information

MDPI

*Article*

# Integer-Wise Functional Bootstrapping on TFHE: Applications in Secure Integer Arithmetics †

**Hiroki Okada [1,*], Shinsaku Kiyomoto [1] and Carlos Cid [2,3]**

[1] KDDI Research, Inc., Saitama 356-8502, Japan; kiyomoto@kddi-research.jp
[2] Royal Holloway, University of London, London WC1E 7HU, UK; Carlos.Cid@rhul.ac.uk
[3] Simula UiB, Thormøhlens Gate 53D, 5006 Bergen, Norway
* Correspondence: ir-okada@kddi-research.jp
† This paper is an extended version of our presentation in the 23rd Information Security Conference (ISC 2020), Online, 16–20 December 2020.

**Abstract:** TFHE is a fast fully homomorphic encryption scheme proposed by Chillotti et al. in Asiacrypt' 2018. Integer-wise TFHE is a generalized version of TFHE that can encrypt the plaintext of an integer that was implicitly presented by Chillotti et al., and Bourse et al. presented the actual form of the scheme in CRYPTO' 2018. However, Bourse et al.'s scheme provides only homomorphic integer additions and homomorphic evaluations of a sign function. In this paper, we construct a technique for operating any 1-variable function in only one bootstrapping of the integer-wise TFHE. For applications of the scheme, we also construct a useful homomorphic evaluation of several integer arithmetics: division, equality test, and multiplication between integer and binary numbers. Our implementation results show that our homomorphic division is approximately 3.4 times faster than any existing work and that its run time is less than 1 second for 4-bit integer inputs.

**Keywords:** fully homomorphic encryption; secure computation; secure division; LWE; TFHE



**Citation:** Okada, H.; Kiyomoto, S.; Cid, C. Integer-Wise Functional Bootstrapping on TFHE: Applications in Secure Integer Arithmetics. *Information* **2021**, *12*, 297. https://doi.org/10.3390/info12080297

Academic Editor: Giovanni Russello

Received: 5 July 2021
Accepted: 22 July 2021
Published: 26 July 2021



## 1. Introduction

Gentry first proposed a fully homomorphic encryption (FHE) construction in 2009 [1,2], and a variety of new features and hardness assumptions for FHE schemes [3–13] have been considered. The FHE performs computations on encrypted data without decryption of the data.

One notable application is secure computation in order to protect the secrecy of data against computation servers. An example of secure computation services is privacy-preserving machine learning as a service (MLaaS) [14,15]. In such a service, users want to hide sensitive data (e.g., genomic, anamnesis data, etc.) from the server and the server does not want to provide its cognitive models to users. FHE should be a fundamental component of secure multi-party computation (MPC). However, most applications are constructed without FHE operations because the MPC may require FHE operations many times and the run time of the operations would be substantial due to the inefficiency of FHE schemes.

Most FHE schemes consist of *bit-wise* or *integer-wise* encryption operations; the plaintext space is $\mathbb{Z}_2$ in *bit-wise* encryption, and the plaintext space is $\mathbb{Z}_p$ for some $p > 2$ in *integer-wise* encryption [16–18]. Some schemes have been executed via integer arithmetics, but bit-wise integer addition and multiplication are not practical even though the bit-wise integer comparison algorithm is efficient [19]. In particular, the full adder algorithm has to execute homomorphic multiplication $l$ times for an $l$-bit length integer. Thus, homomorphic integer multiplication based on the full adder must execute approximately $l^2$ [18] operations.

On the other hand, *integer-wise* FHE schemes can efficiently perform addition and multiplication because the construction of those calculations in a homomorphic manner is

merely integer addition and multiplication. Recent studies [14,15,20,21] on secure MLaaS used integer-wise homomorphic encryption. Secure integer-wise comparison [22] and integer-wise division [23] have recently been proposed; however, arithmetic operations such as division are remain impractical. The design of efficient algorithms for basic arithmetic operations is an open issue required to reduce computational costs of the operations.

HElib [24], HEAAN [25], PALISADE [26], SEAL [27], and TFHE [6–8] are open-source implementations of homomorphic encryption. TFHE is a variant of the FHEW scheme [28] in which the bootstrapping procedure can be performed in less than 0.1 seconds, which is a major bottleneck of FHE. TFHE successfully improved the overall performance of arithmetic operations, although TFHE does not explicitly provide integer-wise arithmetics. Moreover, the number of bootstrapping operations required is equal to the number of bits. Bourse et al. [14] modified TFHE to enable integer-wise encryption, which only supports homomorphic integer addition and homomorphic evaluation of the sign function.

*Contribution*

In this paper, we present an approach where an arbitrary function is executed during a bootstrap procedure in the integer-wise TFHE scheme with a generalization of the constant variable "test vector" in the bootstrapping. Our bootstrapping procedure is applicable to homomorphic operations of fundamental arithmetics, where specific settings of the value of test vector are given: We propose the homomorphic equality test (Eq, Algorithm 6) and its variant, which test the equality between a ciphertext and a plaintext (ConstEq, Algorithm 7); the homomorphic multiplication between an integer and a binary number (MultbyBin, Algorithm 8); and homomorphic division, where the dividend is a ciphertext and the divisor is a constant plaintext (DivbyConst, Algorithm 9). Based on these building blocks, we propose a homomorphic algorithm for integer division (Div, Algorithm 10). Our implementation of Divruns approximately 3.4 times faster than the fastest existing work. We also generalize the Divalgorithm to a homomorphic evaluation of an any 2-variable function.

