Hitchhiker's Guide to the TFHE Scheme

Jakub Klemsa[1,1*]

[1]CTU in Prague, Prague, Czech Republic.
[2]EURECOM, Sophia-Antipolis, France.

Corresponding author(s). E-mail(s): jakub.klemsa@eurecom.fr;

Abstract

Also referred to as the holy grail of cryptography, *Fully Homomorphic Encryption* (FHE) allows for arbitrary calculations over encrypted data. First proposed as a challenge by Rivest *et al.* in 1978, the existence of an FHE scheme has only been shown by Gentry in 2009. However, until these days, existing general-purpose FHE schemes suffer from a substantial computational overhead, which has been vastly reduced since the original construction of Gentry, but it still poses an obstacle that prevents a massive practical deployment of FHE. The *TFHE scheme* by Chillotti *et al.* represents the state-of-the-art among general-purpose FHE schemes. This paper aims to serve as a thorough guide to the TFHE scheme, with a strong focus on the reliability of computations over encrypted data, and help researchers and developers understand the internal mechanisms of TFHE in detail. In particular, it may serve as a baseline for future improvements and/or modifications of TFHE or related schemes.

Keywords: Fully homomorphic encryption, TFHE scheme, Reliable homomorphic evaluation, Correctness

1 Introduction

Fully Homomorphic Encryption (FHE), first discovered by Gentry [1] in 2009, enables the evaluation of an arbitrary (computable) function over encrypted data. As a basic use-case of FHE, we outline a secure cloud-aided computation: we describe how a user (**U**) may delegate a computation over her sensitive data to a semi-trusted cloud (**C**):

- **U** generates secret keys sk, and (public) evaluation keys ek, which she sends to **C**;
- **U** encrypts her sensitive data d with sk, and sends the encrypted data to **C**;
- **C** employs ek to evaluate function f, homomorphically, over the encrypted data (i.e., without ever decrypting it), yielding an encryption of $f(d)$, which it sends back to **U**;
- **U** decrypts the message from **C** with sk, obtaining the desired result: $f(d)$ in plain.

We illustrate the concept of homomorphic evaluation of a function over the plain vs. over the encrypted data in Figure 1.

Among existing FHE schemes, two main means of homomorphic evaluation can be identified: (i) the *leveled* approach (e.g., [2, 3]), and (ii) the *bootstrapped* approach (e.g., [1, 4]), while a combination of both is also possible [5]. The leveled approach (i) is particularly useful for functions that are represented by an evaluation circuit of limited depth which is known in advance. After evaluating it, no additional operation can be performed with the result, otherwise, the data might be corrupted. Based on the circuit's depth and other properties, parameters must be chosen accordingly. On the other hand, for the bootstrapped approach (ii), there is no limit on the circuit depth, which also means that the circuit does not need to be known in advance. The TFHE

$$\begin{tikzcd}
 & \textbf{Plain data } d \arrow[r, "\text{FHE encryption}"] \arrow[d] & \textbf{Encrypted data } \mathsf{Encr}(d) \arrow[d] & \\
\text{User} & \text{evaluate } f \arrow[d] & \text{evaluate } f \textit{ homomorphically} \arrow[d] & \text{Cloud} \\
 & \textbf{Plain result } f(d) & \textbf{Encrypted result } \mathsf{Encr}\big(f(d)\big) \arrow[l, "\text{FHE decryption}"'] &
\end{tikzcd}$$

Fig. 1 Illustration of an evaluation of function f over the plain and over the encrypted data in the User's and in the Cloud's domain, respectively. In both ways, the same result is obtained.

scheme by Chillotti et al. [4] is currently considered as the state-of-the-art FHE scheme that follows the bootstrapped approach.

For a basic overview of the evolution of FHE schemes, we refer to a survey by Acar et al. [6] (from 2018; in particular for implementations, much progress has been made since then).

1.1 Basic Overview of TFHE

First, let us provide a high-level overview of TFHE and its abilities, and let us outline its structure.

Similar to many other FHE schemes, the TFHE scheme builds upon the *Learning With Errors* (LWE) encryption scheme, first introduced by Regev [7]. There are two important properties of LWE: *additive homomorphism*, and the *presence of noise*; let us comment on either:

Additive homomorphism: LWE ciphertexts, also referred to as *samples*, are represented by vectors of (additive) group elements. By the nature of LWE encryption, LWE samples are additively homomorphic, which means that—roughly speaking—for two samples $\mathbf{c}_1$ and $\mathbf{c}_2$, which encrypt respectively μ_1 and μ_2, it holds that $\mathbf{c}_1 + \mathbf{c}_2$ encrypts $\mu_1 + \mu_2$.
Noise: To achieve security, LWE samples need to contain a certain amount of noise. However, with each homomorphic addition, noises also add up, which may ultimately destroy the accuracy/correctness of the plaintext.

Like many other FHE schemes, TFHE deals with the noise growth by defining a procedure referred to as *bootstrapping*. Bootstrapping aims at resetting the noise to an—on average—fixed level. Otherwise, if the noise exceeded a certain bound, the probability of correct decryption would drop rapidly.

To sum up, TFHE offers two operations: (i) *homomorphic addition*, and (ii) *bootstrapping.* Homomorphic addition (i) is a very cheap operation, however, the noise accumulates. On the other hand, bootstrapping (ii) is a costly operation, but it refreshes the noise and—in case of TFHE—it is inherently capable of evaluating homomorphically a custom *Look-Up Table* (LUT), which can be moreover encrypted. These two operations are sufficient for the *full homomorphism*, i.e., the possibility to evaluate any computable function over encrypted data. In Figure 2, we introduce the TFHE *gate*, which comprises (i) homomorphic addition(s), grouped into a homomorphic *dot-product* with integer weights, followed by (ii) TFHE bootstrapping. For bootstrapping, we outline its internal structure that consists of four sub-operations: KeySwitch, ModSwitch, Blind-Rotate and SampleExtr.

1.2 Aim of this Work

The aim of this work is to provide FHE researchers and developers with a comprehensive and intelligible guide to the TFHE scheme. In particular, in this paper, we thoroughly analyze the noise growth of TFHE's operations, which is decisive for the correctness and reliability of homomorphic evaluations in the wild. Besides that, we comment on the purpose of selected tricks that are intended to decrease the noise growth. Therefore, our TFHE guide is supposed to provide useful insights for any prospective improvements and/or design modifications to TFHE(-related schemes).

Related Work. The original full paper on TFHE [4] is followed by other papers that recall or redefine TFHE in numerous ways [8–11], while changing the notation as well as the approach to describe the bootstrapping procedure.

Joye [12] provides a SoK paper on TFHE, supported by many examples with concrete values. Our TFHE guide complements their SoK in

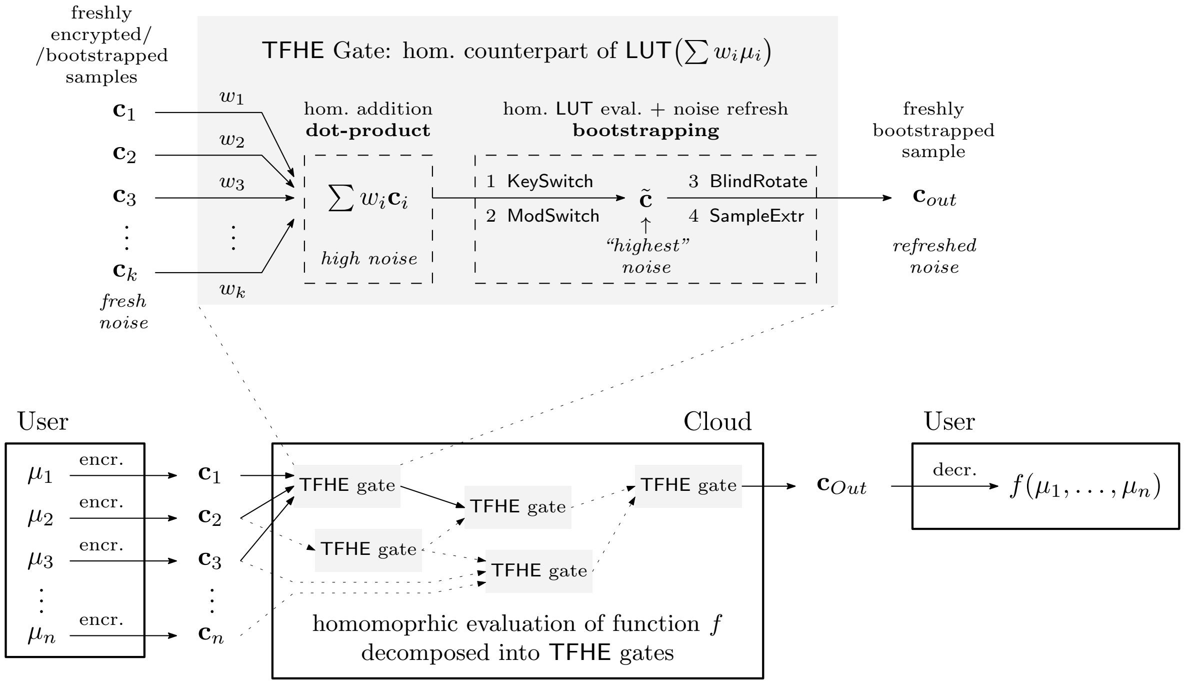

Fig. 2 TFHE gate: homomorphic addition(s) and bootstrapping, which comprises four sub-operations. The sample $(b', \mathbf{a}')$ may proceed to another TFHE gate, or it may go to the output and decryption.

particular by providing a thorough noise growth analysis, which is one of its pillars.

1.3 Paper Outline

We introduce building blocks of the TFHE scheme in Section 2. Next, in Section 3, we describe the construction of TFHE in detail with a particular focus on noise propagation. We further focus on the correctness of homomorphic evaluation in Section 4. In Section 5, we briefly comment on implementation aspects of TFHE. We conclude our paper in Section 6.

2 Building Blocks of TFHE

In this section, we first briefly outline flavors of LWE, we outline a technical notion, referred to as the *concentrated distribution*, and we provide a list of symbols and notation. Then, we introduce in detail a generalized variant of LWE, denoted GLWE, and we comment on its additive homomorphism, security and other properties. Finally, we define the *decomposition* operation that we use to build up a compound scheme called GGSW, which enables multiplicative homomorphism.

The Torus and the Ring Variant of LWE

Internally, TFHE employs two variants of LWE, originally referred to as TLWE and TRLWE, which stand for *(Ring)* LWE *over the Torus*. In a nutshell, let us outline what *torus* and *ring* mean in this context.

The *torus* is the underlying additive group of LWE that is used in TFHE, denoted $\mathbb{T}$ and defined as $\mathbb{T} := \mathbb{R}/\mathbb{Z}$ with the addition operation. The torus can be represented by the interval $[0, 1)$, with each addition followed by reduction mod 1, e.g., $0.3+0.8 = 0.1$. Since $\mathbb{T}$ is an abelian group, we may perceive $\mathbb{T}$ as an algebraic $\mathbb{Z}$*-module*, i.e., we further have scalar multiplication $\mathbb{Z} \times \mathbb{T} \to \mathbb{T}$, defined as repeated addition.

The *ring* variant of LWE, introduced by Lyubashevsky et al. [13], extends the module's ring to a ring of polynomials with a bounded degree. In TFHE, we will work with the ring $\mathbb{Z}[X]/(X^N + 1)$, denoted $\mathbb{Z}^{(N)}[X]$, with N a power of two. Then, the underlying $\mathbb{Z}^{(N)}[X]$-module comprises torus polynomials modulo $X^N + 1$, denoted $\mathbb{T}^{(N)}[X]$.

Concentrated Distribution

Unlike (scalar) multiplication, the division of a torus element by an integer cannot be defined without ambiguity, the same holds for the expectation of a distribution over the torus. However, this can be fixed for a *concentrated distribution* [4], which is a distribution with support limited to a ball of radius 1/4, up to a negligible subset. For further details, we refer to [4].

Symbols & Notation

Throughout the paper, we use the following symbols & notation; we denote:

- $\mathbb{B} := \{0,1\} \subset \mathbb{Z}$ the set of binary coefficients,
- $\mathbb{T}$ the additive group $\mathbb{R}/\mathbb{Z}$, referred to as the *torus* (i.e., real numbers modulo 1),
- $\mathbb{Z}_n$ the quotient ring $\mathbb{Z}/n\mathbb{Z}$ (or its additive group),
- $\lfloor \cdot \rceil \colon \mathbb{R} \to \mathbb{Z}$ the standard rounding function,
- for vector $\mathbf{v}$, v_i stands for its i-th coordinate,
- $\langle \mathbf{u}, \mathbf{v} \rangle$ the dot product of two vectors $\mathbf{u}$ and $\mathbf{v}$,
- $M^{(N)}[X]$ the additive group (or ring) of polynomials modulo $X^N + 1$ with coefficients from M, where $N \in \mathbb{N}$ is a power of two,
- for polynomial $p(X)$, $p^{(i)}$ stands for the coefficient of p at X^i,
- for vector of polynomials $\mathbf{w}$, $w_i^{(j)}$ stands for the coefficient at X^j of the i-th coordinate of $\mathbf{w}$,
- $\|p(X)\|_2^2$ the square of the l^2-norm of polynomial $p(X)$ (i.e., the sum of squared coefficients),
- $a \xleftarrow{\$} M$ the uniform draw of random variable a from M,
- $a \xleftarrow{\alpha} M$ the draw of random variable a from M with distribution α (for $\alpha \in \mathbb{R}$, we consider the zero-centered /discrete/ Gaussian draw with standard deviation α),
- $\mathrm{E}[X]$, $\mathsf{Var}[X]$ the expectation and the variance of random variable X, respectively.

2.1 Generalized LWE

First, we define a generalized variant of the LWE scheme, referred to as GLWE, which combines plain LWE with its ring variant. We define GLWE solely over the torus, although another underlying structure might be used, e.g., $\mathbb{Z}_q$ with prime q that is taken in some other schemes.

