Setting Up Efficient TFHE Parameters for Multivalue Plaintexts and Multiple Additions [*]

Jakub Klemsa

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

Abstract

Unlike traditional and/or standardized ciphers, TFHE offers much space for the setup of its parameters. Not only the parameter choice affects the plaintext space size and security, it also greatly impacts the performance of TFHE, in particular, its bootstrapping.

In this paper, we provide an exhaustive description of TFHE, including its foundations, (functional) bootstrapping and error propagation during all operations. In addition, we outline a bootstrapping scenario without the key switching step. Based on our thorough summary, we suggest an approach for the setup of TFHE parameters with particular respect to bootstrapping efficiency. Finally, we propose twelve setups of real-world TFHE parameters for six different scenarios with and without key switching, respectively, and we compare their performance.

N.b.: This is a technical paper, which is mainly intended for researchers interested in TFHE. However, due to its self-containment, it shall be accessible also for readers with a basic knowledge of TFHE.

1 Introduction

The fully homomorphic encryption scheme by Chillotti et al. named TFHE (Fully Homomorphic Encryption over the Torus; [4]) has received a lot of attention recently [1, 2, 3, 7, 8]. However, to the best of our knowledge, there is no study focusing on the choice of the TFHE parameters, let alone with respect to the bootstrapping efficiency. Indeed, existing papers refer to the original TFHE paper [4] or to the TFHE Library [11].

Our Contributions

In this paper, we present the results of our study on the TFHE parameter setup, while we particularly focus on the bootstrapping efficiency. Most notably, we suggest an approach that allows for tuning the plaintext space size as well as the limit of homomorphic additions. We demonstrate our method on six different <plaintext space size> : <homomorphic additions limit> setups. These setups are intended for our prospective applications, hence they have currently no direct motivation.

[*]A related study on error-free computations with TFHE is to appear at Computing Conference '21 in London.

Paper Outline

In Section 2, we revisit TFHE, in particular we focus on multivalue plaintext space and multiple ciphertext additions. Next, we provide a detailed analysis of TFHE error propagation in Section 3. We propose an approach for TFHE parameter setup with particular respect to bootstrapping efficiency in Section 4. Finally, in Section 5, we provide TFHE parameter sets for various usage scenarios together with performance measurements. We conclude our paper in Section 6.

2 TFHE with Multivalue Plaintext Space

In this section, we revisit TFHE by Chillotti et al. [4] as well as subsequent enhancements [2, 3]. We explain the (modified) bootstrapping algorithms in closer detail as a prerequisite for parameter derivation in the following sections.

Symbols & Notation. Since this paper has a lot of technical content, we introduce frequent symbols and notations at a 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
  - $\mathbf{w}^{(j)} = (w_0^{(j)}, \ldots, w_{n-1}^{(j)})$,
- $a \overset{\$}{\leftarrow} M$ and $a \overset{\alpha}{\leftarrow} M$ the uniform and the zero-centered α-deviated draw, respectively, of a random variable a from M.

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

Unlike multiplication, the division of a torus 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].

2.1 TFHE Samples

TFHE is a fully homomorphic cipher, which employs internally two encryption schemes: T(R)LWE and TRGSW. It uses TLWE to encrypt the 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 the notation across papers (we follow the updated notation from [4], which was introduced at the end of Section 3 of that paper).

2.1.1 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*, $\alpha \in \mathbb{R}_0^+$ be the *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 let 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

$$b = m + \mathbf{k} \cdot \mathbf{a} + e, \tag{1}$$

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

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

Definition 2 (T(R)LWE phase)**.** Let n, N and α be 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 $\varphi_\mathbf{k} \colon \mathbb{T}^{(N)}[X]^k \times \mathbb{T}^{(N)}[X] \to \mathbb{T}^{(N)}[X]$,

$$\varphi_\mathbf{k}(\mathbf{a}, b) = b - \mathbf{k} \cdot \mathbf{a}, \tag{2}$$

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 call $\mathsf{msg}(\mathbf{c}) \coloneqq \mathrm{E}\big[\varphi_\mathbf{k}(\mathbf{c})\big]$ the *message* of sample $\mathbf{c}$, which equals m for a valid sample, since the noise is zero-centered.

Note 2.1. 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 errors in our decrypted results, which can be considered useful, e.g., in the context of differential privacy [5], or
2. a correctable decryption – for this purpose, we need to control the amount of noise and follow the TRLWE phase function by appropriate rounding (relevant for this paper), or
3. an expectation of the TRLWE phase (i.e., $\mathsf{msg}(\mathbf{c})$) – this is useful for formal definitions and proofs.

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

Theorem 1 (Additive Homomorphism [4])**.** *Let* $\mathbf{c}_1, \ldots, \mathbf{c}_n$ *be valid and independent* TRLWE *samples under a* TRLWE *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*

$$\mathsf{msg}\Big(\sum_{i=1}^n e_i \cdot \mathbf{c}_i\Big) = \sum_{i=1}^n e_i \cdot \mathsf{msg}(c_i) \tag{3}$$

and the noise amplitude and variance are bounded by

$$\|\mathsf{Err}(\mathbf{c})\|_\infty \le \sum_{i=1}^{n} \|e_i\|_1 \cdot \|\mathsf{Err}(\mathbf{c}_i)\|_\infty, \text{ and} \tag{4}$$

$$\mathsf{Var}\big(\mathsf{Err}(\mathbf{c})\big) \le \sum_{i=1}^{n} \|e_i\|_2^2 \cdot \mathsf{Var}\big(\mathsf{Err}(\mathbf{c}_i)\big), \text{ respectively.} \tag{5}$$

2.1.2 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 [4])**.** Let $B_g = 2^\gamma$ for some $\gamma \in \mathbb{N}$ and $l \in \mathbb{N}$ be the decomposition parameters and let N and n be the TRLWE degree and dimension, respectively. We call

$$\mathbf{H} = \left(\begin{array}{c|c|c} 1/B_g & \dots & 0 \\ \vdots & \ddots & \vdots \\ 1/B_g^l & \dots & 0 \\ \hline \vdots & \ddots & \vdots \\ \hline 0 & \dots & 1/B_g \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1/B_g^l \end{array}\right), \tag{6}$$

$\mathbf{H} \in \mathbb{T}^{(N)}[X]^{(n+1)l,n+1}$, 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 [4]
(for gadget matrix $\mathbf{H}$, quality $\beta = B_g/2$ and precision $\varepsilon = 1/2B_g^l$)

```
Input: TRLWE sample $(\mathbf{a}, b) = \big(a_1(X), \dots, a_n(X), b = a_{n+1}(X)\big) \in \mathbb{T}^{(N)}[X]^{n+1}$,
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:     $\bar{a}_i^{(j)} \leftarrow \lfloor B_g^l \cdot a_i^{(j)} \rceil$
3:     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:     $d_{(i-1)l+p}(X) = \sum_{j=0}^{N-1} \bar{a}_{i,p}^{(j)} X^j$
6: return d
```

Note 2.2. 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 [4])**.** *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 [4])**.** Let n, N and α be the parameters of TRLWE with a 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 the 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 the key $\mathbf{k}$.