## 2. Preliminaries

### *2.1. Notation*

We write the security parameter as $\lambda$. We denote by $\mathbb{T}$ the torus of real numbers modulo 1, $\mathbb{R}/\mathbb{Z}$. For any ring $\mathcal{R}$, polynomials of the variable $X$ with coefficients in $\mathcal{R}$ are denoted by $\mathcal{R}[X]$. We define $\mathbb{R}_N[X] := \mathbb{R}[X]/(X^N+1)$, $\mathbb{Z}_N[X] := \mathbb{Z}[X]/(X^N+1)$, and their quotient $\mathbb{T}_N[X] := \mathbb{R}_N[X]/\mathbb{Z}_N[X]$, which are the ring of polynomials of variable $X$ with quotient $X^N+1$ and real coefficients modulo 1. Let $\mathbb{B} := \{0,1\}$ be a set, and we write vectors in bold. We write $\mathbf{s} \xleftarrow{\$} \mathcal{S}$ when $\mathbf{s}$ is uniformly random sampled from $\mathcal{S}$. We denote by $\mathcal{U}(\mathcal{S})$ the uniform distribution over the set $\mathcal{S}$, and we denote by $e \leftarrow \chi$ the process of sampling $e$ according to the distribution $\chi$.

### *2.2. Background of TFHE*

#### 2.2.1. Learning with Errors

The learning with errors (LWE) problem is a computational problem introduced by Regev [29]. We use the torus variant of LWE, as in [6,30]. Let $n \in \mathbb{N}$, and let $\chi$ be a statistical distribution on $\mathbb{R}$. The torus LWE distribution for a fixed uniformly random secret vector $\mathbf{s} \xleftarrow{\$} \mathbb{B}^n$ is defined as follows:

$$\mathsf{LWE}_{n,\mathbf{s},\chi} := \{(\mathbf{a}, b) | \mathbf{a} \xleftarrow{\$} \mathbb{T}^n, e \leftarrow \chi, b = \mathbf{a} \cdot \mathbf{s} + e \in \mathbb{T}\}.$$

The search-LWE problem is to find the secret vector $\mathbf{s}$ from LWE samples $\{(\mathbf{a}_i, \mathbf{a}_i \cdot \mathbf{s} + e_i)\}_{i=1}^m$ sampled from $\mathsf{LWE}_{n,\mathbf{s},\chi}$. The decision-LWE problem is to distinguish whether samples $\{(\mathbf{a}_i, \mathbf{s} \cdot \mathbf{a}_i + e_i)\}_{i=1}^m$ are drawn from $\mathsf{LWE}_{n,\mathbf{s},\chi}$ or $\mathcal{U}(\mathbb{T}^{n+1})$.

TFHE uses a *sub-Gaussian* distribution for the error distribution. A distribution $\chi_\sigma$ over $\mathbb{R}$ such that $\forall t \in \mathbb{R}, E(e^{tX}) \leq e^{\sigma^2 t^2/2}$ is called $\sigma$-sub-Gaussian. Let $Y$ and $Z$ be independent

random variables that are $\sigma$- and $\sigma'$-sub-Gaussian, respectively. Then, for all $k$, $l \in \mathbb{R}$, $k \cdot Y + l \cdot Z$ is $\sqrt{k^2\sigma^2 + l^2\sigma'^2}$-sub-Gaussian.

#### 2.2.2. Bit-Wise LWEEncryption

We briefly explain bit-wise Regev's encryption [29], which is a building block of the (bit-wise) TFHE. Let $m \in \mathbb{B}$ be a plaintext. The encryption scheme works as follows:

- $\mathsf{Setup}(\lambda)$: Let the security parameter be $\lambda$. Select and fix public parameters $n = n(\lambda)$, $\sigma = \sigma(\lambda)$; then, $\mathsf{Setup}(\lambda)$ samples a secret key $\mathbf{s} \xleftarrow{\$} \mathbb{B}^n$ and outputs it.
- $\mathsf{Enc}(\mathbf{s}, m)$: a uniformly random vector $\mathbf{a} \xleftarrow{\$} \mathbb{T}^n$ and a noise $e \leftarrow \mathcal{D}_{\mathbb{T}_N[X],\sigma}$ are sampled, where $\mathcal{D}_{\mathbb{T}_N[X],\sigma}$ is the Gaussian distribution over $\mathbb{T}_N[X]$ with a standard deviation $\sigma$. Then, the algorithm outputs a ciphertext $(\mathbf{a}, b)$, where

$$b = \mathbf{a} \cdot \mathbf{s} + e + m/2. \tag{1}$$

- $\mathsf{Dec}(\mathbf{s}, (\mathbf{a}, b))$: The decryption algorithm returns $\lceil 2(b - \mathbf{a} \cdot \mathbf{s}) \rfloor$. It outputs plaintext correctly if the size of noise $e$ is bounded as

$$|e| < 1/4, \tag{2}$$

Since $2(b - \mathbf{a} \cdot \mathbf{s}) = 2e + m$, $|2e| < \frac{1}{2}$, thus $\lceil 2(b - \mathbf{a} \cdot \mathbf{s}) \rfloor = m$.