Definition 1 (GLWE Sample)**.** Let $k \in \mathbb{N}$ be the *dimension*, $N \in \mathbb{N}$, N a power of two, be the *degree*, $\alpha \in \mathbb{R}_0^+$ be the *standard deviation* of the noise, and let the plaintext space $\mathcal{P} = \mathbb{T}^{(N)}[X]$, the ciphertext (sample) space $\mathcal{C} = \mathbb{T}^{(N)}[X]^{1+k}$ and the key space $\mathcal{K} = \mathbb{Z}^{(N)}[X]^k$. For $\mu \in \mathcal{P}$ and $\mathbf{z} \xleftarrow{\chi} \mathcal{K}$, where χ is a *key distribution*, we call $\bar{\mathbf{c}} = (b, \mathbf{a}) =: \mathsf{GLWE}_{\mathbf{z}}(\mu)$ the GLWE *sample* of message μ under key $\mathbf{z}$, if

$$b = \mu - \langle \mathbf{z}, \mathbf{a} \rangle + e, \tag{1}$$

where $\mathbf{a} \xleftarrow{\$} \mathbb{T}^{(N)}[X]^k$ and $e \xleftarrow{\alpha} \mathbb{T}^{(N)}[X]$.

If $\mathbf{a} = \mathbf{0}$, we call the sample *trivial*, and if $\mu = \mathbf{0}$, we call the sample *homogeneous*. We denote $\bar{\mathbf{z}} := (1, \mathbf{z}) \in \mathbb{Z}^{(N)}[X]^{1+k}$, referred to as the *extended key*. For $N = 1$, we have the (plain) LWE *sample* and we usually denote its dimension by n. We also generalize GLWE sampling to vector messages, yielding a matrix of $1+k$ columns, with one GLWE sample per row.

GLWE sampling is actually *encryption*: in TFHE, plaintext data is encrypted using the plain LWE, while GLWE is used internally. To *decrypt*, we apply the GLWE *phase* function (followed by rounding if applicable).

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

$$\varphi_{\mathbf{z}}(b, \mathbf{a}) = b + \langle \mathbf{z}, \mathbf{a} \rangle = \langle \bar{\mathbf{z}}, \bar{\mathbf{c}} \rangle, \quad (= \mu + e), \tag{2}$$

the GLWE *phase*. We call the sample $\bar{\mathbf{c}}$ *valid* iff the distribution of $\varphi_{\mathbf{z}}(\bar{\mathbf{c}})$ is concentrated. Finally, for valid sample $\bar{\mathbf{c}}$, we call $\mathsf{msg}_{\mathbf{z}}(\bar{\mathbf{c}}) := \mathrm{E}\big[\varphi_{\mathbf{z}}(\bar{\mathbf{c}})\big]$ the *message* of $\bar{\mathbf{c}}$, which equals μ, since the noise is zero-centered and concentrated.

Remark 1. *GLWE phase returns $\mu + e$, i.e., the original message with a small amount of noise. We may define GLWE decryption as either:*

1. *an erroneous decryption via GLWE phase – we accept some errors in the decrypted result, which might be considered harmless or even useful, e.g., in the context of differential privacy [14]; or*
2. *a correctable decryption – for this purpose, we need to control the amount of noise and follow GLWE phase by an appropriate rounding step (relevant for this paper); or*
3. *an expectation of GLWE phase, i.e., $\mathsf{msg}_{\mathbf{z}}(\bar{\mathbf{c}})$ – this is useful for formal definitions and proofs.*

In the following theorem, we state the additively homomorphic property of GLWE.

Theorem 1 (Additive Homomorphism)**.** *Let* $\bar{\mathbf{c}}_1, \dots, \bar{\mathbf{c}}_n$ *be valid and independent* GLWE *samples under* GLWE *key* $\mathbf{z}$ *and let* $w_1, \dots, w_n \in \mathbb{Z}^{(N)}[X]$ *be integer polynomials (weights). In case* $\bar{\mathbf{c}} = \sum_{i=1}^{n} w_i \cdot \bar{\mathbf{c}}_i$ *is a valid* GLWE *sample, it holds*

$$\mathsf{msg}_{\mathbf{z}}\Big(\sum_{i=1}^{n} w_i \cdot \bar{\mathbf{c}}_i\Big) = \sum_{i=1}^{n} w_i \cdot \mathsf{msg}_{\mathbf{z}}(\bar{\mathbf{c}}_i) \qquad (3)$$

and for the noise variance

$$\mathsf{Var}[\bar{\mathbf{c}}] = \sum_{i=1}^{n} \|w_i\|_2^2 \cdot \mathsf{Var}[\bar{\mathbf{c}}_i]. \qquad (4)$$

If all samples $\bar{\mathbf{c}}_i$ *have the same variance* V_0*, we have* $\mathsf{Var}[\bar{\mathbf{c}}] = V_0 \cdot \sum_{i=1}^{n} \|w_i\|_2^2$ *and we define*

$$\nu^2 \coloneqq \sum_{i=1}^{n} \|w_i\|_2^2, \qquad (5)$$

referred to as the quadratic weights*. We refer to the operation* (3) *as the (homomorphic)* dot product (DP)*.*

2.1.1 Discrete-Valued Plaintext Space

As outlined in Remark 1, item 2, in this paper, we focus on a variant of GLWE that restricts its messages to a discrete subspace of the entire torus plaintext space. Denoted by $\mathcal{M}$, we refer to the plaintext subspace as the *cleartext space*, leaving the term *plaintext space* for torus polynomials.

In this paper, we only focus on the cleartext space of the form $\mathcal{M} = \frac{1}{2^\pi}\mathbb{Z}/\mathbb{Z} \subset \mathbb{T}$ (a subgroup of $\mathbb{T}$ isomorphic to $\mathbb{Z}_{2^\pi}$), where we refer to the parameter π as the *cleartext precision.* In terms of Definition 2, if it holds for the noise e that $|e| < {}^1\!/_{2^{\pi+1}}$, then rounding of the value $\varphi_{\mathbf{z}}(b, \mathbf{a}) \in \mathbb{T}$ to the closest element of $\mathcal{M}$ leads to the correct decryption/recovery of μ.

2.1.2 Discretized Torus

For the sake of simplicity of the noise growth analysis, TFHE is defined over the continuous torus, whereas in implementation, a discretized finite representation must be used instead. To cover the unit interval uniformly, TFHE implementations use an integral type—usually 32- or 64-bit `(u)int`—to represent a torus element, where we denote the bit-precision by τ. E.g., for $\tau = 32$-bit `uint32` type, $t \in$ `uint32` represents ${}^t\!/_{2^{32}} \in \mathbb{T} \sim [0, 1)$, where the denominator is usually denoted by $q = 2^\tau$ (in this case $q = 2^{32}$). Using such a representation, we effectively restrict the torus $\mathbb{T}$ to its submodule $\mathbb{T}_q \coloneqq q^{-1}\mathbb{Z}/\mathbb{Z} \subset \mathbb{T}$.

2.1.3 Distribution of GLWE Keys

For the coefficients of GLWE keys, a ternary distribution $\chi_p \colon (-1, 0, 1) \to (p, 1-2p, p)$, parameterized by $p \in (0, {}^1\!/_2)$, can be used. In particular, uniform ternary distribution is suggested by a draft of the homomorphic encryption standard [15], and it also is widely adopted by main FHE libraries like HElib [16], Lattigo [17], SEAL [18], or HEAAN [19], although they implement other schemes than TFHE. With a fixed GLWE dimension and carefully chosen p, the distribution χ_p may achieve better security as well as lower noise growth than uniform binary $\mathcal{U}_2 \colon (0, 1) \to ({}^1\!/_2, {}^1\!/_2)$. On the other hand, it is worth noting that for "small" values of p, such keys are also referred to as *sparse keys* (in particular with a fixed/limited Hamming weight), and there exist specially tailored attacks [20, 21]; we discuss security in the following paragraphs.

Note 1. *In* TFHE*, one instance of* LWE *and one of* GLWE *is employed. For* LWE *keys, usually, uniform binary distribution is used for technical reasons, although attempts to extend the key space can be found in the literature [22]. For* GLWE *keys, a ternary distribution can be used immediately.*

2.1.4 Security of (G)LWE

Estimation of the security of (G)LWE encryption is a complex task: it depends on (i) the size of the secret key (i.e., the dimension and/or the polynomial degree), (ii) the distribution of its coefficients, (iii) the distribution of the noise, which is usually given by its standard deviation, denoted by α, and (iv) the underlying structure (usually integers modulo q). As a rule of thumb, it holds that *the longer key, the better security*, as well as *the greater noise, the better security.*

A state-of-the-art tool that implements an LWE security assessment is known as `lattice-estimator` – a tool by Albrecht et al. [23, 24]. Authors aim at considering all known relevant attacks on LWE, including those targeting sparse keys, as outlined previously. A plot that shows selected results of `lattice-estimator` can be found in Figure 3. A code example of the usage of `lattice-estimator` as well as raw data that were used to generate the figure can be found in our repository[1].

2.1.5 Balancing Parameters

The downside of increasing the key size (improves *security*) is longer evaluation time (reduces *performance*), similarly increasing the amount of noise (improves *security*) leads to an error-prone evaluation (reduces *correctness*). Therefore, the goal is to find the best balance within the triangle of somehow orthogonal goals:

$$\begin{tikzcd}
\textit{security} \arrow[rr, leftrightarrow] \arrow[dr, leftrightarrow] & & \textit{correctness} \arrow[dl, leftrightarrow] \\
& \textit{performance} &
\end{tikzcd}$$

The problem of finding such a balance is thoroughly studied by Bergerat et al. [25], who provide concrete results that aim at achieving the best *performance*, without sacrificing *security*, nor *correctness*.

2.2 Decomposition

To enable homomorphic multiplication and at the same time to reduce its noise growth, torus elements get decomposed into a series of integers. The operation is parameterized by (i) the *decomposition base* (denoted B; we only consider $B = 2^\gamma$ a power of two), and (ii) by the *decomposition depth* (denoted d). We further denote

$$\mathbf{g} := \left({}^1\!/\!_B, {}^1\!/\!_{B^2}, \dots, {}^1\!/\!_{B^d}\right), \tag{6}$$

referred to as the *gadget vector*. We define *gadget decomposition of* $\mu \in \mathbb{T} \sim [-{}^1\!/\!_2, {}^1\!/\!_2) \subset \mathbb{R}$, denoted $\mathbf{g}^{-1}(\mu)$, as the base-B representation of $\tilde{\mu} = \lfloor B^d \cdot \mu \rceil \in \mathbb{Z}$ (multiplied in $\mathbb{R}$) in the alphabet $[-{}^B\!/\!_2, {}^B\!/\!_2) \cap \mathbb{Z}$. Note that such decomposition is unique. For the *decomposition error*, it holds that

$$\left|\mu - \langle \mathbf{g}, \mathbf{g}^{-1}(\mu) \rangle\right| \leq {}^1\!/\!_{2B^d}. \tag{7}$$

We denote

$$\varepsilon^2 := \frac{1}{12B^{2d}} \quad \text{and} \tag{8}$$

$$V_B := \frac{B^2 + 2}{12} \tag{9}$$

the variance of the decomposition error and the mean of squares of the alphabet $[-{}^B\!/\!_2, {}^B\!/\!_2) \cap \mathbb{Z}$ (n.b., we assume B is even), respectively; for both we consider a uniform distribution. Note that with the alphabet $[0, B) \cap \mathbb{Z}$, the respective value of V_B would have been higher, i.e., this is one of the little tricks to reduce later the noise growth.

For $k \in \mathbb{N}$, $k \geq 2$, we further denote

$$\mathbf{G}_k := \mathbf{I}_k \otimes \mathbf{g}, \tag{10}$$

where $\mathbf{I}_k$ is identity matrix of size k and $\otimes$ stands for the tensor product, i.e., we have $\mathbf{G}_k \in \mathbb{T}^{kd \times k}$, referred to as the *gadget matrix*.

We generalize $\mathbf{g}^{-1}$ to torus vectors and torus polynomials (and their combination) in a natural way: for vector $\mathbf{t} \in \mathbb{T}^n$, $\mathbf{g}^{-1}(\mathbf{t})$ is the concatenation of respective component-wise decompositions $\mathbf{g}^{-1}(t_i)$, for polynomial $t \in \mathbb{T}^{(N)}[X]$, $\mathbf{g}^{-1}(t)$ proceeds coefficient-wise, i.e., the output is a vector of integer polynomials. Finally, for a vector of torus polynomials, $\mathbf{g}^{-1}$ outputs a concatenation of respective vectors of integer polynomials.

2.3 GGSW & Homomorphic Multiplication

Unlike GLWE, which encrypts torus polynomials, GGSW encrypts integer polynomials. The main aim of GGSW is to allow homomorphic multiplication of a (GLWE-encrypted) torus polynomial by a (GGSW-encrypted) integer polynomial. The multiplicative homomorphic operation is referred to as the *External Product*, denoted by $\boxdot$: GGSW × GLWE → GLWE.

Definition 3 (GGSW Sample)**.** Let k, N and α be the parameters of a GLWE instance with key $\mathbf{z}$. We call $\bar{\mathbf{C}} = \bar{\mathbf{Z}} + m \cdot \mathbf{G}_{1+k}$, $\bar{\mathbf{C}} \in \mathbb{T}^{(N)}[X]^{(1+k)d, 1+k}$, the GGSW *sample* of $m \in \mathbb{Z}^{(N)}[X]$ if rows of $\bar{\mathbf{Z}}$ are mutually independent, homogeneous GLWE samples under the key $\mathbf{z}$. We call the sample *valid* iff there exists $m \in \mathbb{Z}^{(N)}[X]$ such that each row of $\bar{\mathbf{C}} - m \cdot \mathbf{G}_{1+k}$ is a valid homogeneous GLWE sample.

