TFHE Parameter Setup for Effective and Error-Free Neural Network Prediction on Encrypted Data *

Jakub Klemsa

Czech Technical University in Prague, Prague, Czechia,
jakub.klemsa@fel.cvut.cz

Abstract. With the rise of Cloud and Big Data technologies, Machine Learning as a Service (MLaaS) receives much attention, too. However, in some sense, the current situation resembles the era of wild digitalization, where security was pushed to the sideline. Back then, the problem was mostly about misunderstanding of severe consequences that insecure digitalization might bring. To date, common awareness of security has improved significantly, however, currently we are facing rather a *technological* challenge. Indeed, we are still missing a *competitive* and *satisfactory* solution that would secure MLaaS.
In this paper, we contribute to the very recent line of research, which utilizes a Fully Homomorphic Encryption (FHE) scheme by Chillotti et al. named TFHE. It has been shown that TFHE is particularly suitable for securing MLaaS. In addition, its security relies on the famous LWE problem, which is considered quantum-proof. However, it has not been stuied yet how all the TFHE parameters are to be set. Hence we provide a thorough analysis of error propagation through TFHE homomorphic computations, based on which we derive constraints on the parameters as well as we suggest a convenient representation of internal objects. We particularly focus on effective resource utilization in order to achieve the best performance of any prospective implementation.

Keywords: Privacy in the Cloud, Machine Learning, Fully Homomorphic Encryption, Post-Quantum Cryptography

1 Introduction

There are many Cloud service providers that offer a platform for easy-to-mount MLaaS. Based on Xie et al. [1], there are at least Google [2], Microsoft [3], GraphLab (now Turi) [4] or Ersatz Lab [5], who were offering MLaaS commercially already in 2014, many new providers have joined since then.

However, the main concern, which Xie et al. study in their paper and which has been put forward by Graepel et al. [6] in 2012, is *user data privacy* with

* This work was supported by the Grant Agency of CTU in Prague, grant No. SGS19/109/OHK3/2T/13.

respect to MLaaS. On the one hand, MLaaS users not only need not maintain the servers or prediction models, they can even process their data on a commercial model, which can be kept confidential to the users. On the other hand, in the traditional approach, encryption is only employed for data transmission. I.e., the sensitive user data gets decrypted before it enters the prediction model at the server. Besides the fact that such a trust model requires absolute confidence of the user in the cloud, it may also violate some legal restrictions (e.g., GDPR [7] in the EU).

Recently, multiple approaches how a prediction model can be evaluated on encrypted (and never decrypted) user data with existing cryptographic tools have emerged. Approaches by Graepel et al. [6] or by Xie et al. [1] employ so called *Leveled Homomorphic Encryption* (LHE), which limits the depth of non-linear operations in the evaluated circuit. That might be a problem in particular in case of still more popular *Deep Neural Networks* (DNN's), which may contain even thousands of layers, where non-linear operations need to be evaluated. This issue has been addressed by Bourse et al. [8], who suggest to employ the TFHE scheme by Chillotti et al. [9], which does not limit the number of non-linear operations. TFHE—with slight modification—allows to perform addition and evaluate a function, both on encrypted data. Recently, a different approach for privacy-preserving DNN evaluation has been proposed by Tillem et al. [10], who suggest to employ *Additive Homomorphic Encryption* (AHE) in combination with secure *Multi-Party Computations* (MPC's) in a form of an interactive protocol, which they call *SwaNN*.

We decided to build upon the approach by Bourse et al., i.e., we aim to employ an extended variant of TFHE for DNN evaluation on encrypted data. First, we find useful that the protocol is *non-interactive*, which might be an advantage, e.g., in the world of IoT. We also spot a potential in prospective generic usage of (the extended) TFHE.

(Extended) TFHE in Brief. TFHE is a recent FHE scheme, which builds upon the famous *Learning With Errors* (LWE) problem introduced by Regev [11], who also discusses its quantum hardness. In its original form, TFHE encrypts a single bit, however, an extension by Carpov et al. [12] introduces multivalue plaintext space. Homomorphic properties of multivalue TFHE can be written as follows:

$$\mathsf{TFHE}(a) \oplus \mathsf{TFHE}(b) \approx \mathsf{TFHE}(a+b), \text{ and} \tag{1}$$

$$\mathsf{eval}_f\big(\mathsf{TFHE}(a)\big) \approx \mathsf{TFHE}\big(f(a)\big), \tag{2}$$

where $\approx$ means "encrypts the same", and $\oplus$ and eval_f stand for particular homomorphic addition and function evaluation algorithms, respectively. Note that TFHE is a randomized encryption scheme, hence each time a value is encrypted, it might output a different ciphertext.

In addition to randomized ciphertexts, TFHE also adds certain amount of noise to plaintext representation. With each $\oplus$ operation, the internal noise grows additively. After the noise exceeds certain bound, correct decryption may not be guaranteed.

The cornerstone of TFHE (as well as many other FHE schemes) is a procedure referred to as *bootstrapping*, which aims at reducing the noise under certain fixed bound. Since the first-ever FHE scheme by Gentry [13] from 2009, bootstrapping runs internally the decryption procedure with encrypted key bits, referred to as the *bootstrapping keys*. Most importantly for multivalue TFHE, its bootstrapping is capable of function evaluation with no overhead, hence eval_f in (2) is in fact bootstrapping.

Our Contributions. Due to its noisy nature, TFHE can be tuned by its parameters to work more or less precisely. On the one hand, a few errors might pose only a little problem, e.g., in case of approximate classification as presented by Bourse et al. [8]. On the other hand, in case we need to guarantee the output correctness, multivalue TFHE must process the data in an error-free manner. This can be found useful, e.g., in case we evaluate a certified DNN – any error is definitely undesirable. We also find convenient to know where the edge of error-free evaluation resides.

Prior works [8, 12] discuss or employ multivalue TFHE, however, they do not provide any hint how all the TFHE parameters are to be set, nor suggest how the evaluated bootstrapping function is to be encoded in order to guarantee error-free homomorphic operations on given multivalue plaintext space. In this paper, we primarily focus on this goal: namely, we thoroughly analyze error propagation through the whole DNN evaluation process, we suggest to use three different precision levels for underlying structures, and we propose how the evaluated function is to be encoded properly. In addition, we suggest a simplification of the bootstrapping procedure, which might be useful in prospective FPGA implementations, and we also fix some minor bugs.

Paper Outline. Section 2 serves as a brief yet exhaustive summary of TFHE algorithms with their recent improvements, including our observations. Next, it outlines neural network evaluation on encrypted data. It also provides many technical details that will be referenced in the rest of the paper. In Section 3, we carefully analyze error propagation through TFHE in order to guarantee correct decryption. We also suggest internal representation of some TFHE variables as well as we propose a simplification to the TFHE bootstrapping algorithm. We conclude our paper in Section 4.

Symbols & Notation. Since this paper has a lot of technical content, we introduce frequent symbols and notations at single place, details will be given later. We denote:

- $\mathbb{B}$ the binary Galois field GF_2, $\mathbb{T}$ the set $\mathbb{R}/\mathbb{Z}$ referred to as the *torus*,
- $M^{(N)}[X]$ the set of polynomials modulo $X^N + 1$, for a set M and $N \in \mathbb{N}_0$,
- for a polynomial $p(X) = p^{(0)} + p^{(1)}X + p^{(2)}X^2 + \ldots + p^{(N-1)}X^{N-1}$, we denote $\mathsf{coeffs}(p) = (p^{(0)}, p^{(1)}, \ldots, p^{(N-1)})$,

- for a vector of polynomials $\mathbf{w} = (w_0^{(0)} + w_0^{(1)}X + \ldots + w_0^{(N-1)}X^{N-1}, \ldots, w_{n-1}^{(0)} + w_{n-1}^{(1)}X + \ldots + w_{n-1}^{(N-1)}X^{N-1})$, we denote
  - $w_i(X) = w_i^{(0)} + w_i^{(1)}X + \ldots + w_i^{(N-1)}X^{N-1}$, and
  - $\mathbf{w}^{(j)} = (w_0^{(j)}, \ldots, w_{n-1}^{(j)})$,
- $a \xleftarrow{\$} M$ and $a \xleftarrow{\alpha} M$ the uniform and the zero-centered α-deviated draw, respectively, of a random variable a from M.