We explain later in Section 2.4 that this scheme can be extended to integer-wise encryptions.

#### 2.2.3. Torus LWE(TLWE)

The General-LWEproblem (GLWE) proposed by Brakerski, Gentry, and Vaikuntanathan [31] is a general problem that includes the LWE problem, and the Ring-LWE problem proposed by Lyubashevsky, Peikert, and Regev in [32]. TLWE is a torus analogue of the GLWE problem. Let $k$ be a positive integer, $N$ be a power of 2, and $\chi$ be a probability distribution over $\mathbb{R}_N[X]$. A TLWE secret key $\bar{\mathbf{s}}$ is a vector of $k$ polynomials over $\mathbb{Z}_N[X]$ with binary coefficients, denoted as $\bar{\mathbf{s}} \in \mathbb{B}_N[X]^k$. Given a polynomial message $\mu \in \mathbb{T}_N[X]$, a "fresh" TLWE ciphertext of $\mu$ under the key $\bar{\mathbf{s}}$ is a TLWE sample $(\bar{\mathbf{a}}, \bar{b}) \in \mathbb{T}_N[X]^k \times \mathbb{T}_N[X]$, where $\bar{\mathbf{a}} \leftarrow \mathbb{T}_N[X]^k$ and $\bar{b} = \bar{\mathbf{s}} \cdot \bar{\mathbf{a}} + \mu + e$, where $e \leftarrow \chi$.

Let $(\bar{\mathbf{a}}, \bar{b})$ be a TLWE ciphertext of a polynomial message $\mu \in \mathbb{T}_N[X]$ under a TLWE key $\bar{\mathbf{s}} \in \mathbb{B}_N[X]^k$. Notice that the coefficients of the TLWE sample is an LWE ciphertext of the constant term of the polynomial message $\mu$. We define the function SampleExtract that simply extracts the coefficients vector of the TLWE sample, and we denote the vector of coefficients of a polynomial $a \in \mathbb{T}_N(X)$ by $\mathsf{coefs}(a(X))$. Then, we can obtain the extracted LWE ciphertext as $(\mathbf{a}', b') := \mathsf{SampleExtract}((\bar{\mathbf{a}}, \bar{b}))$, where $\mathbf{a}' = (\mathsf{coefs}(\bar{a}_1(1/X)), \ldots, \mathsf{coefs}(\bar{a}_k(1/X)))$ and $b'$ is the constant term of the polynomial $\bar{b}$. This is the ciphertext encrypted with an extracted key $\mathbf{s}' := \mathsf{KeyExtract}(\bar{s}) := (\mathsf{coefs}(\bar{s}_1(X)), \ldots, \mathsf{coefs}(\bar{s}_k(X))) \in \mathbb{Z}^{kN}$.

As mentioned before, TLWE includes the LWE problem and the Ring-LWE problem. When $N > 1$ is large and $k = 1$, TLWE is the (binary-secret) Ring-LWE. Conversely, when $N = 1$ and $k$ is large, TLWE is the (binary-secret) LWE. Note that, in the TFHE library [33], $N = 1024$ and $k = 1$ by default; thus, TLWE is simply a Ring-LWE in the library, i.e., we can simple consider the TLWE samples as Ring-LWE samples $(a, b) \in \mathbb{T}_N[X] \times \mathbb{T}_N[X]$ in this paper.

#### 2.2.4. TGSW

TGSW is a generalized variant of the GSW FHE scheme proposed by Gentry, Sahai, and Waters [10]. TGSW can be regarded as the matrix extension of TLWEbecause GSW is the matrix extension of LWE. Each row of a TGSW sample is a TLWE sample. As with TFHE [6], an external product $\boxdot$ that maps $\boxdot : \mathsf{TGSW} \times \mathsf{TLWE} \to \mathsf{TLWE}$ can be defined. The product of the TGSW ciphertext of a polynomial message $\mu_{\mathsf{TGSW}} \in \mathbb{T}_N[X]$ and the TLWE ciphertext of a polynomial message $\mu_{\mathsf{TLWE}} \in T_N[X]$ becomes a TLWE ciphertext

of a polynomial message $(\mu_{\mathsf{TGSW}} \cdot \mu_{\mathsf{TLWE}}) \in T_N[X]$. We refer the readers to [6] for more details.

### 2.3. Overview of TFHE Bootstrapping

We refer to the bootstrapping procedure of the TFHE in [6]. We present the bootstrapping algorithm in Algorithm 1. As we mentioned before, in the default settings of the TFHE library [33], the dimension of the TLWE sample is set as $k = 1$. This means that the TLWE sample is simply the Ring-LWEsample $(\bar{a}, \bar{b}) \in \mathbb{T}_N[X] \times \mathbb{T}_N[X]$. In the rest of this paper, we set $k = 1$: we also consider the TLWE sample to be the Ring-LWEsample $(a, b) \in \mathbb{T}_N[X] \times \mathbb{T}_N[X]$.