Definition 4 (External Product)**.** For GLWE sample $\bar{\mathbf{c}} = (b, \mathbf{a}) \in \mathbb{T}^{(N)}[X]^{1+k}$ and GGSW sample $\bar{\mathbf{A}}$ of corresponding dimensions, we define the *External Product*, $\boxdot$: GGSW × GLWE → GLWE, as

$$\mathbf{g}^{-1}(\bar{\mathbf{c}})^T \cdot \bar{\mathbf{A}} =: \bar{\mathbf{A}} \boxdot \bar{\mathbf{c}}. \tag{11}$$

[1] https://github.com/fakub/LWE-Estimates

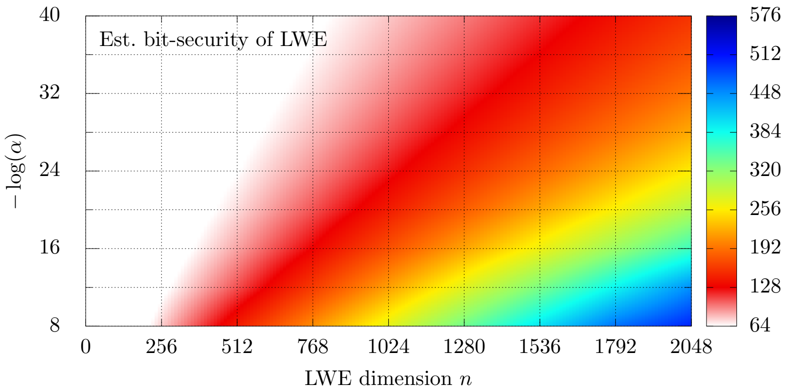

Fig. 3 Bit-security of LWE as estimated by `lattice-estimator` by Albrecht et al. [23, 24] (commit ID `f9dc7c`), using underlying group size $q = 2^{64}$. Interpolated between grid points. Raw data can be found at https://github.com/fakub/LWE-Estimates.

In the following theorem, we state the multiplicative homomorphic property of the external product and we evaluate its excess noise.

Theorem 2 (Correctness & Noise Growth of $\boxdot$). *Given* GLWE *sample* $\bar{\mathbf{c}}$ *of* $\mu_c \in \mathbb{T}^{(N)}[X]$ *under* GLWE *key* $\mathbf{z}$ *and noise parameter* α*, and* GGSW *sample* $\bar{\mathbf{A}}$ *of* $m_A \in \mathbb{Z}^{(N)}[X]$ *under the same key and noise parameters, external product returns* GLWE *sample* $\bar{\mathbf{c}}' = \bar{\mathbf{A}} \boxdot \bar{\mathbf{c}}$*, which holds excess noise* $e_\boxdot$*, given by* $\langle \bar{\mathbf{z}}, \bar{\mathbf{c}}' \rangle = m_A \cdot \langle \bar{\mathbf{z}}, \bar{\mathbf{c}} \rangle + e_\boxdot$*, for which it holds*

$$\mathsf{Var}[e_\boxdot] \approx \underbrace{dNV_B\alpha^2(1+k)}_{\textit{amplified } \mathsf{GGSW} \textit{ noise}} + \\ + \underbrace{\|m_A\|_2^2 \cdot \varepsilon^2(1 + kNV_\mathbf{z})}_{\textit{decomp. errors}}, \qquad (12)$$

where $V_\mathbf{z}$ *is the variance of individual coefficients of the* GLWE *key* $\mathbf{z}$ *and other parameters are as per previous definitions. If* $e_\boxdot$ *and the noise of* $\bar{\mathbf{c}}$ *are "sufficiently small",* $\bar{\mathbf{c}}'$ *encrypts* $\mathsf{msg}_\mathbf{z}(\bar{\mathbf{c}}') = m_A \cdot \mu_c$*, i.e., external product is indeed multiplicatively homomorphic.*

Proof. Find the proof in Appendix A.1. □

3 Constructing the TFHE Scheme

By far, there are two issues with (G)LWE:

1. As shown in Theorem 1, each additive homomorphic operation over (G)LWE samples leads to *noise growth* in the resulting aggregate sample, which limits the number of additions and which may also lead to incorrect results if the noise grows "too much".
2. Besides that, no homomorphic operation other than addition has been defined yet, which is not sufficient to achieve the *full homomorphism.*

The procedure referred to as *bootstrapping* aims at resolving them both at the same time: while *refreshing the noise* to a certain, constant-on-average level, bootstrapping also inherently *evaluates a function*, referred to as the *bootstrapping function*, represented by a *Look-Up Table* (LUT). This makes TFHE fully homomorphic.

Note 2. *An approach that clearly achieves the full homomorphism is presented in the original paper by Chillotti et al. [4], where authors define several logical gates, including* $\neg$*,* $\vee$*, and* $\wedge$*. We refer to this variant as the* binary TFHE*, however, in this paper we rather focus on the variant of* TFHE *with a discrete multi-value cleartext space* $\mathbb{Z}_{2^\pi}$*, as outlined in Section 2.1.1. Binary* TFHE *might then be perceived as a special case.*

As already outlined in Figure 2, TFHE gate is a combination of a homomorphic dot-product (cf. Theorem 1) and the bootstrapping procedure, which consists of four algorithms: KeySwitch, ModSwitch, BlindRotate and SampleExtr; find a more detailed illustration of bootstrapping in Figure 4. In the rest of this section, we discuss each algorithm in detail (except ModSwitch, which we cover together with BlindRotate) and we combine them into the Bootstrap algorithm.

According to the results of Bergerat et al. [25], it shows that in most cases, more efficient TFHE parameters can be found if dot-product is

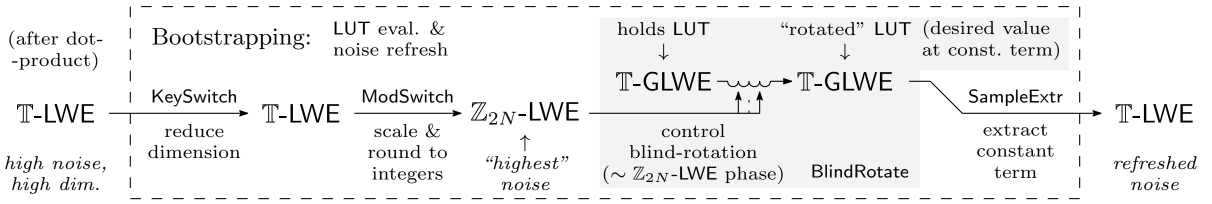

Fig. 4 Outline of the internal structure of TFHE's bootstrapping. The aim of KeySwitch is to improve performance, whereas BlindRotate (inside the gray box) evaluates the LUT and refreshes the noise.

moved *before* key-switching (originally proposed by Bourse et al. [26]), as opposed to the original variant of TFHE [4]. I.e., in the new variant, key-switching appears at the beginning of bootstrapping, whereas in the original variant, key-switching is the last step. Authors of [25] also consider a variant that omits key-switching, but they do not find it more efficient either. Hence, we describe solely the new, re-ordered variant with key-switching in this paper.

3.1 Key-Switching

The first step towards refreshing the noise, which happens in blind-rotate, is key-switching. Since blind-rotate is a demanding operation, the aim of key-switching is to reduce the dimension of the input sample. Attempts to omit key-switching were also tested by Bergerat et al. [25], however, achieving a poorer performance than the variant with key-switching.

The key-switching operation, denoted KeySwitch, effectively changes the encryption key of LWE sample $(b', \mathbf{a}')$ from LWE key $\mathbf{s}' \in \mathbb{B}^{n'}$ to LWE key $\mathbf{s} \in \mathbb{B}^{n}$. Besides the input LWE sample, KeySwitch requires a series of *key-switching keys*, while the j-th key is defined as

$$\mathsf{KS}_j := \mathsf{LWE}_{\mathbf{s}}(s'_j\,\mathbf{g}'), \qquad j \in [1, n'], \tag{13}$$

where $\mathbf{g}'$ is a gadget vector given by decomposition base B' and depth d', and where each component of $s'_j\,\mathbf{g}'$ produces one LWE sample, independent from others. I.e., $\mathsf{KS}_j \in \mathbb{T}^{d',1+n}$ is interpreted as a matrix, where rows are actual LWE samples. We denote the set of key-switching keys from $\mathbf{s}'$ to $\mathbf{s}$ as $\mathsf{KS}_{\mathbf{s}'\to\mathbf{s}} := (\mathsf{KS}_j)_{j=1}^{n'}$. Note that key-switching keys consist of LWE samples and they can therefore be published as a part of evaluation keys.

Given LWE sample $(b', \mathbf{a}') \in \mathbb{T}^{1+n'}$ of μ under $\mathbf{s}'$, key-switching keys $\mathsf{KS}_{\mathbf{s}'\to\mathbf{s}}$, generated with gadget vector $\mathbf{g}'$, we define *key-switching* as

$$\mathsf{KeySwitch}_{\mathbf{s}'\to\mathbf{s}}(b', \mathbf{a}') = (b', \mathbf{0}) - \sum_{j=1}^{n'} \mathbf{g}'^{-1}(a'_j)^T \cdot \mathsf{KS}_j, \tag{14}$$

which returns an LWE sample of μ under $\mathbf{s}$. Note that in fact, KeySwitch homomorphically evaluates the phase function. In the following theorem, we evaluate the excess noise induced by KeySwitch.

Theorem 3 (Correctness & Noise Growth of Key-Switching)**.** *Given* LWE *sample* $\bar{\mathbf{c}}'$ *of* $\mu \in \mathbb{T}$ *under* LWE *key* $\mathbf{s}'$ *and key-switching keys* $\mathsf{KS}_{\mathbf{s}'\to\mathbf{s}}$*, encrypted with noise parameter* α'*,* $\mathsf{KeySwitch}_{\mathbf{s}'\to\mathbf{s}}$ *returns* LWE *sample* $\bar{\mathbf{c}}$*, which holds excess noise* e_{KS}*, given by* $\langle \bar{\mathbf{s}}, \bar{\mathbf{c}} \rangle = \langle \bar{\mathbf{s}}', \bar{\mathbf{c}}' \rangle + e_{\mathsf{KS}}$*, for which it holds*

$$\mathsf{Var}[e_{\mathsf{KS}}] \approx \underbrace{n' V_{\mathbf{s}'} \varepsilon'^2}_{\textit{decomp. errors}} + \underbrace{n' d' V_{B'} \alpha'^2}_{\textit{amplif. } \mathsf{KS} \textit{ noise}}, \tag{15}$$

where ε'^2 *and* $V_{B'}$ *are as per* (8) *and* (9)*, respectively, with* B' *and* d'*,* $V_{\mathbf{s}'}$ *is the variance of individual coefficients of the* LWE *key* $\mathbf{s}'$*, and other parameters are as per previous definitions. If* e_{KS} *and the noise of* $\bar{\mathbf{c}}'$ *are "sufficiently small", it holds* $\mu = \mathsf{msg}_{\mathbf{s}}(\bar{\mathbf{c}}) = \mathsf{msg}_{\mathbf{s}'}(\bar{\mathbf{c}}')$*, i.e.,* KeySwitch *indeed changes the key, without modifying the message.*

Proof. Find the proof in Appendix A.2. □

3.2 Blind-Rotate

The blind-rotate operation, denoted BlindRotate, is the cornerstone of bootstrapping since this is where the noise gets refreshed. It combines two ingredients: the *decryption (phase) function* $\varphi_{\mathbf{s}}(b, \mathbf{a}) = \mu + e$ (cf. (2)), and the *multiplicative homomorphism* of GGSW × GLWE samples (cf. Theorem 2).

Internally, blind-rotate evaluates a relation reminiscent of the phase function $\varphi_{\mathbf{s}}(b, \mathbf{a})$. However, as such, the phase function does not get rid of the noise – indeed, the error term remains present in the original amount; cf. (2). Therefore, a rounding step—as outlined in Section 2.1.1—must be included, too. Blind-rotate achieves rounding using a staircase LUT, i.e., a LUT that encodes a staircase function, where the "stairs" are responsible for rounding. Such a LUT is provided in a form of a (possibly encrypted) polynomial, which is referred to as the *test vector*, denoted by $tv(X)$, whose coefficients represent the LUT values.

Roughly speaking, we aim at multiplying $tv(X)$ by X^{-M}, where M holds somehow the erroneous phase $\mu + e$. This "shifts" the coefficients of $tv(X)$ by M positions towards lower powers of X (we can think of discarding the coefficients that underflow for now). Then, evaluating $tv(X) \cdot X^{-M}$ at $X = 0$ (i.e., taking the constant term of the product) yields the originally M-th coefficient of $tv(X)$.

In the following paragraphs, we outline more concretely how a LUT can be encoded into a torus polynomial, which we further reduce modulo $X^N + 1$, so that it can be taken as a plaintext for a (possibly trivial) GLWE sample.

3.2.1 Encoding a LUT into a Polynomial Modulo $X^N + 1$

The product of degree-N polynomial $tv(X)$ and monomial X^m (with $0 \leq m < N$) holds the coefficients of tv shifted by m positions towards higher degrees. Reducing the product $tv(X) \cdot X^m$ modulo $X^N + 1$ brings the coefficients of powers higher than or equal to N back to lower powers (namely by N positions) while flipping their sign (e.g., aX^{N+k} is reduced to $-aX^k$). Hence, multiplication of a polynomial by a monomial modulo $X^N + 1$ yields a *negacyclic rotation*. These are the consequences for LUT evaluation in blind-rotate:

- multiplying $tv(X)$ by $X^{-m} \bmod X^N + 1$ with $0 \leq m < N$ results in moving the m-th coefficient of $tv(X)$ to the constant position, which is then taken as a result of the LUT;
- for $N \leq m < 2N$, the result equals to the $(m - N)$-th coefficient of $tv(X)$ with a flipped sign, due to the negacyclic rotation; and
- for greater m, it is worth noting that the period is $2N$.

Since we assume that tv is a torus polynomial, we have $\mathsf{LUT}\colon \mathbb{Z}_{2N} \to \mathbb{T}$ and it holds

$$\mathsf{LUT}(N + m) = -\mathsf{LUT}(m), \quad m \in [0, N), \tag{16}$$

i.e., only the first N values of a LUT need to be provided explicitly, while the other N values are given implicitly by the negacyclic extension, and the rest is periodic with a period of $2N$. Encoded in a test vector $tv \in \mathbb{T}^{(N)}[X]$ as

$$tv^{(m)} = \mathsf{LUT}(m), \quad m \in [0, N), \tag{17}$$

the LUT is evaluated at $m \in \mathbb{Z}$ as

$$\begin{aligned} &\left(X^{-m} \cdot tv(X) \bmod X^N + 1\right)^{(0)} = \\ &= (-1)^{\lfloor m/N \rfloor} \cdot tv(X)^{(m \bmod N)} = \\ &= \mathsf{LUT}(m \bmod 2N). \end{aligned} \tag{18}$$

We illustrate encoding of a LUT into a polynomial (test vector) $tv(X)$ in Figure 5. Next, we outline the overall idea of blind-rotate.