Definition 5 (External Product [4])**.** For the decomposition parameters β and ε, we define the *External Product*, $\boxdot$: TRGSW $\times$ TRLWE $\to$ TRLWE, as

$$\mathbf{A} \boxdot \mathbf{b} = \mathsf{Dec}_{\mathbf{H},\beta,\varepsilon}(\mathbf{b})^T \cdot \mathbf{A}, \tag{7}$$

where TRLWE is the underlying cipher of TRGSW.

Theorem 3 (Multiplicative Homomorphism [4])**.** *Let* $\mathbf{k}$ *be a* TRLWE *key,* $\mathbf{A}$ *a valid* TRGSW *sample of* $m_A \in \mathbb{Z}^{(N)}[X]$ *under the 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 the key* $\mathbf{k}$*.*

2.1.3 Security of T(R)LWE

For T(R)LWE encryption to be secure, a key length $\leftrightarrow$ noise balance must be met. It holds that *the longer key, the better security*, as well as *the bigger noise, the better security.* However, the noise magnitude must be kept low, since with too much noise, correct decryption (as per Note 2.1, point 2.) cannot be guaranteed. We propose an approach for finding the security $\leftrightarrow$ usability $\leftrightarrow$ performance balance together with the guarantee of correct decryption with high probability in Section 4.

A practical security analysis of T(R)LWE is provided by Chillotti et al. [4], Section 7. They expressed the effective bit-security parameter λ as a function of the entropy of the T(R)LWE key (at most its dimension n) and the standard deviation of the noise (denoted by α). They ran a numeric optimization to obtain the final estimate on λ and they summarized their results in a figure. Based on their figure, we introduce a parameter s_λ, which stands for the slope of an equi-λ line:

$$-\log(\alpha_n) \approx s_\lambda \cdot n. \tag{8}$$

Note that due to the collision attack, the slope is further left-bounded by $n = 2\lambda$; see the original figure ([4], Figure 9). We summarize approximate values of s_λ in Table 1.

λ	40	80	128	192	256	384	512
s_λ	0.051	0.040	0.033	0.028	0.024	0.020	0.017

Table 1: Approximate values of the parameter s_λ.

2.2 Bootstrapping

The procedure referred to as *bootstrapping* aims at reducing the internal noise of a TLWE sample to a certain fixed level, while it internally runs the decryption procedure. In addition, it is capable of function evaluation at no extra cost.

TFHE bootstrapping consists of the following three algorithms: BlindRotate, SampleExtract and KeySwitch. In this paper, we only recall their basic practical variants, e.g., we fix the TRGSW dimension, which will no longer be different than 1. In addition, we particularly focus on multivalue plaintext space and 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 using encrypted key bits. With the still-encrypted resulting "plaintext", it "blindly rotates" its second input – a TRLWE sample, which encodes (and possibly encrypts) the evaluated function in the 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 finally KeySwitch does the job and switches the key to the original one (if applicable).

2.2.1 Blind Rotate

BlindRotate is the cornerstone of bootstrapping, since this is where the homomorphic decryption is performed, 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 be given 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 a TRLWE key $k'(X)$, referred to as *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. Then (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 2.3. When multiplied by a power of X, the coefficients of a polynomial modulo $X^N + 1$ rotate *negacyclically* with a period of $2N$. For this reason, the rest of the *actually* evaluated bootstrapping function f is the negacyclic extension of its first N values, i.e., we have rather $f\colon \mathbb{Z}_{2N} \to \mathbb{T}$ and it holds that

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

In the 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. Cf. Figure 1, which illustrates the idea of BlindRotate, how it looks in the plain domain.

Note 2.4. Due to (12), $\bar{a}_i$'s and $\bar{b}$ will be scaled to $\mathbb{Z}_{2N}$ as $\bar{a} = \lfloor 2Na \rceil$.

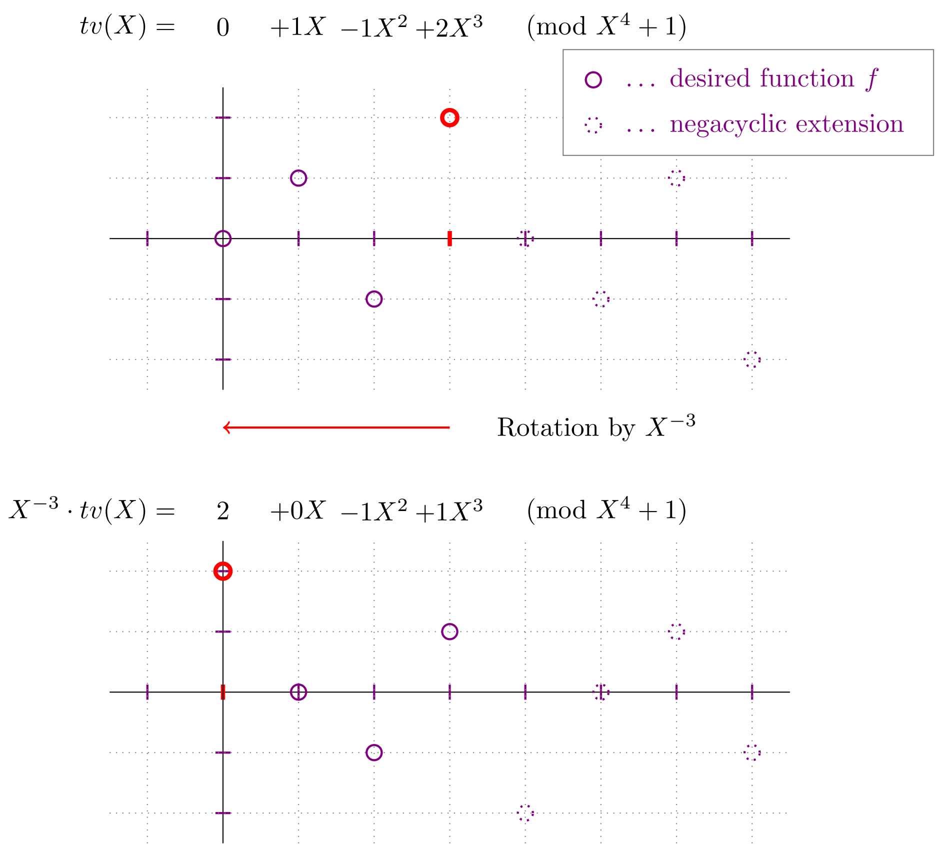


Figure 1: BlindRotate for $m = 3$, i.e., rotation by X^{-3}, in the plain domain. Desired value $f(3)$ is emphasized in red. For simplicity, the values are rescaled to integers.

Note 2.5. 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. [12] suggested 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 to group the terms k_i into pairs. Bourse et al. [2] further improved the technique by Zhou et al. by reducing the number of required bootstrapping keys from 4 to 3 (per pair of TLWE 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 4 adds a +ACC term, which is missing in [2]).