**Algorithm 1:** TFHE bootstrapping [6] for binary arithmetics.

**Input:** An LWE sample $(\mathbf{a}, b) \in \mathsf{LWE}_{\mathbf{s}}(m_{\text{in}})$ whose plaintext $m_{\text{in}} \in \{0, 1\}$, a constant $m_{\text{set}} \in \{0, 1\}$, a bootstrapping key $\mathsf{BK}_{\mathbf{s}\to\mathbf{s}'',\alpha}$, and a keyswitching key $\mathsf{KS}_{\mathbf{s}'\to\mathbf{s},\gamma}$, where $\mathbf{s}' = \mathsf{KeyExtract}(\mathbf{s}'')$.

**Output:** An LWE sample $\mathsf{LWE}_{\mathbf{s}}(m_{\text{out}})$, where $m_{\text{out}} = m_{\text{in}} \cdot m_{\text{set}}$.

1 $\mu := \frac{m_{\text{set}}}{2} \in \mathbb{T}$, $\mu' = \frac{\mu}{2} \in \mathbb{T}$

2 $\bar{b} := \lceil 2Nb \rfloor$, $\bar{a_i} := \lceil 2Na_i \rfloor$ for each $i \in [1, n]$

3 $\mathsf{testv} := (1 + X + \cdots + X^{N-1}) \cdot X^{\frac{N}{2}} \cdot \mu' \in \mathbb{T}_N[X]$

4 $\mathsf{ACC} \leftarrow X^{\bar{b}} \cdot (\mathbf{0}, \mathsf{testv}) \in \mathbb{T}_N[X] \times \mathbb{T}_N[X]$

5 **for** $i = 1$ to $n$ **do** $\mathsf{ACC} \leftarrow [\mathbf{h} + (X^{-\bar{a}_i} - 1) \cdot \mathsf{BK}_i] \boxdot \mathsf{ACC}$

// $\mathsf{ACC} = \mathsf{TLWE}(X^{(\bar{b} - \bar{a}s)} \cdot \mathtt{testv}) = \mathsf{TLWE}(X^{(m_{\text{in}}N + e)} \cdot \mathtt{testv})$

6 $\mathbf{u} := (\mathbf{0}, \mu') + \mathsf{SampleExtract}(\mathsf{ACC})$ // $\mathsf{msg}(\mathsf{SampleExtract}(\mathsf{ACC})) = \mu' \ (m_{\text{in}} = 1), \ -\mu' \ (m_{\text{in}} = 0)$

7 **return** $\mathsf{KeySwitch}_{\mathsf{KS}}(\mathbf{u})$

#### 2.3.1. Input

The input for the bootstrapping is an LWE ciphertext of message binary message $m_{\text{in}} \in \mathbb{B}$, $(\mathbf{a}, b) \in \mathbb{T}^n \times \mathbb{T}$, where $b = \mathbf{a} \cdot \mathbf{s} + e + \frac{m_{\text{in}}}{2}$. The noise $e$ is sampled according to the distribution such that $|e| < \frac{1}{4}$ to satisfy the condition for correct decryption of the LWE encryption.

#### 2.3.2. Rounding

After the rounding operation in line 2, we obtain an rounded LWE ciphertext $(\bar{\mathbf{a}}, \bar{b}) \in \mathbb{Z}_{2N}^n \times \mathbb{Z}_{2N}$, where

$$\begin{aligned} \bar{b} - \bar{\mathbf{a}} \cdot \mathbf{s} &= \lceil 2Nb \rfloor - \sum_{i=1}^{n} \lceil 2Na_i \rfloor s_i = 2Nb + \xi_0 - \sum_{i=1}^{n} (2Na_i + \xi_i) s_i \\ &= 2N(e + m_{\text{in}}/2) + e_{\mathsf{ACC}}, \end{aligned} \tag{3}$$

and $e_{\mathsf{ACC}} := \xi_0 - \sum_{i=1}^{n} \xi_i s_i$ and $\xi_0, \ldots \xi_n$ are rounding errors that are uniformly distributed over $(-\frac{1}{2}, \frac{1}{2})$. Note that $e_{\mathsf{ACC}} = 0$ when the coefficients $(\mathbf{a}, b)$ are in $\frac{1}{2N}\mathbb{Z}/\mathbb{Z}$; thus, we can ignore $e_{\mathsf{ACC}}$ when we use the default parameter setting in the TFHE library of $N = 1024$.

#### 2.3.3. BlindRotate

At the line 4, ACC is a trivial TLWE ciphertext $(0, X^{\bar{b}} \cdot \mathrm{testv}) \in \mathbb{T}_N[X] \times \mathbb{T}_N[X]$; thus, $\|\mathsf{Error}(\mathsf{ACC}_1)\|_\infty = 0$. After the loop in line 5, from [6] (Theorem 4.6), we have

$$\|\mathsf{Error}(\mathsf{ACC}_n)\|_\infty \leq 2n(k+1)lN\beta\alpha_{\mathsf{BK}} + n(1 + kN)\epsilon, \tag{4}$$

where $\beta = B_g/2$ and $\epsilon = 1/2B_g^l$ are the precision parameters of the gadget decomposition, $l \in \mathbb{N}$, $B_g \in \mathbb{N}$, and $\alpha_{\mathsf{BK}}$ is a error size ratio of the bootstrapping key BK. After the loop in