2 Fully Homomorphic Encryption and Neural Networks

In this section, we revisit TFHE by Chillotti et al. [9] as well as subsequent enhancements [8, 12]. In particular, we focus on its usage with neural networks, which we will refer to as WTFHE (*neural-net**W**ork-ready Torus Fully Homomorphic Encryption*; [14]). We explain the (modified) bootstrapping algorithms in closer detail as a prerequisite for parameter derivation in the following sections. Finally, we outline homomorphic neural network evaluation with WTFHE.

The Torus and Concentrated Distribution. We call $\mathbb{T} = \mathbb{R}/\mathbb{Z}$—real numbers modulo 1—with standard addition operation the *torus*. Torus forms a module over $\mathbb{Z}$, i.e., we can multiply its elements by integers yielding torus elements. The operation can be extended to integer-torus polynomials.

Unlike multiplication, torus division by an integer cannot be defined uniquely, same holds for expectation of a distribution on the torus. However, this can be fixed for a *concentrated distribution* [9], which is a distribution with support limited to a ball of radius $1/4$, up to a negligible amount. For further technical details, we refer the reader to [9].

2.1 (W)TFHE Samples

(W)TFHE is a fully homomorphic cipher, which employs internally two encryption schemes: T(R)LWE and TRGSW. It uses TLWE to encrypt its plaintexts—i.e., the *global* encryption function—, while TRGSW and TRLWE are used internally within the bootstrapping procedure. In our paper, we simplify and unify notation across papers (we follow the updated notation from [9], which was introduced at the end of Section 3 of that paper).

T(R)LWE.

Definition 1 (T(R)LWE Sample). *Let $n \in \mathbb{N}$ be the* dimension*, $N \in \mathbb{N}$, $N = 2^\nu$ for some $\nu \in \mathbb{N}_0$, be the* degree*, $\alpha \in \mathbb{R}_0^+$* standard deviation *and let the plaintext space $\mathcal{P} = \mathbb{T}^{(N)}[X]$, the ciphertext (sample) space $\mathcal{C} = \mathbb{T}^{(N)}[X]^{n+1}$ and the key space $\mathcal{K} = \mathbb{B}^{(N)}[X]^n$. For $m \in \mathcal{P}$, we call $\mathbf{c} = (\mathbf{a}, b)$ the* TRLWE sample *of message m with key $\mathbf{k} \in \mathcal{K}$ if*

$$b = m + \mathbf{k} \cdot \mathbf{a} + e, \tag{3}$$

where $\mathbf{a} \xleftarrow{\$} \mathbb{T}^{(N)}[X]^n$ *and* $e \xleftarrow{\alpha} \mathbb{T}^{(N)}[X]$. *Further, for* $\mathbf{a} = \mathbf{0}$, *we call the sample* trivial, *for* $m = \mathbf{0}$, *we call the sample* homogeneous, *and for* $N = 1$, *we call the sample the* TLWE sample.

Note that TRLWE sampling is actually *encryption*. For *decryption*, we apply TRLWE *phase* function (followed by rounding if applicable), definition follows.

Definition 2 (T(R)LWE phase). *Let* n, N *and* α *be the* TRLWE *parameters as per Definition 1, and let* $\mathbf{c} = (\mathbf{a}, b)$ *be a* TRLWE *sample of* m *under a* TRLWE *key* $\mathbf{k}$. *We call the function* $\varphi_{\mathbf{k}} \colon \mathbb{T}^{(N)}[X]^k \times \mathbb{T}^{(N)}[X] \to \mathbb{T}^{(N)}[X]$,

$$\varphi_{\mathbf{k}}(\mathbf{a}, b) = b - \mathbf{k} \cdot \mathbf{a}, \tag{4}$$

the TRLWE phase. *Next, we call the sample* $(\mathbf{a}, b)$ valid *iff the distribution of* $\varphi_{\mathbf{k}}(\mathbf{a}, b)$ *is concentrated. Finally, we denote* $\mathsf{msg}(\mathbf{c}) \coloneqq \mathrm{E}\big[\varphi_{\mathbf{k}}(\mathbf{c})\big]$ *the* message *of sample* $\mathbf{c}$, *which equals* m *for a valid sample, since the noise is zero-centered.*

In general, the TRLWE phase function returns $m + e$, i.e., the plaintext with a (zero-centered) noise. TRLWE decryption can be understood as either:

1. an erroneous decryption via TRLWE phase – we accept some error in our decrypted results, which can be considered useful, e.g., in the context of differential privacy [15], or
2. a correctable decryption – for this purpose, we need to control the amount of noise and follow the TRLWE phase by appropriate rounding (the main focus of this paper), or
3. an expectation of the TRLWE phase (i.e., $\mathsf{msg}(\mathbf{c})$) – this is useful for formal definitions and proofs.

Next, we state the additively homomorphic property.

Theorem 1 (Additive Homomorphism [9]). *Let* $\mathbf{c}_1, \ldots, \mathbf{c}_n$ *be valid and independent* TRLWE *samples under key* $\mathbf{k}$ *and let* $e_1, \ldots, e_n \in \mathbb{Z}^{(N)}[X]$ *be integer polynomials. In case* $\mathbf{c} = \sum_{i=1}^{n} e_i \cdot \mathbf{c}_i$ *is a valid* TRLWE *sample, it holds*

$$\mathsf{msg}\Big(\sum_{i=1}^{n} e_i \cdot \mathbf{c}_i\Big) = \sum_{i=1}^{n} e_i \cdot \mathsf{msg}(c_i) \tag{5}$$

and the noise amplitude is bounded by

$$\|\mathsf{Err}(\mathbf{c})\|_\infty \leq \sum_{i=1}^{n} \|e_i\|_1 \cdot \|\mathsf{Err}(\mathbf{c}_i)\|_\infty. \tag{6}$$

TRGSW. Unlike torus polynomials in TRLWE, TRGSW encrypts integer polynomials. For the purposes of bootstrapping, it defines so called *External Product*, $\boxdot$: TRGSW $\times$ TRLWE $\to$ TRLWE, which is multiplicatively homomorphic on TRGSW $\times$ TRLWE samples. Definitions follow.

Definition 3 (Gadget Matrix [9]). *Let $B_g = 2^\gamma$ for some $\gamma \in \mathbb{N}$ and $l \in \mathbb{N}$ be decomposition parameters and let N and n be* TRLWE *degree and dimension, respectively. We call*

$$\mathbf{H} = \left(\begin{array}{c|c|c} 1/B_g & \dots & 0 \\ \vdots & \ddots & \vdots \\ 1/B_g^l & \dots & 0 \\ \hline \vdots & \ddots & \vdots \\ \hline 0 & \dots & 1/B_g \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1/B_g^l \end{array}\right), \tag{7}$$

$\mathbf{H} \in \mathbb{T}^{(N)}[X]^{(n+1)l,n+1}$*, the* gadget matrix*.*

Next, we recall the *Gadget Decomposition Algorithm* as Algorithm 1, which is—in this particular form—entangled with the gadget matrix $\mathbf{H}$.

Algorithm 1 Gadget Decomposition of a TRLWE Sample [9]
(for gadget matrix $\mathbf{H}$, quality $\beta = B_g/2$ and precision $\varepsilon = 1/2B_g^l$)

Input: TRLWE sample $(\mathbf{a}, b) = \big(a_1(X), \dots, a_k(X), b = a_{n+1}(X)\big) \in \mathbb{T}^{(N)}[X]^{n+1}$,
Input: decomposition parameters B_g, l.
Output: Vector of integer polynomials $\mathbf{d} \in \mathbb{Z}^{(N)}[X]^{(n+1)l}$.