Encoding the Stairs

Let us put forward explicitly the process of encoding the desired (negacyclic) bootstrapping function $\bar{f}\colon \mathbb{Z}_{2^\pi} \to \mathbb{Z}_{2^\pi}$ (which acts on cleartexts) into the respective $\mathsf{LUT}\colon \mathbb{Z}_{2N} \to \mathbb{T}$, represented by the test vector $tv(X)$, including the "stairs":

$$\mathsf{LUT}(k) = \bar{f}\left(\left\lfloor k \cdot \frac{2^\pi}{2N} \right\rceil\right), \quad k \in [0, 2N). \tag{19}$$

We provide an illustration of such an encoding in Figure 6. We recall that only the LUT values for $k \in [0, N)$ are actually encoded into the test vector; cf. (17), (18) and Figure 5.

Recall that the "stairs" are supposed to be responsible for rounding, which in turn refreshes the noise. From 19, it follows that the width of such a stair is $1/2^\pi$. By $E_{\max}$, defined as

$$E_{\max} := \frac{1}{2^{\pi+1}}, \tag{20}$$

we denote the maximum of error magnitude that leads to the correct LUT evaluation; cf. Figure 6.
Note 3. *Bootstrapping cannot be applied to only refreshing the noise, i.e., setting identity as the*

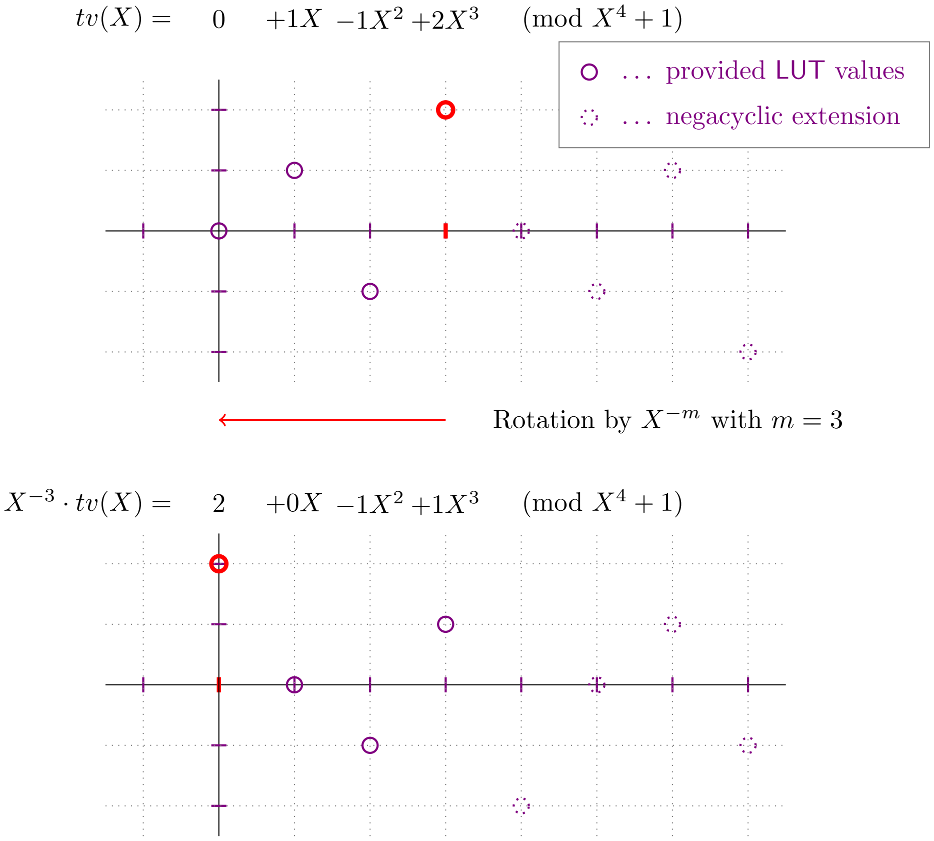

Fig. 5 Illustration of encoding of a LUT into a polynomial mod $X^N + 1$ that is used in blind-rotate. We set $N = 4$ and we evaluate at $m = 3$, which means "rotation" by X^{-3}. The desired output value $\mathsf{LUT}(3)$ is emphasized in red. N.b., in this illustration, we omit the "stairs" for simplicity.

bootstrapping function, since identity is not negacyclic. A workaround must be made, with respect to a particular use case. Specifically, many existing implementations prepend an extra bit of padding, which they set to zero and do not use it. Note that such implementations need to somehow prevent possible overflows of homomorphic additions.

3.2.2 Idea of Blind-Rotate

First, let us get back to the phase function, which is responsible for decryption. Originally, with a LWE sample $(b, \mathbf{a})$ to be bootstrapped, $\varphi_{\mathbf{s}}(b, \mathbf{a})$ is (i) evaluated over the *torus*, and (ii) it is using *known bits* of the key $\mathbf{s}$.

For (i): based on previous observations, we first rescale & round the sample $(b, \mathbf{a}) \in \mathbb{T}^{1+n}$ to the $\mathbb{Z}_{2N}$ domain, which preserves periodicity. Therefore, we calculate the scaled and rounded value of the phase function as

$$\tilde{m} = \tilde{b} + \langle \mathbf{s}, \tilde{\mathbf{a}} \rangle, \tag{21}$$

where $\tilde{b} = \lfloor 2Nb \rceil$ and $\tilde{a}_i = \lfloor 2Na_i \rceil$, which is also referred to as *modulus switching*. As outlined previously, we aim at performing the evaluation of $\tilde{m}$ as per (21) in powers of X; cf. (18).

For (ii): the secret key $\mathbf{s}$ is clearly not known to the evaluator (the cloud). Instead, bits s_i of the key are provided in a form of GGSW samples BK_i, encrypted with GLWE key $\mathbf{z}$, and referred to as the *bootstrapping keys*, denoted by $\mathsf{BK}_{\mathbf{s}\to\mathbf{z}} = (\mathsf{BK}_i)_{i=1}^n$.

From (i) and (ii), it follows that given the sample $(b, \mathbf{a})$ and the encrypted bits of the key $\mathbf{s}$ (bootstrapping keys), we can apply homomorphic operations to obtain the (encrypted) monomial $X^{\tilde{m}}$ modulo $X^N + 1$, which we employ for LUT evaluation as per (18). I.e., the test vector gets *blindly rotated*.

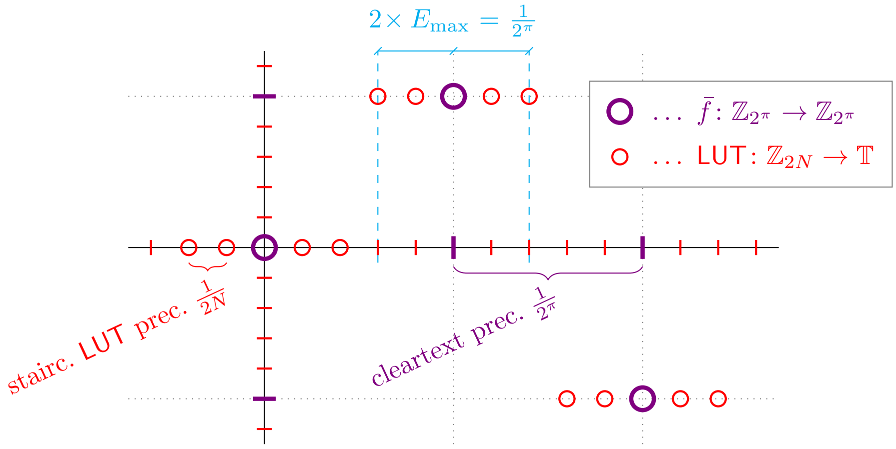
$2\times E_{\max} = \frac{1}{2^\pi}$
$\dots\ \bar{f}\colon \mathbb{Z}_{2^\pi} \to \mathbb{Z}_{2^\pi}$
$\dots$ LUT: $\mathbb{Z}_{2N} \to \mathbb{T}$
stairc. LUT prec. $\frac{1}{2N}$
cleartext prec. $\frac{1}{2^\pi}$

Fig. 6 Relation between a negacyclic bootstrapping function $\bar{f}$ and respective staircase LUT (illustrative). If the evaluated value does not leave its "stair", i.e., the input's error magnitude is lower than $E_{\max}$, LUT gets evaluated correctly.

In the following paragraphs, we provide a full technical overview of modulus-switching and blind-rotate, respectively.

3.2.3 Modulus-Switching

As outlined, blind-rotate is preceded by a technical step, referred to as modulus-switching and denoted by ModSwitch, which is parameterized by $N \in \mathbb{N}$. $\mathsf{ModSwitch}_N$, which inputs LWE sample $(b, \mathbf{a}) \in \mathbb{T}^{1+n}$ under LWE key $\mathbf{s}$ and outputs LWE sample $(\tilde{b}, \tilde{\mathbf{a}}) \in \mathbb{Z}_{2N}^{1+n}$ under the same key, is defined as

$$\mathsf{ModSwitch}_N(b, \mathbf{a}) = \left(\lfloor 2Nb \rceil, \lfloor 2Na_i \rceil_{i=1}^{n}\right) =: (\tilde{b}, \tilde{\mathbf{a}}), \tag{22}$$

where multiplications of type $2N \cdot a_i$ are performed in $\mathbb{R}$ (using any unit interval for $\mathbb{T}$), then rounding brings result back to $\mathbb{Z}$, from where we easily obtain $\mathbb{Z}_{2N}$.

Due to rounding, additional noise is induced by ModSwitch; we evaluate it in the following lemma.

Lemma 1 (Noise Growth of Modulus-Switching)**.** $\mathsf{ModSwitch}_N$ *induces an excess noise, given by* ${}^{1}\!/_{2N} \cdot \langle \bar{\mathbf{s}}, (\tilde{b}, \tilde{\mathbf{a}}) \rangle = \langle \bar{\mathbf{s}}, (b, \mathbf{a}) \rangle + e_{\mathsf{MS}}$*, for which it holds*

$$\mathsf{Var}[e_{\mathsf{MS}}] = \frac{1 + {}^{n}\!/_{2}}{48N^2}, \tag{23}$$

where n is the LWE *dimension.*

Proof. We write

$$e_{\mathsf{MS}} = \langle (1, \mathbf{s}), ({}^{\tilde{b}}\!/_{2N} - b, {}^{\tilde{\mathbf{a}}}\!/_{2N} - \mathbf{a}) \rangle =$$

$$= \underbrace{{}^{\tilde{b}}\!/_{2N} - b}_{\in(-{}^{1}\!/_{4N}, {}^{1}\!/_{4N}]} + \sum s_i \cdot \underbrace{({}^{\tilde{a}_i}\!/_{2N} - a_i)}_{\in(-{}^{1}\!/_{4N}, {}^{1}\!/_{4N}]}, \tag{24}$$

where each underbraced term is assumed to have a uniform distribution on $(-{}^{1}\!/_{4N}, {}^{1}\!/_{4N}]$, i.e., the variance of ${}^{1}\!/_{48N^2}$. For s_i, we have $\mathrm{E}[s_i^2] = {}^{1}\!/_{2}$. For independent variables with $\mathrm{E}[Y] = 0$, it holds $\mathsf{Var}[X \cdot Y] = \mathrm{E}[X^2] \cdot \mathsf{Var}[Y]$, which is this case for $X = s_i$ and $Y = {}^{\tilde{a}_i}\!/_{2N} - a_i$. The result follows. □

3.2.4 Description of Blind-Rotate

A description of blind-rotate is given in Algorithm 1. In line 3, if BK_i encrypts $s_i = 0$, the line evaluates to (encrypted) $\mathsf{ACC} = X^{0 \cdot \tilde{a}_i} \cdot \mathsf{ACC}$, if BK_i encrypts $s_i = 1$, we obtain (encrypted) $X^{1 \cdot \tilde{a}_i} \cdot \mathsf{ACC}$. I.e., after blind-rotation, we obtain an encryption of $X^{\tilde{m}} \cdot tv$, where $\tilde{m} = \langle \bar{\mathbf{s}}, (\tilde{b}, \tilde{\mathbf{a}}) \rangle$. Line 3 also mandates $s_i \in \mathbb{B}$, as outlined in Note 1, although generalization attempts exist [22].

Remark 2. *During blind-rotate, the "old" noise is refreshed with a fresh noise, which comes from the bootstrapping keys and from the (possibly encrypted) test vector – the fresh noise* does not *depend on the noise of the input sample. Nevertheless, the "old" noise affects what value from the test vector is selected (gets rotated to), i.e., at which point the (staircase)* LUT *is evaluated; cf. Figure 6. We discuss two types of decryption errors later in Section 4.1.1.*

In the following theorem, we evaluate the noise of the output of BlindRotate – i.e., the refreshed noise, which we denote V_0.

Algorithm 1 $\mathsf{BlindRotate}$

Input: LWE sample $(b, \mathbf{a})$ of $\mu \in \mathbb{T}$ under LWE key $\mathbf{s} \in \mathbb{B}^n$, modulus-switched to $(\tilde{b}, \tilde{\mathbf{a}}) \in \mathbb{Z}_{2N}^{1+n}$,

Input: (usually trivial) GLWE sample $\bar{\mathbf{t}} \in \mathbb{T}^{(N)}[X]^{1+k}$ of $tv \in \mathbb{T}^{(N)}[X]$ (aka. *test vector*) under GLWE key $\mathbf{z} \in \mathbb{Z}^{(N)}[X]^k$,

Input: for $i \in [1, n]$, GGSW samples of s_i under $\mathbf{z}$, referred to as *bootstrapping keys*, denoted $\mathsf{BK}_{\mathbf{s}\to\mathbf{z}} := (\mathsf{BK}_i)_{i=1}^n$.