line 5, the message of ACC becomes a polynomial $X^{\bar{b}-\bar{\mathbf{a}}\mathbf{s}} \cdot \mathsf{testv}$. From (3) and $e_{\mathsf{ACC}} = 0$, we have $\psi((\bar{\mathbf{a}}, \bar{b})) := \bar{b} - \bar{\mathbf{a}} \cdot \mathbf{s} = 2N(e + m_{\mathrm{in}}/2)$. Notice that

$$X^{\bar{b}-\bar{\mathbf{a}}\cdot\mathbf{s}} \cdot \mathsf{testv} = X^{\bar{b}-\bar{\mathbf{a}}\cdot\mathbf{s}+\frac{N}{2}} \cdot (1 + X^{-1} + \cdots + X^{-(N-1)}) \cdot \mu',$$

where $\bar{b} - \bar{\mathbf{a}} \cdot \mathbf{s} + \frac{N}{2} = 2Ne + \frac{N}{2} + Nm_{\mathrm{in}}$, and from (2), we obtain $0 < 2Ne + \frac{N}{2} < N$. Therefore, we have the following facts:

$$\bar{b} - \bar{\mathbf{a}} \cdot \mathbf{s} + \frac{N}{2} \in \begin{cases} [0, N) & (m_{\mathrm{in}} = 0), \\ [N, 2N) & (m_{\mathrm{in}} = 1), \end{cases}$$

$$\text{the constant term of } X^{\bar{b}-\bar{\mathbf{a}}\cdot\mathbf{s}} \cdot \mathsf{testv} = \begin{cases} \mu' & (m_{\mathrm{in}} = 0), \\ -\mu' & (m_{\mathrm{in}} = 1). \end{cases}$$

Note that, here, we used the fact that $X^{-i} \equiv -X^{N+i}$, which follows the definition $X^N + 1 \equiv 0$.

#### 2.3.4. Extract

The output of SampleExtract in line 6 is simply the coefficients vector of the TLWE sample over the torus $(\mathbf{a}', b') := (\mathsf{coefs}(a''(X)), b_0'') \in \mathbb{T}^N \times \mathbb{T}$, where $\mathsf{coefs}(a''(X))$ is a coefficient vector of $a'' \in \mathbb{T}_N(X)$ and $b_0'' \in \mathbb{T}$ is a constant term of the polynomial $b'' \in \mathbb{T}_N(X)$. The message of the extracted sample $\mathsf{msg}((\mathbf{a}', b'))$ is the constant term of the polynomial $(X^{\bar{b}-\bar{\mathbf{a}}\cdot\mathbf{s}} \cdot \mathsf{testv})$. The message is $\mu'$ if $m_{\mathrm{in}} = 0$ and $-\mu'$ if $m_{\mathrm{in}} = 1$. Therefore,

$$\mathsf{msg}(\mathbf{u}) = \mu' + \mathsf{msg}(\mathsf{SampleExtract}(\mathsf{ACC}) = \begin{cases} 2\mu' (= \mu) & (m_{\mathrm{in}} = 0), \\ 0 & (m_{\mathrm{in}} = 1); \end{cases}$$

i.e., $\mathsf{msg}(\mathbf{u}) = \mu \cdot (1 - m_{\mathrm{in}})$. The size of the error of $(\mathbf{a}', b')$ remains $\|\mathsf{Error}(\mathsf{ACC})\|_\infty$ since SampleExtract adds no extra noise.

#### 2.3.5. KeySwitch

The output of KeySwitchin line 7 is a TLWE sample $(\mathbf{a}, b) \in \mathbb{T}^n \times \mathbb{T}$ of a message $\frac{m_{\mathrm{out}}}{2} \in \mathbb{T}$, with the secret key $\mathbf{s}$. We write this TLWE ciphertext as $\mathsf{LWE}_{\mathbf{s}}(m_{\mathrm{out}})$ for simplicity of the notation. The KeySwitchprocedure is the same as that in [6]. We refer to KeySwitch in Algorithm 2 for completeness of this paper. The keyswitching key is defined as $\mathsf{KS}_{\mathbf{s}' \to \mathbf{s}, \gamma, t}$, where $\mathsf{KS}_{i,j} \in \mathsf{LWE}_{\mathbf{s},\gamma}(s_i' \cdot 2^{-j})$ for $i \in [1, N]$ and $j \in [1, t]$. Here, $\gamma \in \mathbb{R}$ is a parameter that decides the size of the noise of the key switching key, which satisfies $\left\|\mathsf{Error}(\mathsf{KS}_{i,j})\right\|_\infty \leq \gamma$, and $t \in \mathbb{N}$ is a parameter that decides the precision. From [6] (Lemma 4.3), we obtain

$$\begin{aligned} \psi_{\mathbf{s}}(\mathbf{a}, b) &= \psi_{\mathbf{s}}(\mathbf{0}, b') - \sum_{i=1}^{N} \sum_{j=1}^{t} a_{i,j}' \varphi_{\mathbf{s}}(\mathsf{KS}_{i,j}), \\ &= \psi_{\mathbf{s}'}(\mathbf{a}', b') - \sum_{i=1}^{N} \sum_{j=1}^{t} a_{i,j}' \mathsf{Error}(\mathsf{KS}_{i,j}) + \sum_{i=1}^{N} (a_i' - \bar{a}_i') s_i'. \end{aligned}$$