1: **for all** $a_i(X) = \sum_{j=0}^{N-1} a_i^{(j)} X^j$, $a_i^{(j)} \in \mathbb{T}$, **do**
2: $\quad \bar{a}_i^{(j)} \leftarrow \lfloor B_g^l \cdot a_i^{(j)} \rceil$
3: $\quad$ let $[\bar{a}_{i,1}^{(j)}, \dots \bar{a}_{i,l}^{(j)}]$ be a B_g-ary representation of $\bar{a}_i^{(j)}$ s.t. $\bar{a}_i^{(j)} = \sum_{p=1}^{l} \bar{a}_{i,p}^{(j)} B_g^{l-p}$
4: **for** $i = 1 \dots n+1$ **and** $p = 1 \dots l$ **do**
5: $\quad d_{(i-1)l+p}(X) = \sum_{j=0}^{N-1} \bar{a}_{i,p}^{(j)} X^j$
6: **return** $\mathbf{d}$

Note 1. For the gadget matrix $\mathbf{H}$, quality $\beta = B_g/2$ and precision $\varepsilon = 1/2B_g^l$, we denote the gadget decomposition algorithm as $\mathsf{Dec}_{\mathbf{H},\beta,\varepsilon}(\mathbf{a}, b)$.

Theorem 2 (Quality and Precision of Gadget Decomposition [9]). *Algorithm 1 outputs $\mathbf{d} \in \mathbb{Z}^{(N)}[X]^{(n+1)l}$ such that $\|\mathbf{d}\|_\infty \leq \beta$ and $\|\mathbf{d} \cdot \mathbf{H} - (\mathbf{a}, b)\|_\infty \leq \varepsilon$.*

Definition 4 (TRGSW Sample [9]). *Let n, N and α be the parameters of* TRLWE *with key $\mathbf{k}$. We call $\mathbf{C} = \mathbf{Z} + m \cdot \mathbf{H}$ the* TRGSW *sample of $m \in \mathbb{Z}^{(N)}[X]$ if each row of $\mathbf{Z}$ is an independent homogeneous* TRLWE *sample under key $\mathbf{k}$, and we call m the message of $\mathbf{C}$, denoted $\mathsf{msg}(\mathbf{C})$. The phase of $\mathbf{C}$ is defined as the vector of the $(n+1)l$* TRLWE *phases, denoted $\varphi_{\mathbf{k}}(\mathbf{C})$, and the error of $\mathbf{C}$ is defined as the vector of the $(n+1)l$* TRLWE *errors, denoted $\mathsf{Err}(\mathbf{C})$. We call $\mathbf{C} \in \mathbb{T}^{(N)}[X]^{(n+1)l,n+1}$ a* valid TRGSW *sample under key $\mathbf{k}$ iff there exists $m \in \mathbb{Z}^{(N)}[X]$ such that each row of $\mathbf{C} - m \cdot \mathbf{H}$ is a valid homogeneous* TRLWE *sample under key $\mathbf{k}$.*

Definition 5 (External Product [9]). *For decomposition parameters β and ε, we define the* External Product*, $\boxdot$: TRGSW $\times$ TRLWE $\to$ TRLWE, as*

$$\mathbf{A} \boxdot \mathbf{b} = \mathsf{Dec}_{\mathbf{H},\beta,\varepsilon}(\mathbf{b})^T \cdot \mathbf{A}, \tag{8}$$

where TRLWE *is the underlying cipher of* TRGSW*.*

Theorem 3 (Multiplicative Homomorphism [9]). *Let $\mathbf{k}$ be a* TRLWE *key, $\mathbf{A}$ a valid* TRGSW *sample of $m_A \in \mathbb{Z}^{(N)}[X]$ under key $\mathbf{k}$, and $\mathbf{b}$ a valid* TRLWE *sample of $m_b \in \mathbb{T}^{(N)}[X]$ under the same key. Then $\mathbf{A} \boxdot \mathbf{b}$ is a* TRLWE *sample of $m_A \cdot m_b \in \mathbb{T}^{(N)}[X]$ under key $\mathbf{k}$.*

2.2 Bootstrapping

As outlined in the Introduction, *bootstrapping* aims at reducing the internal noise of a TLWE sample to some fixed level, while it runs the decryption procedure internally. In addition, it is capable of function evaluation at no extra cost. (W)TFHE bootstrapping consists of three algorithms: BlindRotate, SampleExtract and KeySwitch. In this paper, we only recall their variants, which are relevant to prospective neural network evaluation, e.g., we fix the TRGSW dimension, which will no longer be different than 1. In addition and unlike plain TFHE bootstrapping, we consider multivalue plaintext space as well as function evaluation.

In a high level overview, bootstrapping proceeds as follows. First, BlindRotate takes the bootstrapped TLWE sample and runs homomorphically a decryption-like procedure with encrypted key bits (referred to as the *bootstrapping key*). This way, it "blindly rotates" the second input – a TRLWE sample, which encodes (and encrypts) the bootstrapping function in a form of a torus polynomial.

Next, SampleExtract extracts the constant term of the TRLWE-encrypted polynomial back into a TLWE sample. Note that at this point, the sample is encrypted with a (possibly) different key, hence KeySwitch does the job and switches the key to the original one, if applicable.

Blind Rotate. BlindRotate is the cornerstone of bootstrapping, since this is where homomorphic decryption takes place, i.e., where the noise is refreshed. It inputs the bootstrapped TLWE sample $(\mathbf{a}, b)$ in a scaled and rounded integer form $(\bar{\mathbf{a}}, \bar{b}) \in \mathbb{Z}^{n+1}$ (details to come later). In accordance with TLWE decryption (phase function, cf. Definition 2), BlindRotate internally calculates

$$-\bar{m} = -\bar{b} + \sum k_i \cdot \bar{a}_i, \tag{9}$$

where k_i's are TRGSW-encrypted under key $k'(X)$, referred to as the *bootstrapping keys* and denoted by BK_i or $\mathsf{BK}_{\mathbf{k}\to k'}$. In BlindRotate, the (hidden) value $-\bar{m}$ emerges as a power of X, by which the other input—a TRLWE sample $(u, v) \in \mathbb{T}^{(N)}[X]^2$—is multiplied. The multiplicative homomorphic property is applied, hence (u, v) gets *blindly rotated*.

The (possibly trivial) TRLWE sample (u, v) encrypts a torus polynomial $tv(X)$, referred to as the *test vector*. Its torus coefficients encode the (rescaled) bootstrapping function $f\colon \mathbb{Z}_N \to \mathbb{T}$, for now and for simplicity, as

$$tv^{(k)} = f(k), \quad k \in [0, N-1]. \tag{10}$$

Note 2. When multiplied by a power of X, the polynomial coefficients rotate *negacyclically* with period $2N$. For this reason, the rest of the *actually* evaluated bootstrapping function f is a negacyclic extension of its first N values, i.e., we have rather $f\colon \mathbb{Z}_{2N} \to \mathbb{T}$ and it must hold that

$$f(N+k) = -f(k), \quad k \in [0, N-1]. \tag{11}$$

In plain domain, the following occurs at the constant term:

$$\left(X^{-\bar{m}} \cdot tv(X)\right)^{(0)} = tv^{(\bar{m} \bmod 2N)} = f(\bar{m} \bmod 2N), \tag{12}$$

i.e., there emerges the desired function value, which will be extracted by the subsequent algorithms.

Note 3. Due to (12), $\bar{a}_i$'s and $\bar{b}$ will be scaled to $\mathbb{Z}_{2N}$ as $\bar{a} = \lfloor 2Na \rceil$.

Note 4. During BlindRotate, the "old" noise is refreshed with a fresh noise, which comes from the bootstrapping keys (TRGSW-encrypted k_i's) as well as from the (possibly) encrypted test vector (i.e., it can be zero).

Enhancement of BlindRotate. Zhou et al. [16] suggest to unfold the original BlindRotate loop, which multiplies the TRLWE sample (u, v) one by one by $X^{k_i \bar{a}_i}$, cf. (9), and group the terms by two. Bourse et al. [8] further improve the technique by Zhou et al. by reducing the number of required encryptions of bootstrapping keys from 4 to 3 (per pair of key bits).

