithm and $f : \mathbb{Z}_p \to \mathbb{Z}_p$ is the target function. Indeed, Step 1 computes an encryption of $\frac{\text{Id}_{\text{neg}}(m)}{p}$ since Id_{neg} is a negacyclic function. Step 2 computes an encryption of $\frac{\text{Id}_{\text{neg}}(m) - \frac{1}{2}}{p} = \frac{(\text{Id}_{\text{neg}} - \frac{1}{2})(m)}{p}$. Let us refer by f_{neg} to the negacyclic function corresponding to f_{odd} over $[\![0, \frac{p}{2} - 1]\!]$. Then Step 3 computes an encryption of $\frac{f_{\text{neg}} \circ (\text{Id}_{\text{neg}} - \frac{1}{2})(m)}{p}$: the encoding of $f_{\text{odd}}(m)$ over $\mathbb{T}$ (as discussed in the paragraph about pseudo odd functions in Section 4.4). Similarly, Steps 4 to 6 compute an encryption of the encoding of

Algorithm 5 ComBo

Input: a TLWE sample $[\frac{m}{p}]_{\mathbb{T}} \in \mathrm{TLWE}_{\boldsymbol{s}}(\frac{m}{p})$ with $m \in \mathbb{Z}_p$, 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 $\boldsymbol{s'}$, a target function $f : \mathbb{Z}_p \to \mathbb{Z}_p$, and the two functions $f_{\mathrm{odd}} : \begin{array}{ccc}\mathbb{Z}_p & \to & \mathbb{R}_p \\ x & \mapsto & \frac{f(x)-f(-x-1)}{2}\end{array}$ and $f_{\mathrm{even}} : \begin{array}{ccc}\mathbb{Z}_p & \to & \mathbb{R}_p \\ x & \mapsto & \frac{f(x)+f(-x-1)}{2}\end{array}$

Output: a TLWE ciphertext $(\boldsymbol{a'},b') = [\frac{f(m)}{p}]_{\mathbb{T}} \in \mathrm{TLWE}_{\boldsymbol{s}}(\frac{f(m)}{p})$

1: $(\boldsymbol{a},b) = \mathsf{FB}[\mathrm{Id}_{\mathrm{neg}}]([\frac{m}{p}]_{\mathbb{T}})$ ▷ Start of pseudo odd computation
2: $(\boldsymbol{a},b) = (\boldsymbol{a}, b-\frac{1}{2p})$
3: $(\boldsymbol{a}_{\mathrm{odd}},b_{\mathrm{odd}}) = \mathsf{FB}[f_{\mathrm{odd}}]((\boldsymbol{a},b))$ ▷ End of pseudo odd computation
4: $(\boldsymbol{a},b) = \mathsf{FB}[\mathrm{abs}_{\mathrm{neg}}]([\frac{m}{p}]_{\mathbb{T}})$ ▷ Start of pseudo even computation
5: $(\boldsymbol{a},b) = (\boldsymbol{a}, b-\frac{1}{2p}-\frac{1}{4})$
6: $(\boldsymbol{a}_{\mathrm{even}},b_{\mathrm{even}}) = \mathsf{FB}[f_{\mathrm{even}}]((\boldsymbol{a},b))$ ▷ End of pseudo even computation
7: $(\boldsymbol{a'},b') = (\boldsymbol{a}_{\mathrm{odd}},b_{\mathrm{odd}}) + (\boldsymbol{a}_{\mathrm{even}},b_{\mathrm{even}})$

$f_{\mathrm{even}}(m)$ over $\mathbb{T}$. Finally, Step 7 computes the sum of the pseudo odd and pseudo even outputs which results in an encryption of $\frac{f(m)}{p}$: the encoding of $f(m)$ over $\mathbb{T}$.

In practice, we can reduce the (single-shot) computation time by using parallelism (e.g. multithreading or SIMD) for evaluating the pseudo odd and pseudo even functions simultaneously. So, we end-up with a computation time of 2 bootstrappings. We can alternatively reduce the number of bootstrappings to 3 thanks to the multi-value functional bootstrapping (see Section 3.4).

From now on, we call ComBoMV the ComBo method when used with the multi-value bootstrapping, and ComboP with parallelism.