Therefore, we have the bound on the size of the noise as follows:

$$\begin{aligned} \|\mathsf{Error}(\mathbf{a}, b)\|_\infty &\leq \|\mathsf{Error}(\mathsf{ACC})\|_\infty + Nt\gamma + N2^{-(t+1)}, \\ &\leq 2n(k+1)lN\beta\alpha_{\mathsf{BK}} + n(1+kN)\epsilon + Nt\gamma + N2^{-(t+1)}, \end{aligned}$$

where the second inequality follows from (4). The bootstrapping outputs a "fresh" LWE ciphertext when the parameters satisfy the following:

$$2n(k+1)lN\beta\alpha_{\mathsf{BK}} + n(1+kN)\epsilon + Nt\gamma + N2^{-(t+1)} < 1/4. \tag{5}$$

**Algorithm 2:** KeySwitch.

**Input:** An LWE ciphertext $(\mathbf{a}' = (a'_1, \ldots, a'_N), b') \in \mathsf{LWE}_{\mathbf{s}'}(\mu)$, a keyswitching key $\mathsf{KS}_{\mathbf{s}' \to \mathbf{s}}$ where $\mathbf{s}' \in \{0,1\}^N$, $\mathbf{s} \in \{0,1\}^n$ and $t \in \mathbb{N}$ a precision parameter.
**Output:** An LWE sample $\mathsf{LWE}_{\mathbf{s}}(\mu)$

```
for i = 1 to N do
    Let ā'_i be the nearest multiple of 1/2^t to a'_i , thus |ā'_i − a'_i| < 2^{−(t+1)}
    Perform binary decomposition each ā'_i = Σ_{j=1}^{t} a'_{i,j} · 2^{−j}, where a'_{i,j} ∈ {0,1}
end
return (a, b) = (0, b') − Σ_{i=1}^{N} Σ_{j=1}^{t} a'_{i,j} · KS_{i,j}
```

This upper bound follows from (2).

### *2.4. Integer-Wise* LWE *Encryption*

We use a integer-extended variant of Regev's encryption following the scheme proposed by Bourse et al. in [14]. Let $\tau \in \mathbb{N}$, the message $m \in \{-\tau, \ldots, \tau - 1\}$. The integer-wise LWE encryption works as follows:

- $\mathsf{Setup}(\lambda)$: On a input security parameter $\lambda$, $n = n(\lambda)$ and $\sigma = \sigma(\lambda)$ are fixed; a secret key $\mathbf{s} \xleftarrow{\$} \mathbb{B}^n$ is output.
- $\mathsf{Enc}(\mathbf{s}, m)$: A uniformly random vector $\mathbf{a} \xleftarrow{\$} \mathbb{T}^n$ and a noise $e \leftarrow \mathcal{D}_{\mathbb{T}_N[X],\sigma}$ are sampled, where $\mathcal{D}_{\mathbb{T}_N[X],\sigma}$ is a Gaussian distribution over $\mathbb{T}_N[X]$ with a standard deviation $\sigma$. A ciphertext $(\mathbf{a}, b)$ is output, where

$$b = \mathbf{a} \cdot \mathbf{s} + e + \frac{m}{2\tau}. \tag{6}$$

- $\mathsf{Dec}(\mathbf{s}, (\mathbf{a}, b))$: The decryption algorithm returns $\lceil (b - \mathbf{a} \cdot \mathbf{s}) \cdot 2\tau \rfloor$. The decryption works correctly if the size noise $|e|$ is small enough, i.e.,

$$|e| < 1/4\tau, \tag{7}$$

Since $(b - \mathbf{a} \cdot \mathbf{s}) \cdot 2\tau = \left(e + \frac{m}{2\tau}\right) \cdot 2\tau = m + 2\tau e$ and $|2\tau e| < \frac{1}{2}$ holds from (7).

## 3. Integer-Wise General Functional Bootstrapping

We modify TFHE to a variant that can encrypt the plaintext of an integer and perform any 1-variable function in only one bootstrapping. We highlight this section as follows:

- We present our general bootstrapping in Algorithm 3 of Section 3.1, which is built upon the integer-wise LWE encryption Section 2.4.
- Our key technique to construct the general bootstrapping is generalizing the setting of the coefficients of the test vector, which is used in the BlindRotate part of the bootstrapping procedure, in Algorithm 4.
- We also discuss the possible security issue that arose after our modification in Section 3.2.

### *3.1. General Functional Bootstrapping*

We present our bootstrapping scheme in Algorithm 3. Our bootstrapping is different from that of Chillotti et al.'s Algorithm 1 in the setting of the test vector used in the BlindRotatepart. Our bootstrapping procedure is constructed over the integer-wise LWE encryption described in Section 2.4, where homomorphic evaluation is performed in an integer-wise manner. We summarize in Table 1 how the input ciphertext is converted line by line for clarity.

**Algorithm 3:** General functional bootstrapping.

**Input:** A ciphertext $C_{m_{\text{in}}} :=$ An LWE sample $(\mathbf{a}, b) \in \mathsf{LWE}_{\mathbf{s}}(m_{\text{in}})$, such that its plaintext $m_{\text{in}} \in \{-\tau, \ldots, \tau - 1\}$, a bootstrapping key $\mathsf{BK}_{\mathbf{s}\to\mathbf{s}'',\alpha}$, a keyswitching key $\mathsf{KS}_{\mathbf{s}'\to\mathbf{s},\gamma}$, where $\mathbf{s}' = \mathsf{KeyExtract}(\mathbf{s}'')$, a constant function $f : \{0, \ldots, \tau - 1\} \to \{-\tau, \ldots, \tau - 1\}$, and a set of coefficients $\{\mu_0, \ldots, \mu_{N-1}\}$ of the test vector that corresponds to the function $f$.