Algorithm 2 $\mathsf{BlindRotateIm}$ ([4], improved by [2, 12])

Input: TLWE sample $(\mathbf{a}, b) \in \mathbb{T}^{n+1}$ 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 $\bar{m} = \big(\bar{b} - \sum_{i=1}^{n} k_i \cdot \bar{a}_i\big) \bmod 2N$.

```
1: $\bar{a}_i \leftarrow \lfloor 2Na_i \rceil$ for $i \in [1, n]$, $\bar{b} \leftarrow \lfloor 2Nb \rceil$
2: $\mathsf{ACC} \leftarrow X^{-\bar{b}} \cdot (u, v)$    // aka. accumulator
3: for $i \in [1, n/2]$ do
4:     $\mathsf{ACC} \leftarrow \big((X^{a_{2i-1}+a_{2i}} - 1)\mathsf{BK}_{3i-2} + (X^{a_{2i-1}} - 1)\mathsf{BK}_{3i-1} + (X^{a_{2i}} - 1)\mathsf{BK}_{3i}\big) \boxdot \mathsf{ACC} + \mathsf{ACC}$
5: return $\mathsf{ACC}$
```

Theorem 4 ($\mathsf{BlindRotateIm}$ Error)**.** *Algorithm 2 returns a sample with error (variance) bounded as follows:*

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

$$\mathsf{Var}\big(\mathsf{Err}(\mathsf{ACC})\big) \le 6nlN\beta^2\,\mathsf{Var}\big(\mathsf{Err}(\mathsf{BK})\big) + n(1+N)\varepsilon^2 + \mathsf{Var}\big(\mathsf{Err}(u,v)\big). \tag{15}$$

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 $\mathbf{A} = \mathbf{Z_A} + m_\mathbf{A} \cdot \mathbf{H}$. We write the line 4 of Algorithm 2 as (with simplified indexes as per (13))

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

$$\boxdot\, \mathsf{ACC} + \mathsf{ACC} = \tag{16}$$

$$= \underbrace{\mathsf{Dec}(\mathsf{ACC})^T}_{\mathbf{d}} \cdot \mathsf{BK}_\Sigma + \mathsf{ACC} = \ldots = \tag{17}$$

$$= \underbrace{(X^{a+a'} - 1) \cdot \big(\mathbf{d} \cdot \mathbf{Z}_{\mathsf{BK}_{kk'}} + kk' \cdot (\mathbf{d} \cdot \mathbf{H})\big)}_{\text{from } (X^{a+a'}-1)\mathsf{BK}_{kk'} \text{ term}} + \ldots + \mathsf{ACC} = \ldots \tag{18}$$

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

$$\ldots = (X^{a+a'} - 1) \cdot \Big(\underbrace{\mathbf{d} \cdot \mathbf{Z}_{\mathsf{BK}_{kk'}}}_{\heartsuit} + kk' \cdot \big(\underbrace{\mathsf{ACC}}_{\clubsuit} + \underbrace{\boldsymbol{\varepsilon}_{\mathsf{ACC}}}_{\spadesuit}\big)\Big) +$$

$$+ (X^{a} - 1) \cdot \Big(\underbrace{\mathbf{d} \cdot \mathbf{Z}_{\mathsf{BK}_{k(1-k')}}}_{\heartsuit} + k(1-k') \cdot \big(\underbrace{\mathsf{ACC}}_{\clubsuit} + \underbrace{\boldsymbol{\varepsilon}_{\mathsf{ACC}}}_{\spadesuit}\big)\Big) +$$

$$+ (X^{a'} - 1) \cdot \Big(\underbrace{\mathbf{d} \cdot \mathbf{Z}_{\mathsf{BK}_{(1-k)k'}}}_{\heartsuit} + (1-k)k' \cdot \big(\underbrace{\mathsf{ACC}}_{\clubsuit} + \underbrace{\boldsymbol{\varepsilon}_{\mathsf{ACC}}}_{\spadesuit}\big)\Big) + \underbrace{\mathsf{ACC}}_{\clubsuit}. \tag{19}$$

By Chillotti et al. [4], proof of Theorem 3.13:

- $\heartsuit$ is bounded by $2lN\beta E_{\mathsf{BK}}$,

- ♣ carries the 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 the keys multiply to 1.

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

$$\|\mathsf{Err}(\mathsf{ACC}_{\mathrm{new}})\|_\infty \leq 3 \cdot 2 \cdot 2lN\beta E_{\mathsf{BK}} + \|\mathsf{Err}(\mathsf{ACC}_{\mathrm{old}})\|_\infty + 2 \cdot (1+N)\varepsilon. \tag{20}$$

The loop begins with $\|\mathsf{Err}(\mathsf{ACC}_0)\|_\infty = \|\mathsf{Err}(u,v)\|_\infty$ and it keeps adding a constant term $n/2$-times, cf. (20), hence the result (14) follows. By an analogical approach, we obtain the bound on the error variance. □

Note that the enhancement by Zhou et al., further improved by Bourse et al., aims at reducing the total number of external products, which is the most demanding operation. Indeed, the original BlindRotate algorithm [4] loops all n indices (instead of only $n/2$), for which it computes

$$\mathsf{ACC} \leftarrow \mathsf{BK}_i \boxdot (X^{a_i} \cdot \mathsf{ACC} - \mathsf{ACC}) + \mathsf{ACC}, \tag{21}$$

where BK_i encrypts the i-th bit of the global TLWE key $\mathbf{k}$. At this point, it is interesting to observe that (21) can be written also in a form, which resembles the improved variant (Algorithm 2, line 4), and which encrypts the same:

$$\mathsf{ACC} \leftarrow \big((X^{a_i} - 1)\mathsf{BK}_i\big) \boxdot \mathsf{ACC} + \mathsf{ACC}. \tag{22}$$

However, the fundamental difference between (21) and (22) 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.

2.2.2 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 is to extract the constant term in the form of a TLWE sample.

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

$$m^{(0)} = s^{(0)} - \underbrace{\big(k^{(0)}, k^{(1)}, \dots, k^{(N-1)}\big)}_{\text{new } \mathsf{TLWE} \text{ key } \mathbf{k}' = \mathsf{coeffs}(k')} \cdot \big(r^{(0)}, -r^{(N-1)}, \dots, -r^{(1)}\big). \tag{23}$$

It follows that $\big((r^{(0)}, -r^{(N-1)}, \dots, -r^{(1)}), s^{(0)}\big)$ 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 [4]).

Note 2.6. 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 3.3.

Note 2.7. SampleExtract preserves the error – in fact, it only re-arranges coefficients.

Algorithm 3 SampleExtract ([4], 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 $(\mathbf{a}', b') = \big((r^{(0)}, -r^{(N-1)}, \dots, -r^{(1)}), s^{(0)}\big)$
```

2.2.3 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 ([4])

```
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 $(k'_1, \dots, k'_N) = \mathbf{k}'$ (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 $[\bar{a}'_{i,1}, \bar{a}'_{i,2}, \dots, \bar{a}'_{i,t}]$ 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 [4]). *Algorithm 4 returns a sample with error (variance) bounded as follows:*

$$\|\mathsf{Err}(\mathbf{a}, b)\|_\infty \le \|\mathsf{Err}(\mathbf{a}', b')\|_\infty + tN\|\mathsf{Err}(\mathsf{KS})\|_\infty + 2^{-(t+1)}N, \tag{24}$$

$$\mathsf{Var}\big(\mathsf{Err}(\mathbf{a}, b)\big) \le \mathsf{Var}\big(\mathsf{Err}(\mathbf{a}', b')\big) + tN\,\mathsf{Var}\big(\mathsf{Err}(\mathsf{KS})\big) + 2^{-2(t+1)}N. \tag{25}$$

2.2.4 TFHE Bootstrapping

The bootstrapped TLWE sample $(\mathbf{a}, b)$ is first scaled and rounded by the BlindRotate algorithm, while rounding may introduce a rounding error. A lemma follows.