Output: GLWE sample of $X^{\tilde{m}} \cdot tv$ under $\mathbf{z}$, where $\tilde{m} = \langle \tilde{\mathbf{s}}, (\tilde{b}, \tilde{\mathbf{a}})\rangle \approx 2N\mu$.

```
ACC ← $X^{\tilde{b}} \cdot \bar{\mathbf{t}}$                ▷ aka. accumulator
for $i \in [1, n]$ do
    ACC ← ACC + $\mathsf{BK}_i \boxdot (X^{\tilde{a}_i} \cdot$ ACC − ACC)
end for
return ACC
```

Theorem 4 (Correctness & Noise Growth of Blind-Rotate). *Given inputs of Algorithm 1, where bootstrapping keys are encrypted with noise parameter α and test vector is (possibly) encrypted with noise parameter α_t (i.e., $\alpha_t = 0$ or α),* $\mathsf{BlindRotate}$ *returns the last-step* ACC *with noise variance given by*

$$\begin{aligned}\mathsf{Var}[\langle \bar{\mathbf{z}}, \mathsf{ACC}\rangle] \approx \alpha_t^2 + ndNV_B\alpha^2(1+k) + \\ + n\varepsilon^2(1 + kNV_{\mathbf{z}}) =: V_0,\end{aligned} \tag{25}$$

which we denote by V_0, other parameters are as per previous theorems and definitions. If the noise of $\langle \bar{\mathbf{z}}, \mathsf{ACC}\rangle$ is "sufficiently small", it holds $\mathsf{msg}_{\mathbf{z}}(\mathsf{ACC}) = X^{\tilde{m}} \cdot tv$*, where $\tilde{m} = \langle \tilde{\mathbf{s}}, (\tilde{b}, \tilde{\mathbf{a}})\rangle \approx 2N\mu$, i.e.,* $\mathsf{BlindRotate}$ *indeed "rotates" the test vector by the approximate phase of $(b, \mathbf{a})$, scaled to $\mathbb{Z}_{2N}$.*

Proof. Find the proof in Appendix A.3. □

3.3 Sample-Extract

By far, $\mathsf{BlindRotate}$ outputs a GLWE sample of a polynomial (blindly-rotated test vector), which holds the desired value at its constant term and which is encrypted with a GLWE key. The goal of $\mathsf{SampleExtr}$ is to literally extract a partial LWE sample, which encrypts the constant term, out of the GLWE sample, which we denote by $(b, \mathbf{a}) \in \mathbb{T}^{(N)}[X]^{1+k}$ (the last-step ACC in Algorithm 1). Note that a similar thing happens with the key: the new LWE key is also an "extract" of the original polynomial GLWE key $\mathbf{z} \in \mathbb{Z}^{(N)}[X]^k$. Writing down the constant term of $\langle \bar{\mathbf{z}}, (b, \mathbf{a})\rangle$, which is a torus polynomial, we obtain

$$\begin{aligned}\big\langle \bar{\mathbf{z}}, (b, \mathbf{a})\big\rangle^{(0)} &= b^{(0)} + \sum_{i=1}^{k} \big(z_i(X) \cdot a_i(X)\big)^{(0)} = \\ &= b^{(0)} + \sum_{i=1}^{k} \Big\langle \underbrace{\big(z_i^{(0)}, -z_i^{(N-1)}, \dots, -z_i^{(1)}\big)}_{i\text{-th partial extr. } \mathsf{LWE} \text{ key } \mathbf{z}_i^*}, \\ &\qquad \underbrace{\big(a_i^{(0)}, a_i^{(1)}, \dots, a_i^{(N-1)}\big)}_{i\text{-th partial extr. } \mathsf{LWE} \text{ sample } \mathbf{a}_i^*} \Big\rangle,\end{aligned} \tag{26}$$

where we denote the i-th partial extracted LWE key and sample by $\mathbf{z}_i^* \in \mathbb{Z}^N$ and $\mathbf{a}_i^* \in \mathbb{T}^N$, respectively. We obtain the full extracted LWE key and sample as their concatenations, denoted respectively as $\mathbf{z}^* \in \mathbb{Z}^{kN}$ and $(b^{(0)}, \mathbf{a}^*) \in \mathbb{T}^{1+kN}$, where $b^{(0)}$ is prepended. Note that $\mathbf{a}^*$ is a simple serialization of polynomial coefficients of $\mathbf{a}$, whereas for $\mathbf{z}$, a rearranging is needed, together with negative signs. Finally, we have

$$\big\langle \bar{\mathbf{z}}, (b, \mathbf{a})\big\rangle^{(0)} = \big\langle \bar{\mathbf{z}}^*, (b^{(0)}, \mathbf{a}^*)\big\rangle, \tag{27}$$

while noise preserves. Note that the extracted key $\mathbf{z}^*$ plays the role of the LWE key $\mathbf{s}'$ that is supposed to be encrypted in key-switching keys – we compose the four algorithms and we provide further details in the next section.

3.4 TFHE Bootstrapping

Putting the four algorithms together, we obtain the TFHE *(Programmable) Bootstrapping* algorithm; find it as Algorithm 2, previously outlined in Figure 4. It is worth noting that $\mathsf{BlindRotate}$ (on line 3 of that algorithm) inputs a negative sample: this is due to the LUT encoding that we use; cf. (17) and (18), where a negative sign at m is expected, although by Theorem 4, a positive sign appears in the power of X.

We provide a summary of parameters in Table 1. For an exhaustive technical overview of blind-rotate, preceded by modulus-switching and followed by sample-extract, we refer to Appendix B, Figure B1.

Recall that the noise gets refreshed in Blind-Rotate (cf. Remark 2) and it does not change in

Algorithm 2 Bootstrap

```
Input: LWE sample $(b^*, \mathbf{a}^*) \in \mathbb{T}^{1+kN}$ of $\mu = {}^{m}/_{2^\pi}$, $m \in \mathbb{Z}_{2^\pi}$, under LWE key $\mathbf{z}^* \in \mathbb{Z}^{kN}$, extracted from GLWE key $\mathbf{z} \in \mathbb{Z}^{(N)}[X]^k$,
Input: (possibly trivial) GLWE sample $\bar{\mathbf{t}} \in \mathbb{T}^{(N)}[X]^{1+k}$ of test vector $tv \in \mathbb{T}^{(N)}[X]$ under the key $\mathbf{z}$, where $tv$ encodes negacyclic bootstrapping function $\bar{f} \colon \mathbb{Z}_{2^\pi} \to \mathbb{Z}_{2^\pi}$ as per (17) and (19),
Input: key-switching keys $\mathsf{KS}_{\mathbf{z}^* \to \mathbf{s}}$ and bootstrapping keys $\mathsf{BK}_{\mathbf{s} \to \mathbf{z}}$.
Output: LWE sample of ${}^{\bar{f}(m)}/_{2^\pi}$ under key $\mathbf{z}^*$.
$(b, \mathbf{a}) \leftarrow \mathsf{KeySwitch}\big((b^*, \mathbf{a}^*), \mathsf{KS}_{\mathbf{z}^* \to \mathbf{s}}\big)$
$(\tilde{b}, \tilde{\mathbf{a}}) \leftarrow \mathsf{ModSwitch}_N(b, \mathbf{a})$
$(s, \mathbf{r}) \leftarrow \mathsf{BlindRotate}\big((-\tilde{b}, -\tilde{\mathbf{a}}), \bar{\mathbf{t}}, \mathsf{BK}_{\mathbf{s} \to \mathbf{z}}\big)$
return $(b', \mathbf{a}') \leftarrow \mathsf{SampleExtr}\big((s, \mathbf{r})\big)$
```

Table 1 Summary of parameters' notation. Parameters ε^2, ε'^2 and V_B, $V_{B'}$ are derived from respective decomposition parameters; cf. (8) and (9).

LWE secret key	$\mathbf{s}$	GLWE secret key	$\mathbf{z}$
LWE dimension	n	GLWE dimension	k
		GLWE polyn. degree	N
LWE noise std-dev	α'	GLWE noise std-dev	α
KS decomp. base	B'	BK decomp. base	B
KS decomp. depth	d'	BK decomp. depth	d

SampleExtr, i.e., the variance of a freshly bootstrapped sample is given by V_0 as per (25). N.b., at this point, we do not guarantee the correctness of the output – details will be given in Section 4, where we identify bounds that need to be satisfied so that the bootstrapping function is evaluated at the correct point.

4 Correctness of LUT Evaluation

In this section, we combine the noise growth estimates from the previous section and we derive a condition for the correct evaluation of the bootstrapping function. As outlined, the noise propagates throughout various operations, let us provide an overview first.

Overview of Noise Propagation

The noise, which is present in every encrypted sample, evolves during the evaluation of a TFHE gate, which comprises (i) homomorphic *dot-product* and (ii) *bootstrapping* (with its four sub-operations). Let us comment on either operation:

Dot-product: Provided that the noise of involved samples is independent, the error variance of a weighted sum is additive with weights squared (cf. (4) in Theorem 1).
Bootstrapping: If the noise of the sample-to-be-bootstrapped is smaller than a certain bound, the blind-rotate step of bootstrapping evaluates the bootstrapping function correctly: i.e., the error of $\tilde{m}$ (as per Theorem 4 and Algorithm 1) is smaller than the bound $E_{\max}$; cf. Figure 6. The resulting sample then carries—on average—a fixed amount of noise (independent of the original sample), which solely depends on the TFHE parameters (cf. (25) in Theorem 4).

In Figure 7, we illustrate the error propagation throughout a TFHE gate, where $e_0^{(i)}$ denotes actual noise of the i-th, freshly bootstrapped sample. Note that the overall maximum of relative average noise is achieved within bootstrapping when the modulus-switched sample $(\tilde{b}, \tilde{\mathbf{a}})$ enters blind-rotate, which refreshes the noise; cf. Remark 2.

We denote the maximum error and its variance by $e_{\max}$ and $V_{\max}$, respectively, and we have

$$V_{\max} \approx \nu_{\max}^2 \cdot V_0 + V_{\mathsf{KS}} + V_{\mathsf{MS}}, \tag{28}$$

where $\nu_{\max}^2$ is the maximum of sums of squares of integer weights of dot-products (cf. (5)) across the entire computation, V_0, V_{KS} and V_{MS} are respectively the variance of a freshly bootstrapped sample (cf. (25), combined using (4)), the variance of the excess noise of key-switching (cf. (15)) and that of modulus-switching (cf. (23)). Note that $e_{\max}$ is the relative, torus-scaled error of $\tilde{m} \in \mathbb{Z}_{2N}$ that enters blind-rotate; cf. Algorithm 2. The magnitude of this error is decisive for the correctness of the bootstrapping function evaluation as per Figure 6.

Note 4. *Let us outline an intuition that justifies the design where key-switching is moved to the beginning of bootstrapping (proposed in [26], experimentally shown to be more efficient in [25], presented in this paper), as opposed to the original* TFHE *design [4], where key-switching is the last step of bootstrapping. The variance of the*

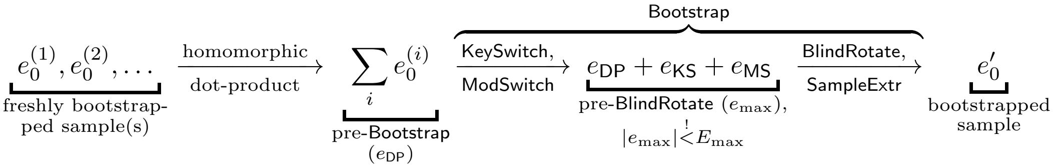
Bootstrap
$e_0^{(1)}, e_0^{(2)}, \ldots$
freshly bootstrapped sample(s)
homomorphic
dot-product
$\sum_i e_0^{(i)}$
pre-Bootstrap (e_{DP})
KeySwitch,
ModSwitch
$e_{\mathsf{DP}} + e_{\mathsf{KS}} + e_{\mathsf{MS}}$
pre-BlindRotate ($e_{\max}$),
$|e_{\max}| \overset{!}{<} E_{\max}$
BlindRotate,
SampleExtr
e_0'
bootstrapped sample

Fig. 7 Noise propagation from a bunch of freshly bootstrapped samples throughout a TFHE gate towards a new, freshly bootstrapped sample. If $|e_{\max}| < E_{\max}$, the bootstrapping function is evaluated correctly.

maximum error of the original variant writes

$$V_{\max} \approx (\underbrace{V_{\mathsf{BR}} + V_{\mathsf{KS}}}_{V_0^{(or.)}}) \cdot \nu^2 + V_{\mathsf{MS}}, \tag{29}$$

whereas in the re-ordered variant, we have

$$V_{\max} \approx \underbrace{V_{\mathsf{BR}}}_{V_0^{(re.)}} \cdot \nu^2 + V_{\mathsf{KS}} + V_{\mathsf{MS}}, \tag{30}$$

where V_{BR} is the variance of the output of BlindRotate. *We may notice that the re-ordered variant is expected to achieve a lower noise growth, in particular for applications with greater ν^2. Also, note that the re-ordered variant in fact only swaps key-switching and dot-product; let us outline both, starting after sample-extract:*

$$\textit{orig.:}\ (\mathsf{SE}) \to \mathsf{KS} \xrightarrow{V_0^{(or.)}} \mathsf{DP} \xrightarrow{\textit{to bs.}} \mathsf{MS} \to \ldots$$

$$\textit{reord.:}\ (\mathsf{SE}) \xrightarrow{V_0^{(re.)}} \mathsf{DP} \xrightarrow{\textit{to bs.}} \mathsf{KS} \to \mathsf{MS} \to \ldots$$

4.1 Correct Evaluation of the Bootstrapping Function

Let us define the quantity κ, which aims at quantifying the probability of correct evaluation of the bootstrapping function (i.e., $|e_{\max}| < E_{\max}$), as

$$\kappa := \frac{E_{\max}}{\sqrt{V_{\max}}}. \tag{31}$$

The aim of κ is to tell how many times the standard deviation of the maximum error, denoted $\sigma_{\max} = \sqrt{V_{\max}}$, fits into the target interval of the size of $2E_{\max}$ around the expected value. The probability that a normally distributed random variable falls within the interval of κ times its standard deviation can be looked-up from *standard normal tables* (aka. the *Z-tables*). Note that by the Central Limit Theorem, we assume a normal distribution for the value of $\tilde{m}$. E.g., for $\kappa = 3$, we have $\Pr[\cdot] \approx 99.73\% \approx 1/370$ (aka. rule of 3σ), however, we recommend higher values of κ (e.g., Bergerat et al. [25] provide their parameters with $\kappa = 4$, which gives error rate $\approx 1/15\,787$). N.b., also the size of the evaluated circuit as well as possible real-world consequences of an incorrect evaluation shall be taken into account.