For pairs (k, k') and (a, a') of consecutive elements of vectors $\mathbf{k}$ and $\mathbf{a}$, respectively, they write

$$X^{ka+k'a'} = kk'(X^{a+a'} - 1) + k(1-k')(X^a - 1) + (1-k)k'(X^{a'} - 1) + 1. \tag{13}$$

I.e., their bootstrapping keys consist of TRGSW encryptions of kk', $k(1-k')$ and $(1-k)k'$ for each pair of bits of the global TLWE key $\mathbf{k}$. Find the improved BlindRotate algorithm as Algorithm 2 (line 3 fixes the missing +ACC term in [8]).

Theorem 4 (BlindRotateIm Error). *Algorithm 2 returns a sample with error bounded as follows:*

$$\|\mathsf{Err}(\mathsf{ACC})\|_\infty \leq 6nlN\beta E_{\mathsf{BK}} + n(1+N)\varepsilon + \|\mathsf{Err}(u, v)\|_\infty. \tag{14}$$

Proof. Recall that by Definition 5, external product $\mathbf{A} \boxdot \mathbf{b} = \mathsf{Dec}(\mathbf{b})^T \cdot \mathbf{A}$, and by Definition 4, TRGSW sample $\mathbf{A} = \mathbf{Z}_\mathbf{A} + m_\mathbf{A} \cdot \mathbf{H}$. We write line 3 of Algorithm 2

Algorithm 2 BlindRotateIm ([9], improved by [8, 16])

Input: $(\bar{\mathbf{a}}, \bar{b}) \in (\mathbb{Z}_{2N})^{n+1}$ (scaled and rounded TLWE sample under key $\mathbf{k} \in \mathbb{B}^n$),
Input: (possibly trivial) TRLWE sample $(u, v) \in \mathbb{T}^{(N)}[X]^2$ of $tv \in \mathbb{T}^{(N)}[X]$ under key $k'(X) \in \mathbb{B}^{(N)}[X]$, and
Input: for $i \in [1, n/2]$, TRGSW samples BK_{3i-2}, BK_{3i-1} and BK_{3i} of $k_{2i-1}k_{2i}$, $k_{2i-1}(1-k_{2i})$ and $(1-k_{2i-1})k_{2i}$, respectively, under key $k'(X)$, where $(k_1, \ldots, k_n) = \mathbf{k}$ (aka. *bootstrapping keys*).
Output: TRLWE sample of $X^{-\bar{m}} \cdot tv$ under key $k'(X)$, where $\bar{m} = \left(\bar{b} - \sum_{i=1}^{n} k_i \cdot \bar{a}_i\right) \bmod 2N$.

1: $\mathsf{ACC} \leftarrow X^{-\bar{b}} \cdot (u, v)$ // aka. *accumulator*
2: **for** $i \in [1, n/2]$ **do**
3: $\quad \mathsf{ACC} \leftarrow \left((X^{a_{2i-1}+a_{2i}} - 1)\mathsf{BK}_{3i-2} + (X^{a_{2i-1}} - 1)\mathsf{BK}_{3i-1} + (X^{a_{2i}} - 1)\mathsf{BK}_{3i}\right) \boxdot \mathsf{ACC} + \mathsf{ACC}$
4: **return** ACC

as (with simplified indexes as per (13))

$$\mathsf{ACC}_{\text{new}} = \underbrace{\left((X^{a+a'} - 1)\mathsf{BK}_{kk'} + (X^{a} - 1)\mathsf{BK}_{k(1-k')} + (X^{a'} - 1)\mathsf{BK}_{(1-k)k'}\right)}_{\mathsf{BK}_{\Sigma}} \boxdot$$

$$\boxdot\, \mathsf{ACC} + \mathsf{ACC} = \tag{15}$$

$$= \underbrace{\mathsf{Dec}(\mathsf{ACC})^T}_{\mathbf{d}} \cdot \mathsf{BK}_{\Sigma} + \mathsf{ACC} = \ldots = \tag{16}$$

$$= \underbrace{(X^{a+a'} - 1) \cdot \left(\mathbf{d} \cdot \mathbf{Z}_{\mathsf{BK}_{kk'}} + kk' \cdot (\mathbf{d} \cdot \mathbf{H})\right)}_{\text{from } (X^{a+a'} - 1)\mathsf{BK}_{kk'} \text{ term}} + \ldots + \mathsf{ACC} = \ldots \tag{17}$$

Next, by Theorem 2, decomposition precision gives $\mathsf{Dec}(\mathbf{c}) \cdot \mathbf{H} = \mathbf{c} + \boldsymbol{\varepsilon}_{\mathbf{c}}$, where $\|\boldsymbol{\varepsilon}_{\mathbf{c}}\|_\infty \leq \varepsilon$.

$$\begin{aligned}
\ldots = (X^{a+a'} - 1) \cdot \Big(&\underbrace{\mathbf{d} \cdot \mathbf{Z}_{\mathsf{BK}_{kk'}}}_{\heartsuit} + kk' \cdot \Big(\underbrace{\mathsf{ACC}}_{\clubsuit} + \underbrace{\boldsymbol{\varepsilon}_{\mathsf{ACC}}}_{\spadesuit}\Big)\Big) + \\
+ (X^{a} - 1) \cdot \Big(&\underbrace{\mathbf{d} \cdot \mathbf{Z}_{\mathsf{BK}_{k(1-k')}}}_{\heartsuit} + k(1-k') \cdot \Big(\underbrace{\mathsf{ACC}}_{\clubsuit} + \underbrace{\boldsymbol{\varepsilon}_{\mathsf{ACC}}}_{\spadesuit}\Big)\Big) + \\
+ (X^{a'} - 1) \cdot \Big(&\underbrace{\mathbf{d} \cdot \mathbf{Z}_{\mathsf{BK}_{(1-k)k'}}}_{\heartsuit} + (1-k)k' \cdot \Big(\underbrace{\mathsf{ACC}}_{\clubsuit} + \underbrace{\boldsymbol{\varepsilon}_{\mathsf{ACC}}}_{\spadesuit}\Big)\Big) + \underbrace{\mathsf{ACC}}_{\clubsuit}.
\end{aligned} \tag{18}$$

By Chillotti et al. [9], proof of Theorem 3.13:

- ♥ is bounded by $2lN\beta E_{\mathsf{BK}}$,
- ♣ carries error from the previous round and, unlike ♥, it cancels out up to one term with a unit-valued monomial (check all combinations of k, k'),
- ♠ is bounded by $(1 + N)\varepsilon$ and it appears at most once among the terms – where keys multiply to 1.

Hence we can bound the error of $\mathsf{ACC}_{\text{new}}$ as follows:

$$\|\mathsf{Err}(\mathsf{ACC}_{\text{new}})\|_\infty \leq 3 \cdot 2 \cdot 2lN\beta E_{\mathsf{BK}} + \|\mathsf{Err}(\mathsf{ACC}_{\text{old}})\|_\infty + 2 \cdot (1+N)\varepsilon. \quad (19)$$

The loop begins with $\|\mathsf{Err}(\mathsf{ACC}_0)\|_\infty = \|\mathsf{Err}(u,v)\|_\infty$ and keeps adding a constant term $n/2$-times, cf. (19), hence the result (14) follows. □

Note that the improvement by Zhou et al., further tuned by Bourse et al., aims at reducing the number of external products, which is the most demanding operation. Indeed, the original BlindRotate algorithm [9] loops full n indices (instead of only $n/2$), for which it computes

$$\mathsf{ACC} \leftarrow \mathsf{BK}_i \boxdot (X^{a_i} \cdot \mathsf{ACC} - \mathsf{ACC}) + \mathsf{ACC}, \quad (20)$$

where BK_i encrypts i-th bit of the global TLWE key $\mathbf{k}$. At this point, it is interesting to observe that (20) can be written also in a form, which resembles the improved variant (Algorithm 2, line 3), and which encrypts the same:

$$\mathsf{ACC} \leftarrow \big((X^{a_i} - 1)\mathsf{BK}_i\big) \boxdot \mathsf{ACC} + \mathsf{ACC}. \quad (21)$$

However, the fundamental difference between (20) and (21) is that the latter introduces higher noise overhead due to the $(X^{a_i} - 1)\mathsf{BK}_i$ term. Unfortunately, such a rearrangement cannot be applied to the improved variant.