Lemma 2.1 (Pre-BlindRotate Error). *The scaled and rounded sample $(\bar{\mathbf{a}}, \bar{b})$ has its error (variance) bounded as follows:*

$$\Big\|\mathsf{Err}\Big(\frac{\bar{\mathbf{a}}}{2N}, \frac{\bar{b}}{2N}\Big)\Big\|_\infty \le \|\mathsf{Err}(\mathbf{a}, b)\|_\infty + \underbrace{\frac{n+1}{4N}}_{E_{round}}, \tag{26}$$

$$\mathsf{Var}\Big(\mathsf{Err}\Big(\frac{\bar{\mathbf{a}}}{2N}, \frac{\bar{b}}{2N}\Big)\Big) \le \mathsf{Var}\big(\mathsf{Err}(\mathbf{a}, b)\big) + \underbrace{\frac{n+1}{48N^2}}_{V_{round}}. \tag{27}$$

where we denote the rounding error terms as E_{round} and V_{round}, respectively.

Proof. After scaling $(\bar{\mathbf{a}}, \bar{b})$ back to the torus domain by ${}^{1}/_{2N}$, rounding to multiples of ${}^{1}/_{2N}$ may change each of the $n+1$ terms of the sample $(\mathbf{a}, b)$ at most by ${}^{1}/_{2} \cdot \frac{1}{2N}$. Variance of the rounding error is then derived from the uniform and independent distribution of a_i's. □

Note 2.8. The error and variance bounds (26) and (27), respectively, are the greatest bounds among bootstrapping before the noise is refreshed. Therefore, we denote the bounds as

$$E_{max} \coloneqq \|\mathsf{Err}(\mathbf{a}, b)\|_\infty + E_{round}, \quad \text{and} \tag{28}$$

$$V_{max} \coloneqq \mathsf{Var}\big(\mathsf{Err}(\mathbf{a}, b)\big) + V_{round}, \tag{29}$$

respectively.

Note 2.9. By (27) and by the 3-sigma rule, we have that $3\times$ standard deviation of the rounding error (scaled to $\mathbb{Z}_{2N}$)

$$2N \cdot 3\sigma_{round} = \frac{\sqrt{3(n+1)}}{2} > \frac{1}{2}, \tag{30}$$

i.e., only the rounding error is likely to change the value, where the bootstrapping function $f \colon \mathbb{Z}_{2N} \to \mathbb{T}$ is evaluated. To evaluate the function correctly, we need to introduce another (and sparser) precision for the global TFHE plaintexts. From now on, we denote the *plaintext space bit-precision* by π, i.e., the plaintext space will be $\mathbb{Z}_{2^\pi}$ and it follows that $2^\pi < 2N$. Specific relation of π and N will be discussed in Section 3.3, where we propose a proper test vector generation procedure.

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

Algorithm 5 Multivalue TFHE Bootstrapping

```
Input: TFHE parameters: $\pi$ (plaintext precision), $n$ (dimension), $N$ (TRLWE degree), $\gamma$, $l$ 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 $tv \in \mathbb{T}^{(N)}[X]$ 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: $(r, s) \leftarrow \mathsf{BlindRotateIm}\big((\mathbf{a}, b), (u, v), \mathsf{BK}_{\mathbf{k}\to k'}\big)$
2: $(\mathbf{a}', b') \leftarrow \mathsf{SampleExtract}\big((r, s)\big)$
3: return $\mathsf{KeySwitch}\big((\mathbf{a}', b'), \mathsf{KS}_{\mathbf{k}'\to\mathbf{k}}\big)$
```

Corollary 6 (Bootstrapping Error [4]). *Algorithm 5 returns a sample $(\mathbf{a}'', b'')$ with error (variance) bounded as follows:*

$$\|\mathsf{Err}(\mathbf{a}'', b'')\|_\infty \le \underbrace{6nlN\beta E_{\mathsf{BK}} + n(1+N)\varepsilon + \|\mathsf{Err}(u, v)\|_\infty + tNE_{\mathsf{KS}} + 2^{-(t+1)}N}_{E_0}, \tag{31}$$

$$\mathsf{Var}\big(\mathsf{Err}(\mathbf{a}'', b'')\big) \le \underbrace{6nlN\beta^2 V_{\mathsf{BK}} + n(1+N)\varepsilon^2 + \mathsf{Var}\big(\mathsf{Err}(u, v)\big) + tNV_{\mathsf{KS}} + 2^{-2(t+1)}N}_{V_0}. \tag{32}$$

where we denoted the error (variance) bound of a freshly bootstrapped sample by E_0 and V_0, respectively.

3 Error Propagation in TFHE

In this section, we provide a thorough analysis of TFHE noise propagation during homomorphic operations, based on which we derive a set of limitations on the TFHE parameters. In particular, we focus on the multivalue TFHE variant and we derive our results with respect to the 3-sigma rule, which is based on the error variance.

3.1 Overview of Error Propagation

During homomorphic addition and bootstrapping, the error evolves as follows:

Addition: The maximum error is additive by (4), the variance is additive with quadratic weights by (5), provided that the noise of the involved samples is independent; cf. Theorem 1.

Bootstrapping: If the noise of the sample-to-be-bootstrapped is smaller than a certain bound, bootstrapping evaluates the bootstrapping function correctly (i.e., at the correct point). On average, the resulting sample carries a fixed amount of noise, independent of the original sample; cf. Corollary 6.

In Figure 2, we summarize the error propagation in terms of respective maximum error bounds. Note that the overall maximum of relative noise is achieved within bootstrapping, right after the initial rounding step; cf. Note 2.8.

$$\underbrace{E_0}_{\substack{\text{fresh/bootstrapped}\\ \text{sample(s)}}} \xrightarrow{\text{addition}} \underbrace{n_\oplus \cdot E_0}_{\substack{\text{pre-bootstrap}\\ (E_{pre})}} \overbrace{\xrightarrow{\text{rounding}} \underbrace{n_\oplus \cdot E_0 + E_{round}}_{\substack{\text{pre-}\mathsf{BlindRotate}\\ (E_{max})}} \xrightarrow[\text{etc...}]{\mathsf{BlindRotate},}}^{\text{bootstrapping}} \underbrace{E_0}_{\substack{\text{bootstrapped}\\ \text{sample}}} \tag{33}$$

Figure 2: Error propagation during TFHE addition and bootstrapping, cf. (26), (28) and (31). Number of additions is bounded by $n_\oplus$.

Note 3.1. Since a freshly bootstrapped sample plays a similar role as a freshly encrypted sample, we will assume and demand equal error magnitude for both.

Note 3.2. We commit on a finite representation of torus elements (i.e., coefficients of samples) as follows: we employ a τ-bit integral type, while such a τ-bit number t represents $t/2^\tau \in \mathbb{T}$. With this approach, we cover the range $[0, 1)$ uniformly and we use the precision most efficiently, unlike with a floating-point type. We call τ the *sample precision*, also referred to as the *torus precision*.