From (31) and (20), we obtain the *fundamental condition* on the variance of the maximum error as

$$\boxed{V_{\max} \leq \frac{1}{\kappa^2 \cdot 2^{2\pi+2}},} \tag{32}$$

where $V_{\max}$ can be further broken down by (28) and other previous equalities. If the fundamental condition is satisfied, the (erroneous) value of $\tilde{m}$ does not leave its "stair" with high probability (related to κ), and the bootstrapping function $\bar{f}$ is evaluated correctly; cf. Figure 6.

4.1.1 Types of Decryption Errors

Correct blind-rotate does not itself guarantee the correctness of the result after decryption – indeed, there is a non-zero probability that the freshly bootstrapped sample (or a dot-product of them) decrypts incorrectly due to the intrinsic LWE noise. Therefore, we define two types of decryption errors that may occur after a dot-product followed by bootstrapping: one due to LWE noise (as just outlined), and one due to incorrect blind-rotate.

Fresh Bootstrap Error (Err_1)

First, let us assume that blind-rotate rotates the test vector correctly (i.e., $|e_{\max}| < E_{\max}$) and we denote the output LWE sample of bootstrapping as $\bar{\mathbf{c}}'$. Then, if the distance of $\langle \bar{\mathbf{s}}, \bar{\mathbf{c}}' \rangle$ from the expected value is greater than $E_{\max}$, we refer to this kind of error as the *type-1 error*, denoted Err_1.

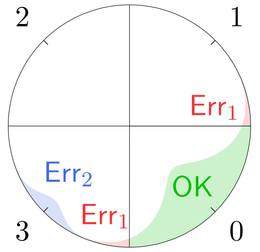

Fig. 8 Illustration of type-1 and type-2 errors: LUT evaluates correctly and incorrectly to 0 and 3, respectively.

The probability of Err_1 relates to the noise of a correctly blind-rotated, freshly bootstrapped sample, which can be estimated from V_0; see (25).

Blind-Rotate Error (Err_2)

Second, we consider the result of a TFHE gate, i.e., we take a dot-product of a bunch of independent, freshly bootstrapped samples, with $\nu^2 \leq \nu^2_{\max}$, and we bootstrap it. Then, if blind-rotate rotates the test vector incorrectly (i.e., $|e_{\max}| > E_{\max}$), we refer to this kind of error as the *type-2 error*, denoted Err_2. Note that a combination of both error types may occur[2].

The probability of Err_2 relates to the error of modulus-switched sample $(\tilde{b}, \tilde{\mathbf{a}})$ that appears inside bootstrapping, and it can be estimated from $V_{\max}$; see (28). We outline both error types in Figure 8.

Corollary 1. *For the probabilities of type-1 and type-2 errors, by* (28) *we have*

$$\Pr[\mathsf{Err}_1] < \Pr[\mathsf{Err}_2]. \tag{33}$$

For common choices of parameters, $\Pr[\mathsf{Err}_1]$ can be neglected. I.e., we may use the fundamental condition (32) to estimate the probability of incorrect evaluation of a single TFHE gate.

4.2 Parameter Constraints

Previously, we justified the use of the fundamental condition (32) to make error probability estimates. Next, we identify four high-level parameters that aim at characterizing the properties of an instance of TFHE. Finally, we combine the fundamental condition to obtain a relation between the four high-level parameters and actual TFHE parameters (like LWE dimension or noise amplitude).

4.2.1 Characteristic Parameters

Given a usage scenario, an instance of TFHE can be characterized by the following four (input) parameters:

1. *cleartext space bit-precision*, denoted by π (cf. Section 2.1.1);
2. *quadratic weights*, denoted by ν^2 (cf. Theorem 1, we take maximum of computation);
3. *bit-security level*, denoted by λ; and
4. *bootstrapping correctness*, denoted by $\eta := \Pr[\mathsf{Err}_2]$ (cf. Corollary 1).

Let us provide more (practical) comments on each of the input parameters.

Cleartext Bit-Precision: π

Regarding the choice of an appropriate cleartext bit-precision, we point out two things: First, it shows that the complexity of the TFHE bootstrapping grows roughly exponentially with the cleartext bit-precision – reasonable bootstrapping times can be achieved for up to about $\pi = 8$ bits, then, splitting the cleartext into multiple chunks comes into play. Second, bootstrapping is capable of evaluating a custom bootstrapping function $\bar{f} \colon \mathbb{Z}_{2^\pi} \to \mathbb{Z}_{2^\pi}$, however, such function must be negacyclic; cf. (16) and (19), unless a workaround is adopted as per Note 3. Both limitations must be carefully considered before choosing the right cleartext space bit-precision π: it might make sense to decrease the cleartext space size at the expense of additional, but cheaper bootstrapping.

Quadratic Weights: ν^2

As outlined in (28), $\nu^2_{\max} := \max_g\{\nu_g^2\}$ is defined as the maximum of sums of squares of integer weights of dot-products across the whole circuit that comprises TFHE gates $g \in G$ (with ν^2 defined in (5)). Note that $\log(\nu_g)$ expresses the number of bits of the standard deviation of the excess noise introduced by the dot-product in gate g.

Security Level: λ

We discuss LWE/GLWE security in Section 2.1.4. Recall that the higher λ is requested, the higher LWE dimension and/or the lower noise must be

[2] The result may combine both kinds of errors and decrypt correctly at the same time – in such a case, we consider that both error types occur simultaneously.

present, and security also depends on the distribution of keys.

Bootstrapping Correctness: η

Introduced in Section 4.1, the parameter κ characterizes the probability of erroneous blind-rotate. In Corollary 1, we use this probability to estimate the overall probability of correct evaluation of a TFHE gate. Hence, to quantify the probability of correct evaluation of a single TFHE gate, we take η, and by standard normal tables, we deduce the value of κ, which we use for the rest of the analysis. Recall that κ relates to the correctness of a single TFHE gate, i.e., for a circuit that consists of multiple TFHE gates, the value of η needs to be modified accordingly.

4.2.2 Parameter Relations

To make the fundamental condition (32) hold, we may combine (28) with (25), (15) and (23), and mandate

$$\begin{aligned} V_{\max} &\approx \nu^2 \cdot V_0 + V_{\mathsf{KS}} + V_{\mathsf{MS}} \approx \\ &\approx \nu^2 \cdot \big(\alpha_t^2 + ndNV_B\alpha^2(1+k) + \\ &\quad + n\varepsilon^2(1+kNV_{\mathbf{z}})\big) + kNV_{\mathbf{z}}\varepsilon'^2 + \\ &\quad + kNd'V_{B'}\alpha'^2 + \frac{1+{}^{n}\!/_{2}}{48N^2} \overset{!}{\leq} \\ &\overset{!}{\leq} \frac{1}{\kappa^2 \cdot 2^{2\pi+2}}, \end{aligned} \tag{34}$$

where the baseline parameters are summarized in Table 1, ε^2 and V_B are defined in (8) and (9), respectively, and $V_{\mathbf{z}}$ stands for the variance of coefficients of the internal GLWE secret key $\mathbf{z}$.

In terms of the *security* ↔ *correctness* ↔ *performance* triangle given in Section 2.1.5, this inequality only provides a guarantee of *correctness*, which is given by η (translated into κ), for prescribed plaintext precision π and quadratic weights ν^2. In particular, *security* is *not* addressed and it must be resolved separately, e.g., using `lattice-estimator` [23]. The combination of constraints on TFHE parameters makes it a complex task to generate a set of parameters, which is further supposed to achieve a good *performance*. Authors of [25] claim to have implemented a generator of efficient TFHE parameters and they provide a comprehensive list of TFHE parameters for many input setups. However, at the time of writing, the tool is not publicly available yet.

5 Implementation Remarks

In this section, we briefly comment on various implementation aspects of TFHE, namely

- (negacyclic) polynomial multiplication;
- additional errors (noise) that stem from particular implementation choices;
- estimated complexity & key sizes; and
- existing implementations of TFHE, including recent trends and advances.

5.1 Negacyclic Polynomial Multiplication

For performance reasons, modular polynomial multiplication in TFHE—which appears, e.g., in GLWE encryption (1) or in external product (11)—is implemented using *Fast Fourier Transform* (FFT). Recall that polynomials mod $X^N + 1$ rotate negacyclically when multiplied by X^k, unlike polynomials mod $X^N - 1$, which rotate cyclically. Note that in such a case, polynomial multiplication is equivalent to the standard cyclic convolution, which can be calculated using Fast Fourier Transform (FFT). However, for polynomials mod $X^N + 1$, other tricks need to be put into place; find a description of negacyclic polynomial multiplication, e.g., in [27].

5.2 Implementation Noise

Notably, FFT is the major source of additional errors (noise) that are *not* captured by the theoretical noise analysis given in Section 4. The magnitude of FFT errors depends particularly on the number representation that is used by selected FFT implementation; find a study on FFT errors in [27]. Although for commonly used parameters and FFT implementations, FFT errors are negligible compared to (G)LWE noises, they shall be kept in mind, in particular in non-standard constructions or new designs.

In addition, compared to the theoretical results of Section 4, torus elements are represented using a finite representation (as outlined in Section 2.1.2; e.g., with 64-bit integers), which also changes the errors slightly. However, as long as the torus precision (e.g., 2^{-64}) is much smaller than the standard deviation of the (G)LWE noise—which is usually the case for common parameter choices—this contribution can be neglected.

5.3 Key Sizes & Bootstrapping Complexity

Below, we provide the sizes of key-switching and bootstrapping keys, as represented in a TFHE implementation with a τ-bit representation of torus elements. The complexity of TFHE bootstrapping, which is the dominant operation of TFHE, is roughly proportional to the key sizes.

Size of Key-Switching Keys

Using notation of Table 1, key-switching keys consist of N sub-keys $\mathsf{KS}_j \in \mathbb{T}^{d',1+n}$; altogether we have

$$\left|(\mathsf{KS}_j)_{j=1}^{N}\right| = Nd'(1+n)\tau \text{ [bits]}. \tag{35}$$

Note that a common method to store/transmit key-switching keys, which can also be applied to bootstrapping keys, is to keep just a seed for a pseudo-random number generator (PRNG), instead of all of the randomness. I.e., for each LWE sample, just the value of b is kept and the values of $\mathbf{a}$ can be re-generated from the seed.

Size of Bootstrapping Keys

Bootstrapping keys consist of n GGSW samples $\mathsf{BK}_i \in \mathbb{T}^{(N)}[X]^{(1+k)d,1+k}$; altogether we have

$$\left|(\mathsf{BK}_i)_{i=1}^{n}\right| = n(1+k)^2 dN\tau \text{ [bits]}. \tag{36}$$

Rough Estimate of Bootstrapping Complexity

Key-switching is dominated by $Nd'(1+n)$ torus multiplications, followed by $1+n$ summations of Nd' elements, which makes key-switching $O\big(Nd'(1+n)\tau\big)$. Blind-rotate is dominated by $n(1+k)^2 d$ degree-N polynomial multiplications, followed by a similar number of additions/subtractions, which makes blind-rotate $O\big(n(1+k)^2 dN\tau\big)$. Note that the entire calculation of BlindRotate (cf. Algorithm 1) can be performed in the Fourier domain – thanks to its linearity and pre-computed bootstrapping keys, i.e., the $O(\tau N \log N)$ term of FFT can be neglected. In this rough estimate, we neglect modulus-switching and sample-extract. Also, we do not distinguish the bit-length of τ for LWE and GLWE, as some implementations do [28].

5.4 Existing TFHE Implementations

The original (experimental) TFHE library [29] is not developed anymore, instead, there are other, more or less active implementations. Here we list selected implementations of TFHE:

`TFHE-rs` *[30]:* written in Rust, `TFHE-rs` implements latest findings by Bergerat et al. [25] (recently separated from the Concrete Library [31] that implements higher-level operations and interfaces);
`FPT` *[32]:* an experimental FPGA accelerator for TFHE bootstrapping (a benchmark of state-of-the-art implementations in software/GPU/FPGA/ASIC can also be found in [32]);
`nuFHE` *[33]:* a GPU implementation of TFHE.

TFHE is also implemented as a part of more generic libraries like OpenFHE [34], which is a successor of PALISADE [35] and which also attempts to incorporate the capabilities of HElib [16] and HEAAN [19]. There exist many other implementations that are not listed here.

6 Conclusion

We believe that our TFHE guide helps many researchers and developers understand the inner structure of TFHE, in particular probably the most mysterious operation – the negacyclic blind-rotate – thanks to a step-by-step explanation, which we support with an illustration. Not only do we provide an intelligible description of each sub-operation of bootstrapping, but we also highlight what tweaks can be put in place to limit the noise growth at little to no additional cost (e.g., the order of key-switch ↔ dot-product or the signed decomposition alphabet). Last but not least, we provide a comprehensive noise analysis, supported by proofs, where we employ an easy-to-follow notation of the decomposition operation using $\mathbf{g}$ and $\mathbf{g}^{-1}$. Finally, we list selected implementation remarks that shall be kept in mind when attempting to implement TFHE and its variants or modifications.

Declarations

Acknowledgments

We would like to thank Melek Önen for her useful comments as well as the anonymous referees for their helpful suggestions.

Availability of Data and Material

The datasets generated during and/or analyzed during the current study are available in the following repository: https://github.com/fakub/LWE-Estimates.

Funding

This work was supported by the MESRI-BMBF French-German joint project UPCARE (ANR-20-CYAL-0003-01), and by the Grant Agency of CTU in Prague, grant No. SGS21/160/OHK3/3T/13.

Conflicts of interest

The authors have no relevant financial or non-financial interests to disclose.

Ethical approval

This article does not contain any studies with human participants or animals performed by any of the authors.