Sample Extract. SampleExtract algorithm inputs the output of BlindRotate, which is a TRLWE sample – let us denote it (r, s) (previously ACC). Recall that (r, s) encrypts the desired value at the constant term of its message under the key $k'(X) \in \mathbb{B}^{(N)}[X]$. As outlined, the goal of SampleExtract it to extract the constant term in a form of a TLWE sample.

First, let us write down the constant term of the message of (r, s). After some rearrangements we get

$$m^{(0)} = s^{(0)} - \underbrace{\big(k^{(0)}, k^{(1)}, \ldots, k^{(N-1)}\big)}_{\text{new TLWE key } \mathbf{k}' = \mathsf{coeffs}(k')} \cdot \big(r^{(0)}, -r^{(N-1)}, \ldots, -r^{(1)}\big). \quad (22)$$

It follows that $\big((r^{(0)}, -r^{(N-1)}, \ldots, -r^{(1)}), s^{(0)}\big)$ is a TLWE sample, which encrypts $m^{(0)}$ under the key $\mathbf{k}' = \mathsf{coeffs}(k')$. SampleExtract algorithm follows as Algorithm 3 (a slightly modified version of [9]).

Algorithm 3 SampleExtract ([9], modified)

Input: TRLWE sample $(r, s) \in \mathbb{T}^{(N)}[X]^2$ of $m(X) \in \mathbb{T}^{(N)}[X]$ under $k'(X) \in \mathbb{B}^{(N)}[X]$,
Output: TLWE sample of $m^{(0)}$ under $\mathbf{k}' = \mathsf{coeffs}(k')$.

1: **return** $(\mathbf{a}', b') = \big((r^{(0)}, -r^{(N-1)}, \ldots, -r^{(1)}), s^{(0)}\big)$

Note 5. In case we use the TRGSW key $k'(X)$ such that $\mathsf{coeffs}(k') = \mathbf{k}$, we can skip key switching. N.b., in such a case, the bit security of k' equals to that of $\mathbf{k}$, which is typically shorter, hence this must be taken into account with respect to security. The possibility of omitting KeySwitch will be thoroughly discussed in Section 3.4.

Note 6. SampleExtract preserves the error – it only re-arranges the coefficients.

Key Switching. KeySwitch algorithm inputs a TLWE sample $(\mathbf{a}', b')$ and its goal is to change the encryption key from $\mathbf{k}'$ back to $\mathbf{k}$. For this purpose, the algorithm inputs a series of TLWE encryptions of fractions of $\mathbf{k}'$'s bits referred to as the *key switching keys*, denoted by $\mathsf{KS}_{\mathbf{k}' \to \mathbf{k}}$. In Algorithm 4, we recall the original KeySwitch algorithm.

Algorithm 4 KeySwitch ([9])

Input: TLWE sample $(\mathbf{a}', b') \in \mathbb{T}^{N+1}$ of m under $\mathbf{k}' \in \mathbb{B}^N$,
Input: for $i \in [1, N]$, $j \in [1, t]$ (t is a precision parameter), TLWE samples $\mathsf{KS}_{i,j}$ of $2^{-j} \cdot k'_i$ under key $\mathbf{k} \in \mathbb{B}^n$, where $(k'_1, \ldots, k'_N) = \mathbf{k}'$ (aka. *key switching keys*).
Output: TLWE sample of m under key $\mathbf{k}$.

1: $\bar{a}'_i \leftarrow \lfloor 2^t a'_i \rceil$ for $i \in [1, N]$
2: let $[\bar{a}'_{i,1}, \bar{a}'_{i,2}, \ldots, \bar{a}'_{i,t}]$ be a binary representation of $\bar{a}'_i$ s.t. $\bar{a}'_i = \sum_{j=1}^{t} \bar{a}'_{i,j} 2^{t-j}$
3: **return** $(\mathbf{a}, b) = (\mathbf{0}, b') - \sum_{i=1}^{N} \sum_{j=1}^{t} \bar{a}'_{i,j} \mathsf{KS}_{i,j}$

Theorem 5 (KeySwitch Error [9]). *Algorithm 4 returns a sample with error bounded as follows:*

$$\|\mathsf{Err}(\mathbf{a}, b)\|_\infty \leq \|\mathsf{Err}(\mathbf{a}', b')\|_\infty + tNE_{\mathsf{KS}} + 2^{-(t+1)}N. \tag{23}$$

WTFHE Bootstrapping. As outlined in Note 3, the bootstrapped TLWE sample $(\mathbf{a}, b)$ enters BlindRotate scaled and rounded, while rounding may introduce a rounding error. A lemma follows.

Lemma 1 (Pre-BlindRotate Error). *The scaled and rounded sample $(\bar{\mathbf{a}}, \bar{b})$ enters* BlindRotate *with an error bounded as follows:*

$$\left\| \mathsf{Err}\Big(\frac{\bar{\mathbf{a}}}{2N}, \frac{\bar{b}}{2N} \Big) \right\|_\infty \leq \|\mathsf{Err}(\mathbf{a}, b)\|_\infty + \underbrace{\frac{n+1}{4N}}_{E_{round}}, \tag{24}$$

where we denote the rounding error term as E_{round}.

Proof. After scaling $(\bar{\mathbf{a}}, \bar{b})$ back to torus domain by $1/2N$, rounding on multiples of $1/2N$ may change each of the $n+1$ terms of the sample $(\mathbf{a}, b)$ by $1/2 \cdot \frac{1}{2N}$. □

Note 7. The error bound (24) is the greatest one, which occurs during bootstrapping before the noise is refreshed. Therefore we denote the bound as

$$E_{max} \coloneqq \|\mathsf{Err}(\mathbf{a}, b)\|_\infty + E_{round}. \tag{25}$$

Note 8. Rounding error (scaled to $\mathbb{Z}_{2N}$) is grater than the domain precision of the evaluated bootstrapping function $f\colon \mathbb{Z}_{2N} \to \mathbb{T}$, hence we need to introduce another (and sparser) level of precision for the global WTFHE plaintexts in order to satisfy our ultimate goal – error-free computations. From now on, we denote the plaintext space bit-precision by π, i.e., the plaintext space will be $\mathbb{Z}_{2^\pi}$, while $2^\pi < 2N$. N.b., the inaccuracy in the input of the evaluated function will be thoroughly discussed in Section 3.3, where we propose proper test vector generation.

Let us finally give the WTFHE bootstrapping algorithm as Algorithm 5. N.b., at this point, we do not specify any relations between its parameters, nor do we guarantee correctness of its output.

Algorithm 5 WTFHE Bootstrapping

Input: WTFHE parameters: π (plaintext precision), n (dimension), N (TRLWE degree), γ, l and t (TRGSW decomposition and KS precision parameters),
Input: TLWE sample $(\mathbf{a}, b) \in \mathbb{T}^{n+1}$ of $m = \bar{m}/2^\pi$, $\bar{m} \in \mathbb{Z}_{2^\pi}$, under key $\mathbf{k} \in \mathbb{B}^n$,
Input: (possibly trivial) TRLWE sample $(u, v) \in \mathbb{T}^{(N)}[X]^2$ of test vector $tv \in \mathbb{T}^{(N)}[X]$, which encodes a (negacyclic) function $\bar{f}\colon \mathbb{Z}_{2^\pi} \to \mathbb{Z}_{2^\pi}$, under key $k'(X) \in \mathbb{B}^{(N)}[X]$,
Input: bootstrapping keys $\mathsf{BK}_{\mathbf{k}\to k'}$, and key switching keys $\mathsf{KS}_{\mathsf{coeffs}(k')\to\mathbf{k}}$.
Output: TLWE sample of $\bar{f}(\bar{m})/2^\pi$ under key $\mathbf{k}$.