3.2 Test Vector Generation

Let us discuss the process of encoding the desired (negacyclic!) bootstrapping function $\bar{f}\colon \mathbb{Z}_{2^\pi} \to \mathbb{Z}_{2^\pi}$, where $\bar{f}$ acts on plaintexts, into an 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.

By Note 2.9, we have that the (relative) rounding error exceeds $1/2N$, i.e., the error of $\bar{m}$ can be greater than 1. Therefore, we need to expand the (negacyclic) bootstrapping function $\bar{f}\colon \mathbb{Z}_{2^\pi} \to \mathbb{Z}_{2^\pi}$ into a *staircase* function $f\colon \mathbb{Z}_{2N} \to \mathbb{T}$—before it is encoded into the test vector tv—as

$$f(k) = \bar{f}\Big(\Big\lfloor \frac{k}{2^{\nu+1-\pi}} \Big\rceil\Big), \quad k \in [0, 2N-1], \tag{34}$$

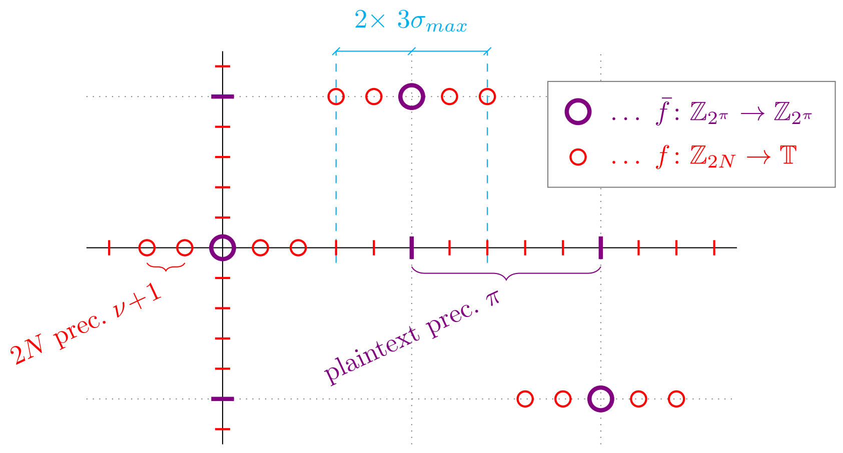
cf. Figure 3. The function f is then encoded into tv—as outlined in (10)—as follows:

$$tv^{(k)} = f(k), \quad k \in [0, N-1]. \tag{35}$$

Recall that since f is negacyclic, the remaining values do not need to be encoded, cf. Note 2.3.

As soon as the actual maximum error is smaller than a half of the plaintext precision with high probability, i.e., $3\sigma_{max} < 1/2^{\pi+1}$, the erroneous value of $\bar{m}$ does not leave the "stair" and the bootstrapping function $\bar{f}$ is evaluated as expected; cf. Figure 3. For the maximum variance V_{max}, defined in (29), we have the fundamental condition

$$\boxed{V_{max} \leq \frac{1}{3^2 \cdot 2^{2\pi+2}}.} \tag{36}$$



$2\times\ 3\sigma_{max}$
$\ldots\ \bar{f}\colon \mathbb{Z}_{2^\pi} \to \mathbb{Z}_{2^\pi}$
$\ldots\ f\colon \mathbb{Z}_{2N} \to \mathbb{T}$
2N prec. $\nu+1$
plaintext prec. π

Figure 3: Conversion of a negacyclic bootstrapping function $\bar{f}$ into a staircase function f. N.b., both functions are appropriately scaled.

3.3 Key Switching: Employ, or Omit?

We propose two possible bootstrapping scenarios: we either employ, or we omit KeySwitch, as outlined in Note 2.6. Recall that KeySwitch aims at changing the key of a TLWE sample from (the bootstrapping TRLWE key interpreted as) a TLWE key of length N to the original TLWE key of length n. For each scenario, we derive a set of inequalities between the TFHE parameters.

First, let us introduce a new parameter Δ, which aims at expressing the amount of homomorphic additions from the error variance point of view. We write the homomorphic addition of independent

TLWE samples (c_i) using a series of integers (w_i) (aka. *weights*) as $C = \sum w_i c_i$. We bound the weights as follows:

$$\sum w_i^2 \leq 2^{2\Delta}. \tag{37}$$

We also assume that the function f is known to the evaluating party, hence the test vector can be given in the form of a trivial (error-free) TRLWE sample.

Note 3.3. Let us outline our error-splitting heuristics: 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 key switching scenarios, the bound (36) on V_{max} (a guarantee of correct evaluation and decryption with high probability) can be satisfied as follows:

$$V_{max} \leq \underbrace{2^{2\Delta} V_0}_{1/(3^2 \cdot 2^{2\pi+3})} + \underbrace{V_{round}}_{1/(3^2 \cdot 2^{2\pi+3})} \overset{!}{\leq} \frac{1}{3^2 \cdot 2^{2\pi+2}}, \tag{38}$$

where we applied the parameter Δ from (37) and the heuristics from Note 3.3.

Rounding Error Variance. Note that the rounding error variance term V_{round} is independent of whether KeySwitch is employed, or not, hence it can be discussed for both scenarios together. By (27), we have

$$V_{round} \leq \frac{n+1}{48N^2} \overset{!}{\leq} \frac{1}{3^2 \cdot 2^{2\pi+3}}, \tag{39}$$

which yields

$$N \geq \sqrt{6(n+1)} \cdot 2^{\pi-1}. \tag{40}$$

3.3.1 Employ KeySwitch

The additive error variance term $2^{2\Delta} V_0$ of (38) can be estimated by (32) and bounded as follows:

$$\begin{aligned} 2^{2\Delta} V_0 \leq & \underbrace{2^{2\Delta-1} \cdot 3nlN2^{2\gamma} V_{\mathsf{BK}}(N)}_{1/(3^2 \cdot 2^{2\pi+5}) \quad (\heartsuit)} + \underbrace{2^{2\Delta} n(1+N) 2^{-2(\gamma l+1)}}_{1/(3^2 \cdot 2^{2\pi+5}) \quad (\diamondsuit)} + \underbrace{2^{2\Delta}\, \mathsf{Var}\big(\mathsf{Err}(u,v)\big)}_{=0} + \\ & + \underbrace{2^{2\Delta} t N V_{\mathsf{KS}}(n)}_{1/(3^2 \cdot 2^{2\pi+5}) \quad (\clubsuit)} + \underbrace{2^{2\Delta} 2^{-2(t+1)} N}_{1/(3^2 \cdot 2^{2\pi+5}) \quad (\spadesuit)} \overset{!}{\leq} \frac{1}{3^2 \cdot 2^{2\pi+3}}, \end{aligned} \tag{41}$$

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 support the error variance terms of bootstrapping and key switching keys V_{BK} and V_{KS} with their bit entropy, which is N and n, respectively. Recall that the 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 the logarithmic domain, we write:

$$2\pi + 4 + 3\log(3) + 2\Delta + \log(n) + \log(N) + \log(l) + 2\gamma + \log\big(V_{\mathsf{BK}}(N)\big) \leq 0, \tag{$\heartsuit$}$$

$$2\pi + 3 + 2\log(3) + 2\Delta + \log(n) + \log(N+1) - 2\gamma l \leq 0, \tag{$\diamondsuit$}$$

$$2\pi + 5 + 2\log(3) + 2\Delta + \log(N) + \log(t) + \log\big(V_{\mathsf{KS}}(n)\big) \leq 0, \tag{$\clubsuit$}$$

$$2\pi + 3 + 2\log(3) + 2\Delta + \log(N) - 2t \leq 0. \tag{$\spadesuit$}$$

3.3.2 Omit KeySwitch

As outlined in Note 2.6, we can set the TRGSW/TRLWE key $k'(X)$ such that $\mathsf{coeffs}(k') = \mathbf{k}$ (we fill the rest with zeros), and then we can omit KeySwitch. N.b., the bit entropy of $k'(X)$ is no longer N – it is only n. For a thorough discussion on the subspace LWE, we refer to Pietrzak [9].