References

[1] Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing, pp. 169–178 (2009)

[2] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 13 (2014)

[3] Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: International Conference on the Theory and Application of Cryptology and Information Security, pp. 409–437 (2017). Springer

[4] Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)

[5] Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 360–384 (2018). Springer

[6] Acar, A., Aksu, H., Uluagac, A.S., Conti, M.: A survey on homomorphic encryption schemes: Theory and implementation. ACM Computing Surveys (Csur) **51**(4), 1–35 (2018)

[7] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing, pp. 84–93 (2005)

[8] Guimarães, A., Borin, E., Aranha, D.F.: Revisiting the functional bootstrap in TFHE. IACR Transactions on Cryptographic Hardware and Embedded Systems, 229–253 (2021)

[9] Chillotti, I., Ligier, D., Orfila, J.-B., Tap, S.: Improved Programmable Bootstrapping with Larger Precision and Efficient Arithmetic Circuits for TFHE. In: International Conference on the Theory and Application of Cryptology and Information Security, pp. 670–699 (2021). Springer

[10] Chen, H., Chillotti, I., Song, Y.: Multi-key homomorphic encryption from tfhe. In: International Conference on the Theory and Application of Cryptology and Information Security, pp. 446–472 (2019). Springer

[11] Kwak, H., Min, S., Song, Y.: Towards Practical Multi-key TFHE: Parallelizable, Key-Compatible, Quasi-linear Complexity. Cryptology ePrint Archive, Paper 2022/1460 (2022). https://ia.cr/2022/1460

[12] Joye, M.: Sok: Fully homomorphic encryption

over the [discretized] torus. IACR Transactions on Cryptographic Hardware and Embedded Systems, 661–692 (2022)

[13] Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 1–23 (2010). Springer

[14] 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)

[15] Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., et al.: Homomorphic encryption standard. Protecting privacy through homomorphic encryption, 31–62 (2021)

[16] homenc: HElib. https://github.com/homenc/HElib (2023)

[17] Mouchet, C.V., Bossuat, J.-P., Troncoso-Pastoriza, J.R., Hubaux, J.-P.: Lattigo: A multiparty homomorphic encryption library in go. In: Proceedings of the 8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography, pp. 64–70 (2020)

[18] Microsoft: SEAL (release 4.1). https://github.com/Microsoft/SEAL (2023)

[19] SNUCrypto: HEAAN (release 1.1). https://github.com/snucrypto/HEAAN (2018)

[20] Cheon, J.H., Hhan, M., Hong, S., Son, Y.: A hybrid of dual and meet-in-the-middle attack on sparse and ternary secret lwe. IEEE Access **7**, 89497–89506 (2019)

[21] Son, Y., Cheon, J.H.: Revisiting the hybrid attack on sparse secret lwe and application to he parameters. In: Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography, pp. 11–20 (2019)

[22] Joye, M., Paillier, P.: Blind rotation in fully homomorphic encryption with extended keys. In: Cyber Security, Cryptology, and Machine Learning: 6th International Symposium, CSCML 2022, Be'er Sheva, Israel, June 30–July 1, 2022, Proceedings, pp. 1–18 (2022). Springer

[23] Albrecht, M.R., contributors: Security Estimates for Lattice Problems. https://github.com/malb/lattice-estimator (2022)

[24] Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)

[25] Bergerat, L., Boudi, A., Bourgerie, Q., Chillotti, I., Ligier, D., Orfila, J.-B., Tap, S.: Parameter optimization & larger precision for (t)fhe (2022)

[26] Bourse, F., Minelli, M., Minihold, M., Paillier, P.: Fast homomorphic evaluation of deep discretized neural networks. In: Annual International Cryptology Conference, pp. 483–512 (2018). Springer

[27] Klemsa, J.: Fast and error-free negacyclic integer convolution using extended fourier transform. In: Cyber Security Cryptography and Machine Learning: 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8–9, 2021, Proceedings, pp. 282–300 (2021). Springer

[28] NuCypher: TFHE.jl. https://github.com/nucypher/TFHE.jl (2022)

[29] TFHE: Fast Fully Homomorphic Encryption Library over the Torus. https://github.com/tfhe/tfhe (2016)

[30] TFHE-rs: Pure Rust implementation of the TFHE scheme for boolean and integers FHE arithmetics (v0.1.12). https://docs.zama.ai/tfhe-rs (2023)

[31] Concrete: State-of-the-art TFHE library for boolean and integer arithmetics (v0.2). https://docs.zama.ai/concrete/ (2022)

[32] Van Beirendonck, M., D'Anvers, J.-P., Verbauwhede, I.: FPT: a Fixed-Point Accelerator for Torus Fully Homomorphic Encryption. Cryptology ePrint Archive, Paper 2022/1635 (2022). https://ia.cr/2022/1635

[33] NuCypher: A GPU implementation of fully homomorphic encryption on torus. https://github.com/nucypher/nufhe (2022)

[34] Al Badawi, A., Bates, J., Bergamaschi, F., Cousins, D.B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., *et al.*: OpenFHE: Open-source fully homomorphic encryption library. In: Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, pp. 53–63 (2022)

[35] Palisade: PALISADE Lattice Cryptography Library. https://gitlab.com/palisade/palisade-release (2022)

Appendix A Proofs

A.1 Proof of Theorem 2

Theorem 2 (Correctness & Noise Growth of $\boxdot$). *Given* GLWE *sample* $\bar{\mathbf{c}}$ *of* $\mu_c \in \mathbb{T}^{(N)}[X]$ *under* GLWE *key* $\mathbf{z}$ *and noise parameter* α*, and* GGSW *sample* $\bar{\mathbf{A}}$ *of* $m_A \in \mathbb{Z}^{(N)}[X]$ *under the same key and noise parameters, external product returns* GLWE *sample* $\bar{\mathbf{c}}' = \bar{\mathbf{A}} \boxdot \bar{\mathbf{c}}$*, which holds excess noise* $e_\boxdot$*, given by* $\big\langle \bar{\mathbf{z}}, \bar{\mathbf{c}}' \big\rangle = m_A \cdot \big\langle \bar{\mathbf{z}}, \bar{\mathbf{c}} \big\rangle + e_\boxdot$*, for which it holds*

$$\mathsf{Var}[e_\boxdot] \approx \underbrace{dNV_B\alpha^2(1+k)}_{\textit{amplified } \mathsf{GGSW} \textit{ noise}} + \underbrace{\|m_A\|_2^2 \cdot \varepsilon^2(1+kNV_{\mathbf{z}})}_{\textit{decomp. errors}}, \tag{A1}$$

where $V_{\mathbf{z}}$ *is the variance of individual coefficients of the* GLWE *key* $\mathbf{z}$ *and other parameters are as per previous definitions. If* $e_\boxdot$ *and the noise of* $\bar{\mathbf{c}}$ *are "sufficiently small",* $\bar{\mathbf{c}}'$ *encrypts* $\mathsf{msg}_{\mathbf{z}}(\bar{\mathbf{c}}') = m_A \cdot \mu_c$*, i.e., external product is indeed multiplicatively homomorphic.*

Proof. Let us denote $\bar{\mathbf{c}} = (b, \mathbf{a}) \in \mathbb{T}^{(N)}[X]^{1+k}$ and let us unfold the construction of $\bar{\mathbf{A}}$ as

$$\bar{\mathbf{A}} = \left(\begin{array}{c|c} -\mathbf{A}_0\mathbf{z} + \mathbf{e} & \mathbf{A}_0 \\ -\mathbf{A}_1\mathbf{z} + \mathbf{e} & \mathbf{A}_1 \\ \vdots & \vdots \\ -\mathbf{A}_k\mathbf{z} + \mathbf{e} & \mathbf{A}_k \end{array}\right) + \begin{pmatrix} m_A\,\mathbf{g} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & m_A\,\mathbf{g} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & m_A\,\mathbf{g} \end{pmatrix}, \tag{A2}$$

where $\mathbf{A}_i \in \mathbb{T}^{(N)}[X]^{d,k}$ with j-th column denoted $\mathbf{A}_i^{(j)}$. Unfolding the construction of $\bar{\mathbf{A}}$ and that of external product, we obtain

$$\big\langle \bar{\mathbf{z}}, \bar{\mathbf{c}}' \big\rangle_{1+k} = \big\langle (1, \mathbf{z}), \mathbf{g}^{-1}(\bar{\mathbf{c}})^T \cdot \bar{\mathbf{A}} \big\rangle_{1+k} = \big\langle \mathbf{g}^{-1}(b), m_A\,\mathbf{g} \underbrace{-\mathbf{A}_0\mathbf{z}}_{-\blacklozenge} + \mathbf{e} \big\rangle_d + \sum_{i=1}^{k} \big\langle \mathbf{g}^{-1}(a_i), \underbrace{-\mathbf{A}_i\mathbf{z}}_{-\heartsuit} + \mathbf{e} \big\rangle_d +$$

$$+ \sum_{j=1}^{k} z_j \Bigg(\underbrace{\big\langle \mathbf{g}^{-1}(b), \mathbf{A}_0^{(j)} \big\rangle_d}_{+\blacklozenge} + \underbrace{\sum_{i=1}^{k} \big\langle \mathbf{g}^{-1}(a_i), \mathbf{A}_i^{(j)} \big\rangle_d}_{+\heartsuit} + \big\langle \mathbf{g}^{-1}(a_j), m_A\,\mathbf{g} \big\rangle_d \Bigg) = \tag{A3}$$

$$= m_A \cdot \Big(\underbrace{\big\langle \mathbf{g}^{-1}(b), \mathbf{g} \big\rangle_d}_{\approx b} \pm b \Big) + m_A \sum_{j=1}^{k} z_j \Big(\underbrace{\big\langle \mathbf{g}^{-1}(a_j), \mathbf{g} \big\rangle_d}_{\approx a_j} \pm a_j \Big) + \big\langle \mathbf{g}^{-1}(b), \mathbf{e} \big\rangle_d + \sum_{i=1}^{k} \big\langle \mathbf{g}^{-1}(a_i), \mathbf{e} \big\rangle_d =$$

$$= m_A \cdot (\underbrace{b + \langle \mathbf{z}, \mathbf{a} \rangle}_{\langle \bar{\mathbf{z}}, \bar{\mathbf{c}} \rangle}) + \underbrace{m_A \cdot \Big(\langle \mathbf{g}^{-1}(b), \mathbf{g} \rangle_d - b + \sum_{j=1}^{k} z_j \big(\langle \mathbf{g}^{-1}(a_j), \mathbf{g} \rangle_d - a_j \big) \Big)}_{\text{decomp. errors: } \|m_A\|_2^2 \cdot \varepsilon^2 (1 + kNV_{\mathbf{z}})} - \tag{A4}$$

$$+ \underbrace{\big\langle \mathbf{g}^{-1}(b), \mathbf{e} \big\rangle_d + \sum_{i=1}^{k} \big\langle \mathbf{g}^{-1}(a_i), \mathbf{e} \big\rangle_d}_{\text{amplified } \mathsf{GGSW} \text{ noise: } dNV_B\alpha^2(1+k)}, \tag{A5}$$

while in (A3), terms denoted ♦ and ♥ cancel out. Next, in (A4), we assume that each decomposition error term (cf. (7)) has a uniform distribution on $[-1/2B^d, 1/2B^d)$, hence variance of ε^2; cf. (8). Finally, in (A5), we assume that the decomposition digits have a uniform distribution on $[-B/2, B/2) \cap \mathbb{Z}$, hence their mean of squares equals V_B; cf. (9). Evaluating the variance of each term, the result follows, while ≈ is due to the possible statistical dependency of variables across terms. Note that we indicate the length of inner products in lower indices. □

A.2 Proof of Theorem 3

Theorem 3 (Correctness & Noise Growth of Key-Switching)**.** *Given* LWE *sample* $\bar{\mathbf{c}}'$ *of* $\mu \in \mathbb{T}$ *under* LWE *key* $\mathbf{s}'$ *and key-switching keys* $\mathsf{KS}_{\mathbf{s}' \to \mathbf{s}}$, *encrypted with noise parameter* α', $\mathsf{KeySwitch}_{\mathbf{s}' \to \mathbf{s}}$ *returns* LWE *sample* $\bar{\mathbf{c}}$, *which holds excess noise* e_{KS}, *given by* $\langle \bar{\mathbf{s}}, \bar{\mathbf{c}} \rangle = \langle \bar{\mathbf{s}}', \bar{\mathbf{c}}' \rangle + e_{\mathsf{KS}}$, *for which it holds*

$$\mathsf{Var}[e_{\mathsf{KS}}] \approx \underbrace{n' V_{\mathbf{s}'} \varepsilon'^2}_{\textit{decomp. errors}} + \underbrace{n' d' V_{B'} \alpha'^2}_{\textit{amplif. }\mathsf{KS}\textit{ noise}}, \tag{A6}$$

where ε'^2 *and* $V_{B'}$ *are as per* (8) *and* (9), *respectively, with* B' *and* d', $V_{\mathbf{s}'}$ *is the variance of individual coefficients of the* LWE *key* $\mathbf{s}'$, *and other parameters are as per previous definitions. If* e_{KS} *and the noise of* $\bar{\mathbf{c}}'$ *are "sufficiently small", it holds* $\mu = \mathsf{msg}_{\mathbf{s}}(\bar{\mathbf{c}}) = \mathsf{msg}_{\mathbf{s}'}(\bar{\mathbf{c}}')$, *i.e.,* $\mathsf{KeySwitch}$ *indeed changes the key, without modifying the message.*