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

For Id, the decomposition in pseudo even and pseudo odd functions gives $\mathsf{Id}(x) = (-\frac{1}{2}) + (x+\frac{1}{2})$. The pseudo even function $\mathsf{Id}_{\mathrm{even}} = -\frac{1}{2}$ is a constant and does not require any bootstrapping. We only have to compute the pseudo odd function $\mathsf{Id}_{\mathrm{odd}} = x + \frac{1}{2}$. In this case, we have no need for multithreading or multi-value bootstrapping.

For ReLU, the decomposition gives $\mathsf{ReLU}(x) = \mathsf{ReLU}_{\mathrm{even}}(x) + \mathsf{ReLU}_{\mathrm{odd}}(x)$ where:

$$\mathsf{ReLU}_{\mathrm{even}} : x \mapsto \begin{cases} \frac{x}{2} & \text{if } x \in [\![0, \frac{p}{2}-1]\!] \\ -\frac{x}{2}-\frac{1}{2} & \text{otherwise} \end{cases}$$

$$\mathsf{ReLU}_{\mathrm{odd}} : x \mapsto \begin{cases} \frac{x}{2} & \text{if } x \in [\![0, \frac{p}{2}-1]\!] \\ \frac{x}{2}+\frac{1}{2} & \text{otherwise} \end{cases}$$

Applying ComBo naively results in 4 bootstrappings. However, we can actually compute $\mathsf{ReLU}_{\mathrm{even}}$ with only 1 bootstrapping as for $\mathrm{abs}_{\mathrm{neg}}$. This specific improvement is useful for ComBo, as it reduces the number of consecutive bootstrappings to 3.

5 Error Rate and Noise Variance

In this section, we analyze the noise variance and error rate for the aforementioned functional bootstrapping methods. We refer to each bootstrapping method by its acronym as defined in Section 4.

5.1 Noise Variance

The noise variance of a bootstrapped ciphertext depends on the operations applied to the input ciphertext during the bootstrapping. Table 1 gives the theoretical variance of each of these operations. These formulas are taken from [18].

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

Table 1: Obtained noise variances when applying basic operations to independent inputs: $\boldsymbol{c}_i$ is a TLWE ciphertext of variance V_i, $\boldsymbol{C}_i$ is a TRLWE ciphertext of variance V_i, P is a plaintext polynomial and $v\in\mathbb{Z}_{2N}^{n+1}$.

Each of the bootstrapping methods of Section 4 relies on a composition of the operations from Table 1. So, we compute their resulting variances in Table 2 by simply composing the formulas from Table 1.

Bootstrapping	**Variance**
Half-Torus	$\mathcal{E}_{BR}+\mathcal{E}_{KS}^{N,1}$
FDFB	$\|\|v_2-v_1\|\|_2^2\cdot(\mathcal{E}_{BR}+\mathcal{E}_{KS}^{N,1}+\mathcal{E}_{KS}^{n,N})+\mathcal{E}_{BR}+\mathcal{E}_{KS}^{N,1}$
TOTA	$\mathcal{E}_{BR}+\mathcal{E}_{KS}^{N,1}$
ComBo & ComBoP	$2\cdot(\mathcal{E}_{BR}+\mathcal{E}_{KS}^{N,1})$
ComBoMV	$(\|\|v_1\|\|_2^2+\|\|v_2\|\|_2^2)\cdot\mathcal{E}_{BR}+2\cdot\mathcal{E}_{KS}^{N,1}$

Table 2: Output noise variance of the aforementioned functional bootstrapping methods

We identify in 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 gate bootstrapping. On the other hand, we have functional bootstrapping algorithms that have a quadratic growth of the output noise variance with respect to the norm of the used test polynomial. For this second category, we can reduce the output noise by using the factorization technique described in Section 3.4.

5.2 Probability of Error

We discuss in this section the probabilities of error of all the functional bootstrapping methods from Section 4. Similar approaches to compute the probability of error of functional bootstrapping can be found in [2] and [24].