1: $\bar{a}_i \leftarrow \lfloor 2Na_i \rceil$ for $i \in [1, n]$, $\bar{b} \leftarrow \lfloor 2Nb \rceil$
2: $(r, s) \leftarrow \mathsf{BlindRotateIm}\big((\bar{\mathbf{a}}, \bar{b}), (u, v), \mathsf{BK}_{\mathbf{k}\to k'}\big)$
3: $(\mathbf{a}', b') \leftarrow \mathsf{SampleExtract}\big((r, s)\big)$
4: **return** $\mathsf{KeySwitch}\big((\mathbf{a}', b'), \mathsf{KS}_{\mathbf{k}'\to\mathbf{k}}\big)$

Corollary 1 (Bootstrapping Error [9]). *Algorithm 5 returns a sample* $(\mathbf{a}, b)$ *with error bounded as follows:*

$$\|\mathsf{Err}(\mathbf{a}, b)\|_\infty \leq \underbrace{6nlN\beta E_{\mathsf{BK}} + n(1+N)\varepsilon + \|\mathsf{Err}(u, v)\|_\infty + tNE_{\mathsf{KS}} + 2^{-(t+1)}N}_{E_0}, \tag{26}$$

where we denoted the error bound of a freshly bootstrapped sample by E_0.

2.3 Evaluation of a Neural Network on Encrypted Data

In this section, we recall[1] how to evaluate a neural network in its simplest form as outlined by Rumelhart et al. [17], later referred to as the *deep feedforward*

[1] N.b., we extended and updated our original text from [14], including figures.

neural network. Next, we outline how it can be evaluated on encrypted input data. Note that the evaluation procedure can be applied to more complicated neural network structures, however, this is out of the scope of this paper.

Artificial Neural Networks. An *(Artificial) Neural Network* (NN) is a series of elementary building blocks referred to as *perceptrons* organized in a net structure. In this paper, we consider only the simplest structure of NN's organized in *layers*, which are evaluated one after each other.

A perceptron P on certain layer inputs k_P values from perceptrons on the preceding layer (or NN inputs) and outputs single value, possibly to several perceptrons on the subsequent layer as their respective input (or NN output). A *weight* $w_i^{(P)}$ is assigned to each (i-th) input of the perceptron P. These weights, together with the structure, define the neural network. The perceptron P evaluates its k_P input values $(v_i)_{i=1}^{k_P}$ as follows:

$$\mathsf{eval}_P(v_i)_{i=1}^{k_P} = f\Big(\sum_{i=1}^{k_P} w_i^{(P)} v_i\Big), \tag{27}$$

where f is referred to as the *activation function*; cf. Figure 1. f is a non-linear and often also an odd function with bounded image, e.g., signum or hyperbolic tangent. In general, NN's operate with real numbers, however, we will (need to) narrow their domain down to an integer interval, namely $(-2^{\pi-1}, 2^{\pi-1})$.

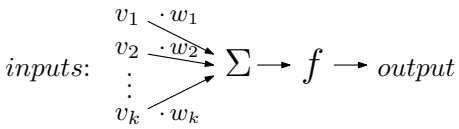


Fig. 1: Perceptron evaluation. Taken from [14].

Discretized Neural Networks. Neural networks with (bounded) integer domain are referred to as *Discretized Neural Networks* (DiNN's; Bourse et al. [8]). In their paper, they define DiNN's formally as well as they suggest an approach how to convert an already-trained real-valued NN into a discretized one. From now on, we will only consider DiNN's whenever we refer to NN's.

NN Evaluation on Encrypted Data. Since NN evaluation melts down to addition and activation function (cf. Figure 1), one can conveniently employ WTFHE, which has exactly the features we need: it has cheap addition (cf. (5)) and it is tailored to evaluate a (negacyclic) function during bootstrapping, i.e.,

$$\mathsf{eval}_{NN}^{(hom.)}\big(\mathsf{WTFHE}(v_i)\big)_{i=1}^{k_{NN}} \approx \mathsf{WTFHE}\big(\mathsf{eval}_{NN}^{(plain)}(v_i)_{i=1}^{k_{NN}}\big). \tag{28}$$

Dealing with Negacyclicity. Since the function, which is evaluated during bootstrapping, must be negacyclic, we need to face this limitation. A popular NN activation function is signum, which is negacyclic itself – up to the zero value at the negacyclic projection of $\sigma(0) = 0$. Note that this value can be avoided by appropriately bounded weights. In case we insist on a different activation function, e.g., hyperbolic tangent, we need to further limit the interval in order to avoid the unwanted negacyclic projections near zero, cf. Figure 2.

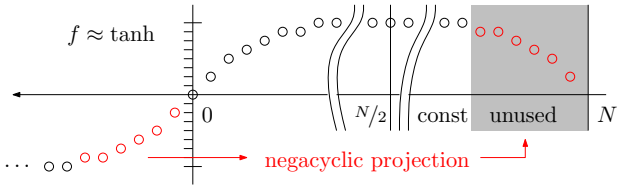


Fig. 2: Discretized and bounded hyperbolic tangent. Taken from [14].

3 WTFHE Parameter Setup for Error-Free Computations

Let us recall that the ultimate goal of our paper is to derive a set of WTFHE parameters, such that it satisfies/guarantees:

error-free computations: decryption *always* results in the expected plaintext, in particular after—appropriately limited—homomorphic operations (addition & bootstrapping),
performance: the parameters are tight, i.e., resources are not wasted due to redundancy, and
security: given bit-security level is achieved.

In this section, we provide a thorough analysis of WTFHE noise propagation, which results in a set of limitations on WTFHE parameters.

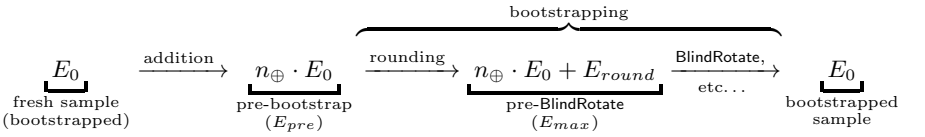
3.1 Overview of Error Propagation

During homomorphic operations, the maximum error bound evolves as follows:

ad addition: the error bound is additive, cf. (6) in Theorem 1,
ad bootstrapping: if the noise of the bootstrapped sample is smaller than certain bound, bootstrapping evaluates the bootstrapping function correctly (i.e., at correct value) as well as the bootstrapped sample carries a fixed amount of noise independent of the original sample; cf. (26) in Corollary 1.

We summarize error propagation in Figure 3, where the number of additions is bounded by $n_{\oplus}$. Note that the overall maximum of noise is achieved within bootstrapping right after the initial rounding, cf. Note 7.

Note 9. Since a freshly bootstrapped sample plays a similar role as a freshly encrypted sample, we will demand the error bound E_0 for a fresh sample, too.



(29)

Fig. 3: Error propagation during WTFHE addition and bootstrapping, cf. (24), (25) and (26). Number of additions is bounded by $n_\oplus$.

3.2 Precision Levels

Before we derive the WTFHE parameters themselves, we discuss what approach is the most suitable for representation of different WTFHE variables at various stages. E.g., sample precision must be larger than plaintext precision in order to be capable of carrying the fine-grained noise. Namely, we are interested in *plaintext precision* (π bits), *internal* TRLWE *degree* N (ν bits), and *sample precision* (τ bits), which is the finest precision throughout WTFHE, also referred to as the *torus precision*.