Note 3.4. 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 [10].

On the one hand, we get rid of the error terms ($\clubsuit$) and ($\spadesuit$) in (41), which gives us more room for the plaintext space, additions and/or security. Another advantage is that we simplify the overall bootstrapping process, which may lead to a performance improvement, most likely in an FPGA implementation, which favors a few operations that are performed repeatedly.

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

Following an approach analogous to the previous one, we obtain:

$$2\pi + 3 + 3\log(3) + 2\Delta + \log(n) + \log(N) + \log(l) + 2\gamma + \log\big(V_{\mathsf{BK}}(n)\big) \leq 0, \tag{$\heartsuit$}$$

$$2\pi + 2 + 2\log(3) + 2\Delta + \log(n) + \log(N+1) - 2\gamma l \leq 0, \tag{$\diamondsuit$}$$

where we changed the entropy used by bootstrapping keys from N to n.

4 TFHE Parameter Setup for Efficient Bootstrapping

In this section, we propose a heuristic approach for a practical TFHE parameter derivation. Given the plaintext space size and the desired level of security, our goal is to obtain a set of TFHE parameters with particular respect to bootstrapping efficiency. We follow the 3σ approach and the limitations proposed in the previous section for both key switching scenarios. Our ultimate goal is to satisfy (36), which we achieve by demanding (40), and either ($\blacktriangledown$)–($\spadesuit$), or ($\heartsuit$) and ($\diamondsuit$), respectively.

Note 4.1. In case we are free to choose the torus precision τ (e.g., for an FPGA implementation), we suggest to use the uniform error distribution on $\{-1/2^\tau, 0, 1/2^\tau\}$. We have

$$\tau = -\log(\alpha) - {}^1\!/_2\big(\log(3) - 1\big), \tag{42}$$

we can further apply $-\log(\alpha) = -{}^1\!/_2 \log\big(V_{\mathsf{BK/KS}}\big)$; cf. Section 2.1.3 for the definition of α.

4.1 Generic Approach

Let us describe our approach for the TFHE parameter derivation, while we consider both key switching scenarios, i.e., with or without KeySwitch. We input the following parameters:

- desired bit-security λ,
- plaintext bit-precision π, and
- limit on the additive weights 2Δ as per (37).

Next, we aim at satisfying (40) and ($\heartsuit$)–($\spadesuit$) for the "with KeySwitch" scenario, or (40), ($\heartsuit$) and ($\diamondsuit$) for "without KeySwitch". Note that some parameters are immediately fixed, while others can be further tuned. E.g., ($\diamondsuit$) requires γl to be greater than a certain bound, which gives us a space to tune the ratio of these parameters to achieve a better performance. See Algorithm 6, where we outline our generic approach in the form of a pseudo-algorithm.

Note 4.2. Algorithm 6 gives no guarantee on the correctness, let alone the optimality of the returned TFHE parameters. It is intended rather as an auxiliary guideline, whereas a certain portion of insight and intuition must be applied, in particular on line 33. Hence, it does not need to be strictly followed, in particular its final step, where the $\gamma : l$ ratio is tuned. Indeed, increasing N (while fine-tuning the $\gamma : l$ ratio) may lead to better parameters, in particular if we would end up with $l \geq 3$. In such a case, it shows that increasing N to $2N$ and decreasing l to 1 (if possible) leads to more efficient parameters. On the other hand, this does not appear to be an advantage for $l \leq 2$.

Algorithm 6 Generic TFHE Parameter Derivation

```
Input: Security level $\lambda$, plaintext precision $\pi$, weight limit $2\Delta$.
Output: TFHE parameters $n$, $N$, $\gamma$, $l$, $(t)$, $-\log(\alpha_{BK})$, $\big(-\log(\alpha_{KS})\big)$.
 1: set minimum feasible $n_{min} \leftarrow 2\lambda$
 2: set initial estimate on $N$ from (40) (n.b., $N$ must be a power of two and $N \geq 2\lambda$)
 3: calculate bound $n_{\max}$ from equality in (40)
 4: if Key Switching then
 5:     get $t$ form (♠)
 6:     get $-\log\big(V_{\mathsf{KS}}(n)\big)$ form (♣)
 7:     calculate $-\log(\alpha_{\mathsf{KS}}(n)) = -{}^{1}\!/_{2}\log\big(V_{\mathsf{KS}}(n)\big)$
 8:     if $s_\lambda \cdot n > -\log\big(\alpha_{\mathsf{KS}}(n)\big)$ then
 9:         if $n = n_{\max}$ then
10:             $N \leftarrow 2 \cdot N$
11:             go to 3
12:         $n \leftarrow \min\big\{{}^{-\log(\alpha_n)}\!/_{s_\lambda}, n_{max}\big\}$        ▷ ceil $n$ (e.g., to a multiple of 10)
13:         go to 6
14: set initial $\gamma \leftarrow 1$
15: get $l$ from (♦), or from (◇), respectively
16: get $-\log\big(V_{\mathsf{BK}}(N)\big)$ form (♥), or $-\log\big(V_{\mathsf{BK}}(n)\big)$ form (♡), respectively
17: calculate $-\log(\alpha_{\mathsf{BK}}(N))$, or $-\log(\alpha_{\mathsf{BK}}(n))$, respectively
18: if Key Switching then
19:     if $s_\lambda \cdot N > -\log\big(\alpha_{\mathsf{BK}}(N)\big)$ then
20:         $N \leftarrow 2 \cdot N$
21:         go to 3
22: else
23:     if $s_\lambda \cdot n > -\log\big(\alpha_{\mathsf{BK}}(n)\big)$ then
24:         if $n = n_{\max}$ then
25:             $N \leftarrow 2 \cdot N$
26:             go to 3
27:         $n \leftarrow \min\big\{{}^{-\log(\alpha_n)}\!/_{s_\lambda}, n_{max}\big\}$        ▷ ceil $n$ (e.g., to a multiple of 10)
28:         go to 15
29: fine tune $\gamma$, $l$:
30:     if $l = 1$ or $(\gamma, l)$ "not to be changed" then
31:         go to 34
32:     increase $\gamma$ s.t. $l$ can be decreased (n.b., $\gamma l$ is lower-bounded by (♦), or (◇), respectively)
33:     go to 16
            (consider rolling back to 32 in case $N$ shall be increased,
             /$\gamma$ was increased too much/; cf. Note 4.2)
34: return $\big(n, N, \gamma, l, t, -\log(\alpha_{BK}(N)), -\log(\alpha_{KS}(n))\big)$, or $\big(n, N, \gamma, l, -\log(\alpha_{BK}(n))\big)$, resp.
```