We first consider a single BlindRotate operation given a message $m^{(p)} \in \mathbb{Z}_p$, a TLWE ciphertext $(\boldsymbol{a},b)$ 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, applying a BlindRotate and extracting the first coefficient outputs $[t^*(\phi(\bar{\boldsymbol{a}},\bar{b}))]_{\mathbb{T}}$. Hence, 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 $t=P_{f,p}\cdot X^{-\frac{N}{p}}$ assuming that p divides N (we motivated this choice in Figure 1 and Section 3.3). Note that $\phi(\bar{\boldsymbol{a}},\bar{b})=2N\cdot(m+e+r) \bmod [2N]$ where r is an error introduced when scaling and rounding the coefficients of $(\boldsymbol{a},b)$ from $\mathbb{T}$ to $\mathbb{Z}_{2N}$. Thus, we have:

$$[t^*(\phi(\bar{\boldsymbol{a}},\bar{b}))]=\left[f\left(\frac{\left\lfloor \frac{p\cdot(\phi(\bar{\boldsymbol{a}},\bar{b})+\frac{N}{p})}{2N}\right\rfloor}{p}\right)\right]=\left[f\left(\frac{\lfloor p\cdot(m+e+r)+\frac{1}{2}\rfloor}{p}\right)\right]$$

It follows that $[t^*(\phi(\bar{\boldsymbol{a}},\bar{b}))]=[f(m)]$ as long as $|e+r|<\frac{1}{2p}$. The error r follows a translated Irwin-Hall distribution with variance $\frac{n+1}{48\cdot N^2}$ that, as is well known, can be closely approximated by a centered Gaussian distribution. With the assumptions that e and r are independent random variables, the probability that $|e+r|<\frac{1}{2p}$ is $\mathcal{P}(\frac{1}{2p},V_c+V_r)$, where V_c and V_r are respectively the variances of the ciphertext and r, and $\mathcal{P}$ is the notation introduced in Section 2.1. The probability of error is then $1-\mathcal{P}(\frac{1}{2p},V_c+V_r)$.

When multiple BlindRotate operations occur during a functional bootstrapping, each of them must succeed to ensure a correct computation. We can use the well known formulas of probabilities for independent or correlated events to find the overall probability of error of a functional bootstrapping method.

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

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

the variance of a simple gate bootstrapping, and by:

$$V_i=||v_i||_2^2\cdot\mathcal{E}_{BR}+\mathcal{E}_{KS}^{N,1}$$

the variance of a bootstrapping using an intermediary polynomial multiplication.

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

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

The variances and the value of p given as inputs to the formulas of Table 3 have a high impact on the error rate. Indeed, $1-\mathcal{P}(a,V)$ gets exponentially closer to 0 when a increases or when V decreases. For example, for a given p and V, the error rate of the Half-Torus method (i.e., $(1-\mathcal{P}(\frac{1}{4p},V))$) is higher than the probability of error of FDFB $(1-\mathcal{P}(\frac{1}{2p},V))$.

6 Experimental Results

In this section, we compare the computation time and the error rate for the functional bootstrapping methods of Section 4. We wrap up this section with a time-error trade-off analysis. All experiments[10] were implemented on an Intel Core i5-8250U CPU @ 1.60GHz by building on the TFHE open source library[11].

6.1 Parameters

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

- We fix the security level λ to 128 bits, which is the lowest security level considered as secure by present day standard.
- 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 large to compute properly a functional bootstrapping. Thus, we choose $N=1024$, which is the default value for the degree of the cyclotomic polynomial with TFHE.
- We note $\sigma_{\mathbb{T}_N[X]}$ the standard deviation used for the noise of the bootstrapping key and the keyswitch key from TLWE to TRLWE. We use the lattice-estimator [4] to set $\sigma_{\mathbb{T}_N[X]}$ as low as possible with respect to the security level λ. Thus, $\sigma_{\mathbb{T}_N[X]}=5.6\cdot10^{-8}$.

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

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

- For efficiency, we choose values of n lower than N. As such, we generate sets of parameters for all n between 700 and 1024 by step of 100.
- We note $\sigma_{\mathbb{T}}$ the standard deviation used for the noise of the keyswitch key from TLWE to TLWE and fresh ciphertexts. For each n, we use the lattice-estimator to set $\sigma_{\mathbb{T}}$ as low as possible with respect to the security level λ.