First, recall that in Note 8 we derived that $\nu+1 > \pi$, otherwise the rounding error would exceed the plaintext precision $1/2^\pi$, cf. (24). Next, let us discuss two possible cases for sample precision τ:

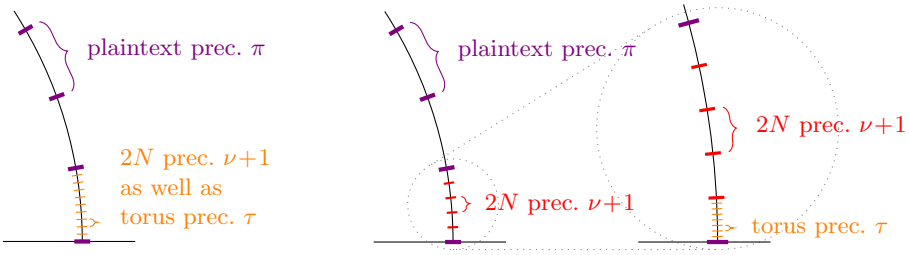
(a) $\tau = \nu + 1$ (i.e., 2-level precision), or
(b) $\tau > \nu + 1$ (i.e., 3-level precision); cf. Figure 4.

2-level Precision. By (26) in Corollary 1, the error of a freshly bootstrapped sample is bounded at least by

$$6nlN\beta E_{\mathsf{BK}} \geq 6nN\frac{B_g}{2} \cdot \frac{1}{2N} \geq \frac{3}{2}, \tag{30}$$

where we applied $\beta = B_g/2$, $E_{\mathsf{BK}} \geq 1/2N = 1/2^\tau$ (smallest possible error for given torus precision) and the fact that other parameters are integers. The error bound is even larger than the size of the torus, hence it would lead to a complete plaintext loss in most cases. It follows that it must be the case of three precision levels.

3-level Precision. With three levels of precision, we have $\tau > \nu+1 > \pi$, which is what we suggest and which will be used in the rest of this paper.



(a) 2-level precision.

(b) 3-level precision.

Fig. 4: Two- vs. three-level precision.

3.3 Test Vector Generation

After we have fixed the setup with three levels of precision, we focus on the process of encoding the (negacyclic) bootstrapping function $\bar{f}\colon \mathbb{Z}_{2^\pi} \to \mathbb{Z}_{2^\pi}$ into actual test vector, which is a torus polynomial $tv \in \mathbb{T}^{(N)}[X]$. Recall that tv comes into play in BlindRotate (Algorithm 2), which "blindly rotates" tv by $X^{-\bar{m}}$, where $\bar{m} = \bar{b} - \sum_{i=1}^{n} k_i \cdot \bar{a}_i$, $\bar{a}_i, \bar{b} = \lfloor 2Na_i, b \rceil$ and $(\mathbf{a}, b)$ is the bootstrapped TLWE sample. The error of $\bar{m}$, down-scaled by $2N$, is bounded by $E_{max} = n_{\oplus} \cdot E_0 + E_{round}$, cf. (25). Note that only $E_{round} = \frac{n+1}{4N} > \frac{1}{2N}$ is greater than the precision of down-scaled $\bar{m}$, i.e., the error of $\bar{m}$ can be greater than 1. For this reason, we suggest that the (negacyclic) bootstrapping function $\bar{f}\colon \mathbb{Z}_{2^\pi} \to \mathbb{Z}_{2^\pi}$ is expanded to a *staircase* function $f\colon \mathbb{Z}_{2N} \to \mathbb{T}$, before it is encoded into the test vector tv, as

$$f(k) = \bar{f}\Big(\Big\lfloor \frac{k}{2^{\nu+1-\pi}} \Big\rceil\Big), \quad k \in [0, 2N-1], \tag{31}$$

cf. Figure 5. N.b., the function f is indeed staircase, since we have $\nu + 1 > \pi$. The function f is then encoded into tv (as already suggested in (10)) as follows:

$$tv^{(k)} = f(k), \quad k \in [0, N-1]. \tag{32}$$

Recall that since f is negacyclic, the rest of the values does not need to be encoded, cf. Note 2.

Next, let us focus on the error bound. As soon as E_{max} is smaller than half of the plaintext resolution, i.e.,

$$\boxed{E_{max} \leq \frac{1}{2^{\pi+1}},} \tag{33}$$

the erroneous value of $\bar{m}$ does not leave the "stair", i.e., the function is evaluated as expected, cf. Figure 5.

3.4 Key Switching: Employ, or Omit?

Let us discuss two possible bootstrapping approaches: either we employ, or we omit key switching, as outlined in Note 5. Recall that key switching aims at

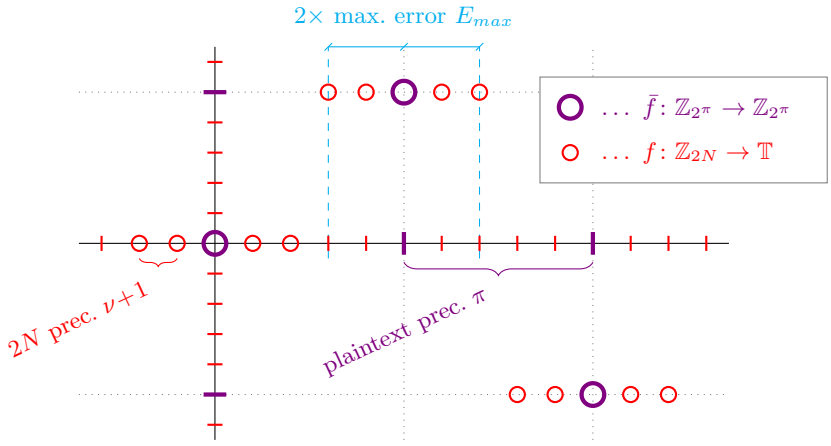


Fig. 5: Conversion of a bootstrapping function $\bar{f}$ into a staircase function f. N.b., both functions are appropriately scaled.

changing the key of a TLWE ciphertext from (the bootstrapping TRLWE key interpreted as) a TLWE key of length N to the original TLWE key of length n.

Note 10. With respect to NN evaluation, we commit on the following:

- in order to prevent overflow, we bound $n_\oplus$—the number of additions; note that we only work with encrypted ± 1 in each fresh(ly bootstrapped) sample—as follows:
$$n_\oplus \leq 2^{\pi-1}, \quad \text{and} \tag{34}$$
- since the activation function is known to the evaluating party, the test vector can be in the form of a trivial TRLWE sample, i.e., $\|\mathsf{Err}(u,v)\|_\infty = 0$.

The bound (34) can be achieved by limiting the weights of each perceptron by

$$\sum_{i=1}^{k_P} |w_i^{(P)}| \leq 2^{\pi-1}. \tag{35}$$

Note 11. Before we go through each scenario, we outline our heuristic: we suggest to split any error bound between individual error terms by equal parts. Note that this approach does not guarantee the best tradeoff.

In both scenarios, the bound (33) on E_{max} (guarantees correct decryption, E_{max} itself bounded by (29)) can be satisfied as follows:

$$E_{max} \leq \underbrace{2^{\pi-1}E_0}_{1/2^{\pi+2}} + \underbrace{E_{round}}_{1/2^{\pi+2}} \overset{!}{\leq} \frac{1}{2^{\pi+1}}, \tag{36}$$

where we applied the heuristic from Note 11.

Rounding Error. Note that the rounding error term E_{round} is independent on whether KeySwitch is employed, or not, hence it can be discussed for both scenarios together. By (24), we have

$$E_{round} \leq \frac{n+1}{4N} \overset{!}{\leq} \frac{1}{2^{\pi+2}}, \tag{37}$$

which yields

$$N \geq 2^{\pi}(n+1). \tag{38}$$

Since N must be a power of 2, we suggest to choose n as a power of 2 minus two, in order to achieve n even; cf. Algorithm 2.