5 Proposed Parameters & Experimental Results

In this section, we put forward practical TFHE parameters for six different $\pi : 2\Delta$ scenarios, while we commit on $\lambda = 128$-bit security, unlike 256-bit security that is used in the TFHE Library [11]. The scenarios, motivated by our actual research, are as follows (cf. (37) for the definition of Δ):

A. 2-bit plaintexts, weights $(1, 1)$, i.e., $2^{2\Delta} = 1^2 + 1^2 = 2$,

B. 2-bit plaintexts, weights $(1, 1, 1)$, i.e., $2^{2\Delta} = 3$,

C. 4-bit plaintexts, weights $(1, 1, 1)$, i.e., $2^{2\Delta} = 3$,

D. 4-bit plaintexts, weights $(1, 1, 1, 1, 4, 4)$, i.e., $2^{2\Delta} = 36$,

E. 5-bit plaintexts, weights $(1, 1, 1, 1, 4, 4)$, i.e., $2^{2\Delta} = 36$, and

F. 7-bit plaintexts, weights $(1, 1, 6, 6)$, i.e., $2^{2\Delta} = 74$.

5.1 Parameters with KeySwitch

We present our TFHE parameters with KeySwitch and their benchmarking results in Table 2. For a Ruby code to generate these parameters, we refer to Appendix A.1.

[11]	$\pi = 2$	$2^{2\Delta} \geq 2$	$n = 630$	$N = 1\,024$	$t = 8$	165 ms
	$\gamma = 7$	$l = 3$	$-\log(\alpha_{KS}) = 15$	$-\log(\alpha_{BK}) = 25$	$\lambda \approx 256$	
A	$\pi = 2$	$2^{2\Delta} = 2$	$n = 400$	$N = 1\,024$	$t = 11$	54 ms
	$\gamma = 15$	$l = 1$	$-\log(\alpha_{KS}) \approx 13.31$	$-\log(\alpha_{BK}) \approx 31.20$	$\lambda \approx 128$	
B	$\pi = 2$	$2^{2\Delta} = 3$	$n = 420$	$N = 1\,024$	$t = 11$	57 ms
	$\gamma = 16$	$l = 1$	$-\log(\alpha_{KS}) \approx 13.61$	$-\log(\alpha_{BK}) \approx 32.53$	$\lambda \approx 128$	
C	$\pi = 4$	$2^{2\Delta} = 3$	$n = 480$	$N = 1\,024$	$t = 13$	92 ms
	$\gamma = 9$	$l = 2$	$-\log(\alpha_{KS}) \approx 15.73$	$-\log(\alpha_{BK}) \approx 28.12$	$\lambda \approx 128$	
D	$\pi = 4$	$2^{2\Delta} = 36$	$n = 540$	$N = 1\,024$	$t = 15$	106 ms
	$\gamma = 10$	$l = 2$	$-\log(\alpha_{KS}) \approx 17.62$	$-\log(\alpha_{BK}) \approx 31.00$	$\lambda \approx 128$	
E	$\pi = 5$	$2^{2\Delta} = 36$	$n = 570$	$N = 1\,024$	$t = 16$	113 ms
	$\gamma = 11$	$l = 2$	$-\log(\alpha_{KS}) \approx 18.67$	$-\log(\alpha_{BK}) \approx 33.04$	$\lambda \approx 128$	
F	$\pi = 7$	$2^{2\Delta} = 74$	$n = 680$	$N = 4\,096$	$t = 20$	512 ms
	$\gamma = 24$	$l = 1$	$-\log(\alpha_{KS}) \approx 22.35$	$-\log(\alpha_{BK}) \approx 49.19$	$\lambda \approx 128$	

Table 2: Original TFHE Library parameters (delimited) and our TFHE parameters for various scenarios with KeySwitch, supported by the measured mean time per bootstrapping. Recall that $\tau = -\log(\alpha) - 1/2\big(\log(3) - 1\big) \approx -\log(\alpha) - 0.29$.

5.2 Parameters without KeySwitch

In Table 3, we propose TFHE parameters without KeySwitch, for the Ruby code we refer to Appendix A.2. However, since the code of the TFHE Library is quite rigid, modifying it to the "without KeySwitch" variant and running benchmarks is out of the scope of this paper.

A	$\pi = 2$	$2\Delta = \log(2)$	$n = 990$	$N = 1\,024$
	$\gamma = 16$	$l = 1$	$-\log(\alpha_{BK}) \approx 32.35$	$\lambda \approx 128$
B	$\pi = 2$	$2\Delta = \log(3)$	$n = 990$	$N = 1\,024$
	$\gamma = 16$	$l = 1$	$-\log(\alpha_{BK}) \approx 32.65$	$\lambda \approx 128$
C	$\pi = 4$	$2\Delta = \log(3)$	$n = 860$	$N = 1\,024$
	$\gamma = 9$	$l = 2$	$-\log(\alpha_{BK}) \approx 28.04$	$\lambda \approx 128$
D	$\pi = 4$	$2\Delta = \log(36)$	$n = 940$	$N = 1\,024$
	$\gamma = 10$	$l = 2$	$-\log(\alpha_{BK}) \approx 30.90$	$\lambda \approx 128$
E	$\pi = 5$	$2\Delta = \log(36)$	$n = 1\,310$	$N = 2\,048$
	$\gamma = 21$	$l = 1$	$-\log(\alpha_{BK}) \approx 43.14$	$\lambda \approx 128$
F	$\pi = 7$	$2\Delta = \log(74)$	$n = 1\,550$	$N = 8\,192$
	$\gamma = 25$	$l = 1$	$-\log(\alpha_{BK}) \approx 50.78$	$\lambda \approx 128$

Table 3: Our TFHE parameters for various scenarios without KeySwitch. Recall that $\tau = -\log(\alpha) - {}^1\!/_2\big(\log(3) - 1\big) \approx -\log(\alpha) - 0.29$.

6 Discussion & Conclusion

Our pseudo-algorithm for TFHE parameter setup shall be considered only as an auxiliary tool to make an initial estimate. In particular, our splitting heuristics does not guarantee the best parameter tradeoff by any means. We believe that for concrete requirements on plaintext space and security, there is still some space to improve the suggested TFHE parameters. On the other hand, we identified several generic relations and limitations that need to be taken into account for prospective TFHE parameter setups anyways.

However, we were positively surprised by the experimental results. Most notably, even for $\pi = 5$, the bootstrapping time is only about twice more than for $\pi = 2$ with our parameters. We conjecture that the product $N \cdot l$ plays the most significant role in the overall bootstrapping performance. Hence, for $\pi = 5$, where N is still as low as $1\,024$, the bootstrapping times are convenient. This allows us to process bigger portions of data in a single bootstrap and gives more freedom for any prospective usecases of functional bootstrapping.