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

- We consider the Half-Torus method as the baseline for the error rate of each method. As such, we tailor sets of parameters to reach an error rate close to 2^{-30} using the Half-Torus method for a plaintext space of $p=8$.
- For faster bootstrapping operations, we need to have l as low as possible. We still need to select l high enough to reach the target error rate.
- For given l, n, N, and $\sigma_{\mathbb{T}_N[X]}$, we choose B_g to minimize the noise of the BlindRotate.
- For lower noise, we need B_{KS} to be as high as possible. Since the size of the keys grows with the basis, we set it to 1024 to avoid memory issues.
- For faster keyswitching operations, we need to have t as low as possible. We still need to select t high enough to reach the target error rate. Given the choice of B_{KS}, we find that $t=2$ is the optimal choice.

Set	n	l	B_g	$\sigma_{\mathbb{T}}$	$\sigma_{\mathbb{T}_N[X]}$
1	1024	5	16	$5.6e^{-08}$	$5.6e^{-08}$
2	1024	4	32	$5.6e^{-08}$	$5.6e^{-08}$
3	900	4	32	$5.1e^{-07}$	$5.6e^{-08}$
4	900	3	64	$5.1e^{-07}$	$5.6e^{-08}$
5	800	4	32	$3.1e^{-06}$	$5.6e^{-08}$
6	800	3	64	$3.1e^{-06}$	$5.6e^{-08}$
7	700	4	32	$1.9e^{-05}$	$5.6e^{-08}$
8	700	3	64	$1.9e^{-05}$	$5.6e^{-08}$

Table 4: Selected parameter sets with $p=8$, $N=1024$, $B_{KS}=1024$, $t=2$, and $\lambda=128$, following the guidelines of Section 6.1

6.2 Error Rate

In this section, 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.

In order to have a fair evaluation of the ability to consecutively bootstrap with the same method, we assume that the input to each method immediately follows a bootstrapping with the same method. We present in Table 5 the obtained error rates with respect to each method.

We note that the error rate of each method does not depend on the function computed during the bootstrapping except for FDFB and ComBoMV. Thus, we define a dedicated analysis methodology for these methods:

- For FDFB, we evaluate the error rate for the functions Id and ReLU as well as the worst case that maximizes the output noise. Since we use the multi-value bootstrapping factorization (described in Section 3.4), the worst case test polynomial $v_2 - v_1$ has $\frac{p}{2}$ non-zero values each equal to p. If we apply the FDFB error variance formula from Table 2, we obtain the worst case noise bound for the output ciphertext: $\frac{p^3}{2} \cdot (\mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1} + \mathcal{E}_{KS}^{n,N}) + \mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1}$.
- For ComBoMV, we follow the decomposition $f_{\text{odd,neg}} \circ \text{Id}_{\text{neg}} + f_{\text{even,neg}} \circ \text{abs}_{\text{neg}}$ given in Section 4.4, and use a multi-value bootstrapping to compute Id_{neg} and abs_{neg} at the same time. As such, the error rate becomes independent from the computed function.

Set		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Half-Torus		34	28	32	20	36	23	39	25
TOTA		33	27	30	18	34	20	36	22
FDFB	**Worst**	7	3	3	1	3	1	3	1
	Id	55	27	31	11	34	13	35	14
	ReLU	55	27	31	11	34	13	35	14
ComBo		116	85	97	50	108	56	116	61
ComBoP		116	85	97	50	108	56	116	61
ComBoMV		46	21	23	8	26	9	29	10

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

In Table 5, we show that for any given set of parameters, the probability of error is almost identical between TOTA and Half-Torus, or slightly in favor of the latter. Meanwhile, ComBo and ComBoP outperform the other methods in every case by at least 30 orders of magnitude.

We notice that FDFB and ComBoMV do not behave in the same fashion as the other methods with respect to changes in parameters:

- They favorably compare to the others when the noise of the input ciphertext is small compared to V_r, as in set 1 where ComBoMV reaches an error rate of 2^{-46} while the Half-Torus method reaches an error rate of 2^{-34}. In these cases, the overhead of the noise created by the intermediary polynomial multiplication is absorbed by V_r.
- They unfavorably compare to the other methods when V_r is small compared to the noise of the input ciphertext, as in set 8 where ComBoMV reaches an error rate of 2^{-10} while the Half-Torus method reaches an error rate of 2^{-25}.

In addition, for FDFB, the specific values of the polynomial ($P_2 - P_1$ from Section4.2) also have to be taken into account when trying to gauge whether the parameters are favorable or not towards FDFB use. Indeed, in simple cases such as the ReLU and Id functions, we can see a huge improvement (from 2^{-7} to 2^{-55} for the set 1) compared to the worst case approximation for FDFB.