**Output:** An LWE sample $\mathsf{LWE}_{\mathbf{s}}(m_{\text{out}}) := \mathsf{Bootstrap}(C_{m_{\text{in}}}, f)$, where

$$m_{\text{out}} = f'(m_{\text{in}}) := \begin{cases} f(m_{\text{in}}) & (m_{\text{in}} \in \{0, \ldots, \tau - 1\}), \\ -f(\tau + m_{\text{in}}) & (m_{\text{in}} \in \{-\tau, \ldots, -1\}). \end{cases} \tag{8}$$

1 $\bar{b} := \lceil 2Nb \rfloor, \bar{a_i} := \lceil 2Na_i \rfloor$ for each $i \in [1, n]$

2 Set the test vector as

$$\text{testv} := \mu_0 + \mu_1 X^{-1} + \cdots + \mu_{N-1} X^{-(N-1)} \in \mathbb{T}_N[X]. \tag{9}$$

3 $\mathsf{ACC} \leftarrow X^{\bar{b}} \cdot (0, \text{testv}) \in \mathbb{T}_N[X] \times \mathbb{T}_N[X]$

4 **for** $i = 1$ to $n$ **do** $\mathsf{ACC} \leftarrow [\mathbf{h} + (X^{-\bar{a}_i} - 1) \cdot \mathsf{BK}_i] \boxdot \mathsf{ACC}$

5 $\mathbf{u} := \mathsf{SampleExtract}(\mathsf{ACC})$

6 **return** $\mathsf{KeySwitch}_{\mathsf{KS}}(\mathbf{u})$

**Table 1.** Overview of the bootstrapping scheme.

| | Ctxt | Ptxt | Noise (or Its Bound) |
|---|---|---|---|
| | $(\mathbf{a}, b) \in \mathbb{T}^n \times \mathbb{T}$ | $\frac{m_{\text{in}}}{2\tau}$ | $e \in \mathbb{T}$ |
| $\xrightarrow{\text{Round}}$ | $(\bar{\mathbf{a}}, \bar{b}) \in \mathbb{Z}_{2N}^n \times \mathbb{Z}_{2N}$ | $2N(\frac{m_{\text{in}}}{2\tau})$ | $2Ne$ |
| $\xrightarrow{\text{BlindRotate}}$ | $(a'', b'') \in$ $\mathbb{T}_N[X] \times \mathbb{T}_N[X]$ | $X^{\bar{b} - \bar{\mathbf{a}}\mathbf{s}} \cdot \text{testv}$ | $\lVert \mathsf{Error}(\mathsf{ACC}) \rVert_\infty$ |
| $\xrightarrow{\text{SampleExtract}}$ | $(\mathbf{a}', b') \in \mathbb{T}^N \times \mathbb{T}$ | $\frac{m_{\text{out}}}{2\tau}$ | $\lVert \mathsf{Error}(\mathsf{ACC}) \rVert_\infty$ |
| $\xrightarrow{\text{KeySwitch}}$ | $(\mathbf{a}, b) \in \mathbb{T}^n \times \mathbb{T}$ | $\frac{m_{\text{out}}}{2\tau}$ | $\lVert \mathsf{Error}(\mathsf{ACC}) \rVert_\infty$ $+ N(t\gamma + 2^{-(t+1)})$ |

#### 3.1.1. Input

The input for our bootstrapping scheme is an LWE ciphertext $(\mathbf{a}, b) \in \mathbb{T}^n \times \mathbb{T}$ of an integer message $m_{\text{in}} \in \{-\tau, \ldots, \tau - 1\}$, where $b = \mathbf{a} \cdot \mathbf{s} + e + m_{\text{in}}/2\tau$. The noise $e$ is sampled according to the distribution such that (7) holds to satisfy the condition for correct decryption of the LWE encryption.

#### 3.1.2. Rounding (Line 1)

The output of the rounding is an rounded LWE ciphertext $(\bar{\mathbf{a}}, \bar{b}) \in \mathbb{Z}_{2N}^n \times \mathbb{Z}_{2N}$. Similar to (3), we have

$$\bar{b} - \bar{\mathbf{a}}\mathbf{s} = 2N(e + m_{\text{in}}/2\tau) + e_{\mathsf{ACC}}. \tag{10}$$

Concretely, we obtain $e_{\mathsf{ACC}} = 0$ when $N = 1024$, as we mentioned in Section 2.3.