References

[1] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. Chimera: Combining ring-lwe-based fully homomorphic encryption schemes. *Journal of Mathematical Cryptology*, 14(1):316–338, 2020.

[2] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Annual International Cryptology Conference*, pages 483–512. Springer, 2018.

[3] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. New techniques for multi-value input homomorphic evaluation and applications. In *Cryptographers' Track at the RSA Conference*, pages 106–126. Springer, 2019.

[4] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[5] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[6] Fork of 'TFHE: Fast Fully Homomorphic Encryption Library over the Torus'. `https://github.com/fakub/tfhe`, 2021.

[7] Antonio Guimarães, Edson Borin, and Diego F Aranha. Revisiting the functional bootstrap in tfhe. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 229–253, 2021.

[8] Kotaro Matsuoka, Ryotaro Banno, Naoki Matsumoto, Takashi Sato, and Song Bian. Virtual secure platform: A five-stage pipeline processor over {TFHE}. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[9] Krzysztof Pietrzak. Subspace lwe. In *Theory of Cryptography Conference*, pages 548–563. Springer, 2012.

[10] Ron D Rothblum. On the circular security of bit-encryption. In *Theory of Cryptography Conference*, pages 579–598. Springer, 2013.

[11] TFHE: Fast Fully Homomorphic Encryption Library over the Torus. `https://github.com/tfhe/tfhe`, 2016.

[12] Tanping Zhou, Xiaoyuan Yang, Longfei Liu, Wei Zhang, and Ningbo Li. Faster bootstrapping with multiple addends. *IEEE Access*, 6:49868–49876, 2018.

Appendix

A Ruby Commands for TFHE Parameter Calculation

Below we provide Ruby commands, which can be used for TFHE parameter calculation. Run commands in an interactive Ruby shell, e.g., in `irb`. First, we initialize a global constant `S`, which holds the security parameters s_λ for different λ's as per Table 1, and we select the desired security level $\lambda = 128$.

```
#   Hash of security parameters s_lambda
S = {_40: 0.051, _80: 0.04, _128: 0.033, _192: 0.028, _256: 0.024, _384: 0.02, \
     _512: 0.017}
lambda = :_128
```

A.1 Parameters with KeySwitch

```
#   Example initial setups with suggested parameters                               # :1-2
#   ------------------------------------------------------------------------------------
pi  = 2 ; _2D = 1 ;              nu = 10 ; n = 400 ; gamma = 15                    #  A
#   ------------------------------------------------------------------------------------
pi  = 2 ; _2D = Math.log2(3)  ; nu = 10 ; n = 420 ; gamma = 16                    #  B
#   ------------------------------------------------------------------------------------
pi  = 4 ; _2D = Math.log2(3)  ; nu = 10 ; n = 480 ; gamma =  9                    #  C
#   ------------------------------------------------------------------------------------
pi  = 4 ; _2D = Math.log2(36) ; nu = 10 ; n = 540 ; gamma = 10                    #  D
#   ------------------------------------------------------------------------------------
pi  = 5 ; _2D = Math.log2(36) ; nu = 10 ; n = 570 ; gamma = 11                    #  E
#   ------------------------------------------------------------------------------------
pi  = 7 ; _2D = Math.log2(74) ; nu = 12 ; n = 680 ; gamma = 24                    #  F
#   ------------------------------------------------------------------------------------
nn      = 2**nu ; n_max = 1.0 * nn**2 / (3 * 2**(2*pi-1)) - 1                     # :3
t       = ((2*pi +3 +2*Math.log2(3) +_2D +nu)/2).ceil                             # :5
logVKSn = -(2*pi +5 +2*Math.log2(3) +_2D +nu +Math.log2(t))                       # :6
logAKSn = logVKSn / 2                                                             # :7
tauKS   = 0.5 * (-logVKSn - Math.log2(3) + 1)
l       = ((2*pi +3 +2*Math.log2(3) +_2D +Math.log2(n) +nu) / (2*gamma)).ceil    # :15
logVBKnn= -2*pi -4 -3*Math.log2(3) -_2D -Math.log2(n) -nu -Math.log2(l) -2*gamma# :16
logABKnn= logVBKnn / 2                                                            # :17
tauBK   = 0.5 * (-logVBKnn - Math.log2(3) + 1)
#   tune gamma & l (if applicable), and recalculate & check:
print "-log(al_KS(n)) = #{-logAKSn}, n = #{n}  |  bnd = #{S[lambda] * n}"       # :8
puts " ... #{-logAKSn < S[lambda] * n ? 'OK' : '!!!'}"
print "-log(al_BK(N)) = #{-logABKnn}, N = #{nn}  |  bnd = #{S[lambda] * nn}"    # :19
puts " ... #{-logABKnn < S[lambda] * nn ? 'OK' : '!!!'}"
#   print final results
printf("    n = %4d  |  N = %4d\n", n, nn)
printf("gamma = %4d  |  l = %4d  |  t = %4d\n", gamma, l, t)
printf("tauKS ~ %.3f |  tauBK ~ %.3f\n", tauKS, tauBK)
```

A.2 Parameters without KeySwitch

```
#   Example initial setups with suggested parameters                                 # :1-2
#   ------------------------------------------------------------------------------------
pi  = 2 ; _2D = 1 ;             nu = 10 ; n =  990 ; gamma = 16                      #  A
#   ------------------------------------------------------------------------------------
pi  = 2 ; _2D = Math.log2(3)  ; nu = 10 ; n =  990 ; gamma = 16                      #  B
#   ------------------------------------------------------------------------------------
pi  = 4 ; _2D = Math.log2(3)  ; nu = 10 ; n =  860 ; gamma =  9                      #  C
#   ------------------------------------------------------------------------------------
pi  = 4 ; _2D = Math.log2(36) ; nu = 10 ; n =  940 ; gamma = 10                      #  D
#   ------------------------------------------------------------------------------------
pi  = 5 ; _2D = Math.log2(36) ; nu = 11 ; n = 1310 ; gamma = 21                      #  E
#   ------------------------------------------------------------------------------------
pi  = 7 ; _2D = Math.log2(74) ; nu = 13 ; n = 1550 ; gamma = 25                      #  F
#   ------------------------------------------------------------------------------------
nn      = 2**nu ; n_max = 1.0 * nn**2 / (3 * 2**(2*pi-1)) - 1                        # :3
l       = ((2*pi +2 +2*Math.log2(3) +_2D +Math.log2(n) +nu) / (2*gamma)).ceil   # :15
logVBKn = -2*pi -3 -3*Math.log2(3) -_2D -Math.log2(n) -nu -Math.log2(l) -2*gamma# :16
logABKn = logVBKn / 2                                                                # :17
tauBK   = 0.5 * (-logVBKn - Math.log2(3) + 1)
#   tune gamma & l (if applicable), and recalculate & check:
print "-log(al_BK(n)) = #{-logABKn}, n = #{n}  |  bnd = #{S[lambda] * n}"       # :19
puts " ... #{-logABKn < S[lambda] * n ? 'OK' : '!!!'}"
#   print final results
printf("    n = %4d  |  N = %4d\n", n, nn)
printf("gamma = %4d  |  l = %4d\n", gamma, l)
printf("tauBK ~ %.3f\n", tauBK)
```