6.3 Time Performance

The Half-Torus method is the fastest as it requires one BlindRotate. Then, TOTA is slightly faster than FDFB as it requires less KeySwitch operations. It is also on par with ComBoP

as the parallelism overhead is negligible. As far as the ComBo 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 bootstrappings. So, a sequential execution of ComBo becomes twice slower than TOTA and FDFB. Note however that these latter methods are intrinsically sequential. As such, they cannot outperform ComBoP.

As a bonus, we obtain a rule of thumb to get the computation time of each functional bootstrapping method. Indeed, multiplying the computation time of one bootstrapping with the number of consecutive BlindRotate gives accurate estimations of the result from Table 6. We remind that the computation time of one bootstrapping is almost equal to the time required to run to 1 BlindRotate plus 1 KeySwitch.

Set		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Half-Torus		135.0	126.1	101.4	94.6	97.4	84.5	85.5	72.0
TOTA		274.7	252.4	209.3	189.3	194.9	169.1	174.3	147.9
FDFB		287.0	268.1	220.5	203.2	207.4	181.2	182.8	157.8
ComBo	**abs**	136.5	126.0	104.9	94.6	97.5	84.5	87.0	74.2
	generic	551.5	503.6	417.7	378.0	389.6	337.5	341.4	296.5
ComBoP		273.6	258.8	211.1	200.1	205.3	182.1	183.3	153.5
ComBoMV		419.0	386.2	319.7	290.9	299.0	260.1	262.0	224.6

Table 6: Computation time in ms

Another way of showing ComBoP advantages is to compute the time performance of each method given their *own* optimized parameter set with respect to the same target error rate and plaintext space of size p. When doing so, we get the following example results with a target error rate of 2^{-32}:

- **p=4:** We achieve a speed up of x1.04 versus TOTA, x1.1 versus FDFB (ReLU) and x2 versus FDFB (worst case).
- **p=8:** We achieve a speed up of x1.09 versus TOTA, x1.12 versus FDFB (ReLU) and x4 versus FDFB (worst case).
- **p=16:** We achieve a speed up of x1.12 versus TOTA, x1.4 versus FDFB (ReLU) and x2 versus FDFB (worst case).

Besides, ComBo, ComBoP and ComBoMV are the only method allowing for parameters using $N=1024$ when $p=16$. This lead to ciphertexts twice smaller in this specific case, which is another important metric for FHE computations.

6.4 Wrapping-up: Time-Error Trade-offs

We summarize the trade-offs between the computation time and the error rate for each method in Figure 2 and Figure 3. We separate the sets defined in Table 4 in order to have better readability of the figures.

For FDFB, we represent both the worst case and the ReLU, which is the best case among the functions we considered. For ComBo, ComBoMV and ComBoP methods, the