Proof. Similar to the proof of Theorem 2, we write

$$\begin{aligned}
\langle \bar{\mathbf{s}}, \bar{\mathbf{c}} \rangle &= \Big\langle (1, \mathbf{s}), (b', \mathbf{0}) - \sum_{j=1}^{n'} \mathbf{g}'^{-1}(a'_j)^T \cdot \mathsf{KS}_j \Big\rangle_{1+n} = b' + \sum_{j=1}^{n'} \Big\langle (1, \mathbf{s}), \mathbf{g}'^{-1}(a'_j)^T \cdot \underbrace{\Big[s'_j\, \mathbf{g}' - \mathbf{A}_j \mathbf{s} + \mathbf{e}_j \;\Big|\; \mathbf{A}_j \Big]}_{\mathsf{KS}_j} \Big\rangle_{1+n} = \\
&= \underbrace{b + \sum_{j=1}^{n'} s'_j \Big(\big\langle \mathbf{g}'^{-1}(a'_j), \mathbf{g}' \big\rangle_{d'} \pm a'_j \Big) \overbrace{- \sum_{j=1}^{n'} \big\langle \mathbf{g}'^{-1}(a'_j), \mathbf{A}_j \mathbf{s} \big\rangle_{d'}}^{-\heartsuit} + \sum_{j=1}^{n'} \big\langle \mathbf{g}'^{-1}(a'_j), \mathbf{e}_j \big\rangle_{d'}}_{1 \,\cdot\, \text{first element of } \sum \mathbf{g}'^{-1}(a'_j)^T \cdot \mathsf{KS}_j} + \\
&\quad \underbrace{+ \sum_{j=1}^{n'} \big\langle \mathbf{s}, \mathbf{g}'^{-1}(a'_j)^T \cdot \mathbf{A}_j \big\rangle_n}_{+\heartsuit} = \underbrace{b' + \langle \mathbf{s}', \mathbf{a}' \rangle_{n'}}_{\langle \bar{\mathbf{s}}', \bar{\mathbf{c}}' \rangle} + \underbrace{\sum_{j=1}^{n'} s'_j \Big(\big\langle \mathbf{g}'^{-1}(a'_j), \mathbf{g}' \big\rangle_{d'} - a'_j \Big)}_{\text{decomp. errors: } n' V_{\mathbf{s}'} \varepsilon'^2} + \underbrace{\sum_{j=1}^{n'} \big\langle \mathbf{g}'^{-1}(a'_j), \mathbf{e}_j \big\rangle_{d'}}_{\text{amplified } \mathsf{KS} \text{ noise: } n' d' V_{B'} \alpha'^2}
\end{aligned} \tag{A7}$$

and the result follows. □

A.3 Proof of Theorem 4

Theorem 4 (Correctness & Noise Growth of Blind-Rotate)**.** *Given inputs of Algorithm 1, where bootstrapping keys are encrypted with noise parameter* α *and test vector is (possibly) encrypted with noise parameter* α_t *(i.e.,* $\alpha_t = 0$ *or* α*),* $\mathsf{BlindRotate}$ *returns the last-step* ACC *with noise variance given by*

$$\begin{aligned}
\mathsf{Var}[\langle \bar{\mathbf{z}}, \mathsf{ACC} \rangle] \approx \alpha_t^2 + n d N V_B \alpha^2 (1 + k) + \\
+ n \varepsilon^2 (1 + k N V_{\mathbf{z}}) =: V_0,
\end{aligned} \tag{A8}$$

which we denote by V_0, *other parameters are as per previous theorems and definitions. If the noise of* $\langle \bar{\mathbf{z}}, \mathsf{ACC} \rangle$ *is "sufficiently small", it holds* $\mathsf{msg}_{\mathbf{z}}(\mathsf{ACC}) = X^{\tilde{m}} \cdot tv$, *where* $\tilde{m} = \langle \bar{\mathbf{s}}, (\tilde{b}, \tilde{\mathbf{a}}) \rangle \approx 2N\mu$, *i.e.,* $\mathsf{BlindRotate}$ *indeed "rotates" the test vector by the approximate phase of* $(b, \mathbf{a})$, *scaled to* $\mathbb{Z}_{2N}$.

Proof. The core of $\mathsf{BlindRotate}$ consists of the sample $\bar{\mathbf{t}}$ being gradually externally-multiplied by BK_i's (plus some other additions/multiplications). For i-th step with $a := \tilde{a}_i \in \mathbb{Z}_{2N}$ and $\mathsf{BK} := \mathsf{BK}_i$ that encrypts $s := s_i \in \{0, 1\}$, we write:

$$\begin{aligned}
&\big\langle \bar{\mathbf{z}}, \mathsf{ACC} + \mathsf{BK} \boxdot (X^a \cdot \mathsf{ACC} - \mathsf{ACC}) \big\rangle = \\
&= \big\langle \bar{\mathbf{z}}, \mathsf{ACC} \big\rangle + s \cdot \big\langle \bar{\mathbf{z}}, X^a \cdot \mathsf{ACC} - \mathsf{ACC} \big\rangle + e_{\boxdot}(s) = \qquad \text{(A9)} \\
&= \big\langle \bar{\mathbf{z}}, X^{s \cdot a} \cdot \mathsf{ACC} \big\rangle + e_{\boxdot}(s), \qquad \text{(A10)}
\end{aligned}$$

where (A9) is by Theorem 2 and the step towards (A10) holds for $s \in \{0, 1\}$. Hence, with each such a step, the noise grows by the additive term $e_{\boxdot}(s)$. The first step $X^{\tilde{b}} \cdot \bar{\mathbf{t}}$ retains the noise of $\bar{\mathbf{t}}$, hence the result follows. □

Appendix B Tech. Overview

In Figure B1, we provide an exhaustive technical overview of blind-rotate, preceded by modulus-switching and followed by sample-extract.

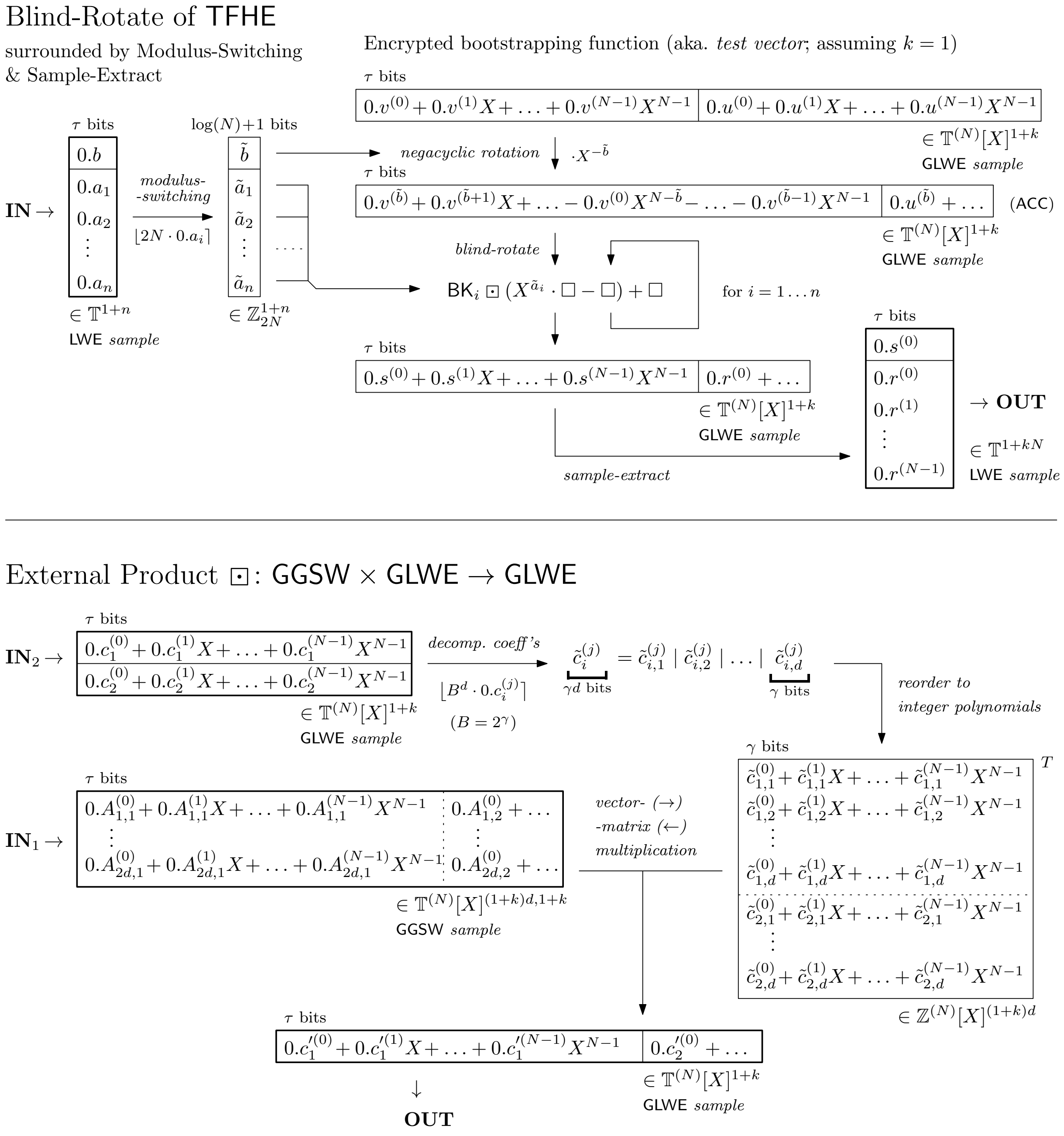
Blind-Rotate of TFHE
surrounded by Modulus-Switching & Sample-Extract
Encrypted bootstrapping function (aka. *test vector*; assuming $k=1$)
τ bits
$0.v^{(0)} + 0.v^{(1)}X + \ldots + 0.v^{(N-1)}X^{N-1}$ | $0.u^{(0)} + 0.u^{(1)}X + \ldots + 0.u^{(N-1)}X^{N-1}$
$\in \mathbb{T}^{(N)}[X]^{1+k}$
GLWE *sample*
τ bits
$\log(N)+1$ bits
IN →
$0.b$
$0.a_1$
$0.a_2$
$\vdots$
$0.a_n$
$\in \mathbb{T}^{1+n}$
LWE *sample*
modulus-
-switching
$\lfloor 2N \cdot 0.a_i \rceil$
$\tilde{b}$
$\tilde{a}_1$
$\tilde{a}_2$
$\vdots$
$\tilde{a}_n$
$\in \mathbb{Z}_{2N}^{1+n}$
negacyclic rotation
$\cdot X^{-\tilde{b}}$
τ bits
$0.v^{(\tilde{b})} + 0.v^{(\tilde{b}+1)}X + \ldots - 0.v^{(0)}X^{N-\tilde{b}} - \ldots - 0.v^{(\tilde{b}-1)}X^{N-1}$ | $0.u^{(\tilde{b})} + \ldots$
(ACC)
$\in \mathbb{T}^{(N)}[X]^{1+k}$
GLWE *sample*
blind-rotate
$\mathsf{BK}_i \boxdot (X^{\tilde{a}_i} \cdot \square - \square) + \square$
for $i = 1 \ldots n$
τ bits
$0.s^{(0)} + 0.s^{(1)}X + \ldots + 0.s^{(N-1)}X^{N-1}$ | $0.r^{(0)} + \ldots$
$\in \mathbb{T}^{(N)}[X]^{1+k}$
GLWE *sample*
sample-extract
τ bits
$0.s^{(0)}$
$0.r^{(0)}$
$0.r^{(1)}$
$\vdots$
$0.r^{(N-1)}$
→ **OUT**
$\in \mathbb{T}^{1+kN}$
LWE *sample*
External Product $\boxdot$: GGSW × GLWE → GLWE
τ bits
IN$_2$ →
$0.c_1^{(0)} + 0.c_1^{(1)}X + \ldots + 0.c_1^{(N-1)}X^{N-1}$
$0.c_2^{(0)} + 0.c_2^{(1)}X + \ldots + 0.c_2^{(N-1)}X^{N-1}$
$\in \mathbb{T}^{(N)}[X]^{1+k}$
GLWE *sample*
decomp. coeff's
$\lfloor B^d \cdot 0.c_i^{(j)} \rceil$
$(B = 2^\gamma)$
$\underbrace{\tilde{c}_i^{(j)}}_{\gamma d \text{ bits}} = \tilde{c}_{i,1}^{(j)} \mid \tilde{c}_{i,2}^{(j)} \mid \ldots \mid \underbrace{\tilde{c}_{i,d}^{(j)}}_{\gamma \text{ bits}}$
reorder to
integer polynomials
γ bits
$\tilde{c}_{1,1}^{(0)} + \tilde{c}_{1,1}^{(1)}X + \ldots + \tilde{c}_{1,1}^{(N-1)}X^{N-1}$
$\tilde{c}_{1,2}^{(0)} + \tilde{c}_{1,2}^{(1)}X + \ldots + \tilde{c}_{1,2}^{(N-1)}X^{N-1}$
$\vdots$
$\tilde{c}_{1,d}^{(0)} + \tilde{c}_{1,d}^{(1)}X + \ldots + \tilde{c}_{1,d}^{(N-1)}X^{N-1}$
$\tilde{c}_{2,1}^{(0)} + \tilde{c}_{2,1}^{(1)}X + \ldots + \tilde{c}_{2,1}^{(N-1)}X^{N-1}$
$\vdots$
$\tilde{c}_{2,d}^{(0)} + \tilde{c}_{2,d}^{(1)}X + \ldots + \tilde{c}_{2,d}^{(N-1)}X^{N-1}$
T
$\in \mathbb{Z}^{(N)}[X]^{(1+k)d}$
τ bits
IN$_1$ →
$0.A_{1,1}^{(0)} + 0.A_{1,1}^{(1)}X + \ldots + 0.A_{1,1}^{(N-1)}X^{N-1}$ | $0.A_{1,2}^{(0)} + \ldots$
$\vdots$
$0.A_{2d,1}^{(0)} + 0.A_{2d,1}^{(1)}X + \ldots + 0.A_{2d,1}^{(N-1)}X^{N-1}$ | $0.A_{2d,2}^{(0)} + \ldots$
$\in \mathbb{T}^{(N)}[X]^{(1+k)d,1+k}$
GGSW *sample*
vector- (→)
-matrix (←)
multiplication
τ bits
$0.c_1'^{(0)} + 0.c_1'^{(1)}X + \ldots + 0.c_1'^{(N-1)}X^{N-1}$ | $0.c_2'^{(0)} + \ldots$
$\in \mathbb{T}^{(N)}[X]^{1+k}$
GLWE *sample*
↓
OUT

Fig. B1 Technical overview of TFHE Blind-Rotate, preceded by Modulus-Switching and followed by Sample-Extract, with a detail on External Product.