Employ KeySwitch. The additive error term $2^{\pi-1}E_0$ of (36) can be estimated by (26) and bounded as follows:

$$\begin{aligned} 2^{\pi-1}E_0 \leq & \underbrace{3 \cdot 2^{\pi-1} n l N 2^{\gamma} E_{\mathsf{BK}}(N)}_{1/2^{\pi+4} \quad (\heartsuit)} + \underbrace{2^{\pi-1} n (1+N) 2^{-(\gamma l+1)}}_{1/2^{\pi+4} \quad (\blacklozenge)} + \underbrace{2^{\pi-1} \|\mathsf{Err}(u,v)\|_{\infty}}_{=0} + \\ & + \underbrace{2^{\pi-1} t N E_{\mathsf{KS}}(n)}_{1/2^{\pi+4} \quad (\clubsuit)} + \underbrace{2^{\pi-1} 2^{-(t+1)} N}_{1/2^{\pi+4} \quad (\spadesuit)} \overset{!}{\leq} \frac{1}{2^{\pi+2}}, \end{aligned} \tag{39}$$

where we applied $\beta = {}^{B_g}/_2 = 2^{\gamma-1}$ and $\varepsilon = {}^1/_{2B_g^l} = 2^{-(\gamma l+1)}$ (cf. Definition 3, Algorithm 1 and Theorem 2), and where we supported the error terms of bootstrapping and key switching keys E_{BK} and E_{KS} with their bit entropy, which is N and n, respectively. Recall that bootstrapping keys are encrypted with an N-bit TRGSW/TRLWE key $k'(X)$, which is independent from the general n-bit TLWE key $\mathbf{k}$ (it also encrypts the keyswitching keys). In logarithmic domain, we write the following set of inequalities:

$$\begin{aligned} 2\pi + 3 + \log(3) + \log(n) + \log(N) + \log(l) + \gamma + \log\big(E_{\mathsf{BK}}(N)\big) \leq 0, \quad & (\heartsuit) \\ 2\pi + 3 + \log(n) + \log(N+1) - \gamma l - 1 \leq 0, \quad & (\blacklozenge) \\ 2\pi + 3 + \log(N) + \log(t) + \log\big(E_{\mathsf{KS}}(n)\big) \leq 0, \quad & (\clubsuit) \\ 2\pi + 3 + \log(N) - t - 1 \leq 0. \quad & (\spadesuit) \end{aligned}$$

Omit KeySwitch. As outlined in Note 5, we can set the TRGSW/TRLWE key $k'(X)$ so that $\mathsf{coeffs}(k') = \mathbf{k}$ (we fill the rest with zeros) and then we can omit KeySwitch. N.b., the bit entropy of such a key is no longer N – it is only n.

Note 12. With this approach, we effectively encrypt the main TLWE key by itself – indeed, bootstrapping keys encrypt bits of the key by itself. This relies on so called *circular security assumption*; find more on circular security in Rothblum [18].

On the one hand, we get rid of the error terms (♣) and (♠) in (39), which gives us more room for the plaintext space or security. The other advantage is that

we simplify the overall bootstrapping process, which may lead to a substantial improvement in case of prospective FPGA implementation.

On the other hand, the bootstrapping keys must carry more noise, since their entropy is reduced from N bits to n bits, in order to sustain certain security level. This can be balanced by appropriate measures, e.g., by using longer TLWE keys.

Following an analogous approach to the previous one, we end up with the following set of inequalities:

$$2\pi + 2 + \log(3) + \log(n) + \log(N) + \log(l) + \gamma + \log\big(E_{\mathsf{BK}}(n)\big) \leq 0, \qquad (\heartsuit)$$
$$2\pi + 2 + \log(n) + \log(N+1) - \gamma l - 1 \leq 0, \qquad (\diamondsuit)$$

where we split the error bound into *two* parts (instead of four parts; now $1/2^{\pi+3}$ each) and where we decreased the bit entropy used for encryption of the bootstrapping keys from N to n.

3.5 Baby Parameters

Finally, we derive a set of baby parameters for testing purposes, where we only insist on $\pi = 2$ in order to have multivalue TFHE. Next, note that the bound (38) on N can be relaxed, provided that the TLWE key $\mathbf{k}$ has limited Hamming weight. Indeed, the bound can be rewritten as $N \geq 2^{\pi}\big(\|\mathbf{k}\|_1 + 1\big)$, cf. proof of Lemma 1. N.b., this is only admissible in a testing scenario. We choose $n = 4$ and $N = 16$, hence we demand $\|\mathbf{k}\|_1 \leq 3$, and we calculate the remaining parameters based on inequalities presented in Section 3.4. For minimal demonstration purposes, we have chosen not to employ KeySwitch, i.e., we applied $(\heartsuit)$ and $(\diamondsuit)$. Find the baby parameters in Table 1. We encourage the reader to test these parameters with the WTFHE Library [14].

$\pi = 2$	$n = 4$	$N = 16$	$\|\mathbf{k}\|_1 \leq 3$
$\gamma = 3$	$l = 4$	$\tau = 20$	$\|\mathsf{BK}\| = 3.75\,\mathrm{KiB}$

Table 1: Baby parameters for $\pi = 2$ bits of plaintext precision.

4 Conclusion

We addressed the problem of parameter setup, bootstrapping function encoding and representation of (W)TFHE variables in order to guarantee correctness of the homomorphic evaluation procedure. We derived a bound, under which expected results are guaranteed. We find the bound particularly useful for the evaluation of certified DNN's on encrypted data. The bound can also serve as a starting point for possible parameter relaxations. Besides that, we suggested a simplified bootstrapping scenario, which omits the KeySwitch algorithm. Last but not least, due to its comprehensivity, our paper can serve as a good starting point for a prospective (W)TFHE implementation.

Discussion & Future Directions. Althouth a practical privacy-preserving MLaaS would find its application speedily, it appears that the research still has a long way to go. Nevertheless, we still believe in the approach using (W)TFHE.

Our next goal is to derive and fine-tune a set of—possibly relaxed—real-world parameters, which we plan to utilize for an FPGA implementation of (W)TFHE. We also aim to run a benchmark to compare both bootstrapping scenarios (with, or without KeySwitch) in order to identify the faster one in a real-world setting.

References

1. Xie, P., Bilenko, M., Finley, T., Gilad-Bachrach, R., Lauter, K., Naehrig, M.: Crypto-nets: Neural networks over encrypted data. arXiv preprint arXiv:1412.6181 (2014)
2. Cloud, G.: AI Platform. https://cloud.google.com/ai-platform
3. Microsoft: Azure Machine Learning. http://azure.microsoft.com/en-us/services/machine-learning/
4. Turi: Graphlab Create™. https://turi.com/
5. Ersatzlabs: AI & Deep Learning. http://www.ersatzlabs.com/
6. Graepel, T., Lauter, K., Naehrig, M.: Ml confidential: Machine learning on encrypted data. In: International Conference on Information Security and Cryptology, pp. 1–21. Springer (2012)
7. Commission, E.: General Data Protection Regulation. https://eur-lex.europa.eu/eli/reg/2016/679
8. Bourse, F., Minelli, M., Minihold, M., Paillier, P.: Fast homomorphic evaluation of deep discretized neural networks. In: Annual International Cryptology Conference, pp. 483–512. Springer (2018)
9. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)
10. Tillem, G., Bozdemir, B., Önen, M.: Swann: Switching among cryptographic tools for privacy-preserving neural network predictions. Preprint (2020)
11. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 1–40 (2009)
12. Carpov, S., Izabachène, M., Mollimard, V.: New techniques for multi-value input homomorphic evaluation and applications. In: Cryptographers' Track at the RSA Conference, pp. 106–126. Springer (2019)
13. Gentry, C., Boneh, D.: A fully homomorphic encryption scheme, vol. 20. Stanford University (2009)
14. Klemsa, J., Novotný, M.: WTFHE: neural-netWork-ready Torus Fully Homomorphic Encryption. In: 2020 9th Mediterranean Conference on Embedded Computing (MECO). IEEE (2020)
15. Dwork, C., Roth, A., et al.: The algorithmic foundations of differential privacy. Foundations and Trends® in Theoretical Computer Science **9**(3–4), 211–407 (2014)
16. Zhou, T., Yang, X., Liu, L., Zhang, W., Li, N.: Faster bootstrapping with multiple addends. IEEE Access **6**, 49,868–49,876 (2018)
17. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. nature **323**(6088), 533–536 (1986)
18. Rothblum, R.D.: On the circular security of bit-encryption. In: Theory of Cryptography Conference, pp. 579–598. Springer (2013)