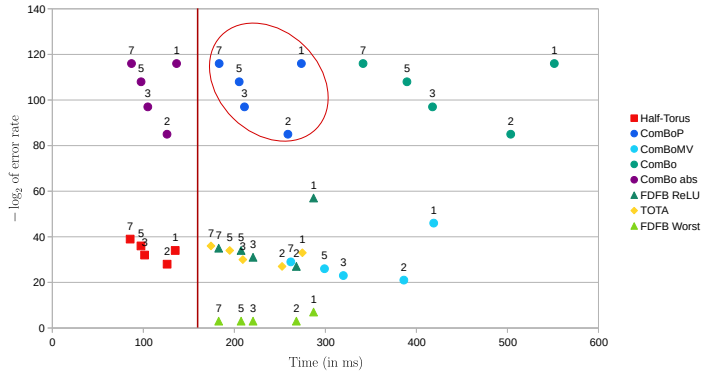

Fig. 2: Time-Error trade-off for parameters 1,2,3,5 and 7

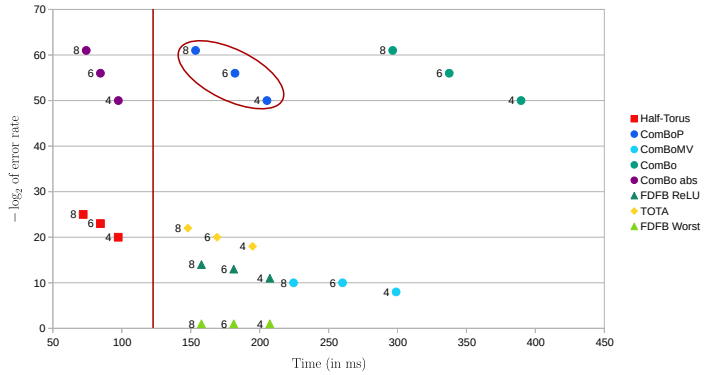

Fig. 3: Time-Error trade-off for parameters 4,6 and 8

best case is represented with the absolute value function and noted ComBo abs. The ComBo, ComBoMV and ComBoP points are all relative to a generic function following the pseudo even and pseudo odd decomposition from Section 4.4.

Fast operations will result in having points closer to the left. Meanwhile, a low error rate corresponds to points close to the upper parts of the graphs from Figures 2 and 3. With those two considerations in mind, we notice that the only methods on the left of the red line are the Half-Torus method and ComBo in the best case scenario. In this specific scenario, the ComBo method is the best in all regards. For functions requiring more bootstrappings, a compromise between speed and error rate must be made. In the red circle lies the points relative to the ComBoP 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.

7 Conclusion

Through the use of several bootstrappings and, most of the time, additional operations, every full domain method adds some output noise when compared to the partial domain method (Section 4.1). So the bottom line is: does a larger initial plaintext space make up for the added noise and computation time?
Table 5 and Table 6 confirm that the Yan et al., [32] (TOTA) method is both less accurate and twice as time-consuming than the partial domain method. Both Kluczniak and Schild's [27] (FDFB) and ComBoP methods provide a better accuracy than the partial domain method for well chosen parameters with varying additional computational costs.

Among the above full-domain methods, ComBoP achieves the best performance and accuracy. Furthermore, it outperforms the partial domain method in the following cases:

- The parameters of the cryptosystem are limited due to application constraints and the error rate of the Half-Torus is too large.
- Intermediate operations such as additions and multiplications push messages out of the Half-Torus space.
- Modular arithmetic is needed (which is impossible with the partial domain method).

When none of the above applies, however, the Half-Torus bootstrapping method still achieves better performances. This illustrates the fact, that there is no universal best method for functional bootstrapping and that one should carefully choose the most appropriate one depending on his or her application constraints. This paper's methodology and unified analysis gives a complete set of tools for making these choices.

ComBo (Section 4.4) has a smaller error rate than any other method available in the literature. In addition, as it allows to perform two bootstrappings in parallel, it may come without additional computational cost compared to the other full domain methods which are intrinsically serial. As such, ComBoP appears especially well adapted to benefit from the SIMD instruction sets available in modern processors. Furthermore, ComBo is particularly suited to homomorphic evaluation of functions such as ReLU, one of the key building-blocks for enabling advanced deep learning functions over encrypted data at larger scale.

References

[1] Abbass Madi et al. "A Secure Federated Learning framework using Homomorphic Encryption and Verifiable Computing". In: 2021, pp. 1–8. DOI: 10.1109/RDAAPS48126.2021.9452005.
[2] Loris Bergerat et al. *Parameter Optimization & Larger Precision for (T)FHE*. Cryptology ePrint Archive, Paper 2022/704. 2022.
[3] Pengtao Xie et al. "Crypto-Nets: Neural Networks over Encrypted Data". In: *CoRR* (2014).
[4] Martin R. Albrecht, Rachel Player, and Sam Scott. *On the concrete hardness of Learning with Errors*. Cryptology ePrint Archive, Paper 2015/046. 2015.

[5] 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.

[6] 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.

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

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

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

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

[15] 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.

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

[17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. *TFHE: Fast Fully Homomorphic Encryption Library*.

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

[19] 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.

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

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

[22] 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.

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

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

[25] 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.

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

[27] 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.

[28] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. "Large-Precision Homomorphic Sign Evaluation Using FHEW/TFHE Bootstrapping". In: *Advances in Cryptology – ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part II*. Taipei, Taiwan: Springer-Verlag, 2023, pp. 130–160. ISBN: 978-3-031-22965-7. DOI: `10.1007/978-3-031-22966-4_5`.

[29] 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.

[30] 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`.

[31] 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.

[32] 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.

[33] 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.

[34] 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.