#### 3.1.3. BlindRotate (Line 3 to Line 4)

At line 3, the ACC is a trivial (noiseless) TLWE sample $(0, X^{\bar{b}} \cdot \text{testv}) \in \mathbb{T}_N[X] \times \mathbb{T}_N[X]$. After the iteration in line 4, the message of ACC becomes a polynomial $X^{\bar{b} - \bar{\mathbf{a}}\mathbf{s}} \cdot \text{testv}$. Since $e_{\mathsf{ACC}} = 0$ and from (10), we have $\psi((\bar{\mathbf{a}}, \bar{b})) := \bar{b} - \bar{\mathbf{a}}\mathbf{s} = 2N\left(e + \frac{m_{\text{in}}}{2\tau}\right)$. Thus, we obtain from (7) that

$$-\frac{N}{2\tau} < 2Ne < \frac{N}{2\tau}. \tag{11}$$

Note that we define the test vector in general form $\text{testv} := \mu_0 + \mu_1 X^{-1} + \cdots + \mu_{N-1} X^{-(N-1)} \in \mathbb{T}_N[X]$, which is the core modification of our scheme. In the following, we

explain how to set the value of the testv coefficients for a homomorphic evaluation of the arbitrary input function $f$ in one bootstrapping. We summarize the setting in Algorithm 4 and show its illustration in Figure 1.

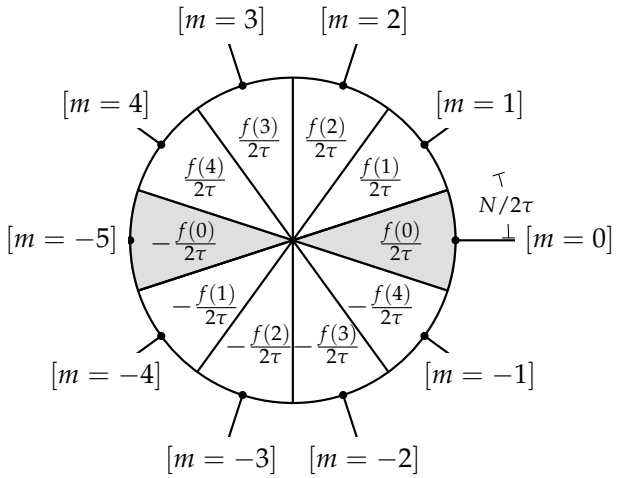


**Figure 1.** Illustration of the "slices" for the integer-wise LWEencryption and bootstrapping procedure when $\tau = 5$.

- When $m_{\text{in}} = 0$, we have $\psi((\bar{\mathbf{a}}, \bar{b})) = 2Ne$ and $(\bar{\mathbf{a}}, \bar{b}) \in \{-\lfloor \frac{N}{2\tau} \rfloor, \ldots, \lfloor \frac{N}{2\tau} \rfloor\}$. Thus, the constant term of the polynomial

$$\begin{aligned} X^{\psi((\bar{\mathbf{a}},\bar{b}))} \cdot \mathsf{testv} &= X^{\psi((\bar{\mathbf{a}},\bar{b}))} \cdot (\mu_0 + \mu_1 X^{-1} + \cdots + \mu_{N-1} X^{-(N-1)}) \\ &= X^{\psi((\bar{\mathbf{a}},\bar{b}))} \cdot (\cdots - \mu_{N-1} X^1 + \mu_0 + \mu_1 X^{-1} \ldots) \end{aligned}$$

  is in

$$\mathcal{M}_0 := \{-\mu_{N-\lfloor \frac{N}{2\tau} \rfloor}, \ldots, -\mu_{N-1}, \mu_0, \mu_1, \ldots, \mu_{\lfloor \frac{N}{2\tau} \rfloor}\}.$$

  We define the value of all elements $\mu \in \mathcal{M}_0$ as $\mu := \frac{f(0)}{2\tau} \in \mathbb{T}$. Note that, since $X^N + 1 \equiv 0$, we now have $X^{-i} \equiv -X^{N-i}$ or, equivalently, $X^i \equiv -X^{-(N-i)}$. Similar to this case, when $m_{\text{in}} = -\tau$, the constant term of the polynomial $X^{\psi((\bar{\mathbf{a}},\bar{b}))} \cdot \mathsf{testv}$ is in $\{\mu_{N-\lfloor \frac{N}{2\tau} \rfloor}, \ldots, \mu_{N-1}, -\mu_0, -\mu_1, \ldots, -\mu_{\lfloor \frac{N}{2\tau} \rfloor}\} := \mathcal{M}_{-\tau}$. The elements of this set are the sign inversions of the elements of the set $\mathcal{M}_0$. Therefore, $\mu \in \mathcal{M}_{-\tau}$ are already defined as $\mu := \frac{-f(0)}{2\tau} \in \mathbb{T}$.
- When $m_{\text{in}} \in \{1, \ldots, \tau - 1\}$, we have $\psi((\bar{\mathbf{a}}, \bar{b})) = 2N\left(e + \frac{m_{\text{in}}}{2\tau}\right)$. From (11), we have $\frac{N}{2\tau} < \left(m_{\text{in}} - \frac{1}{2}\right)\frac{N}{\tau} < \psi((\bar{\mathbf{a}}, \bar{b})) < \left(m_{\text{in}} + \frac{1}{2}\right)\frac{N}{\tau} < N$. Thus, we obtain $\psi((\bar{\mathbf{a}}, \bar{b})) \in \left\{\left\lceil \left(m_{\text{in}} - \frac{1}{2}\right)\frac{N}{\tau} \right\rceil, \ldots, \left\lfloor \left(m_{\text{in}} + \frac{1}{2}\right)\frac{N}{\tau} \right\rfloor\right\}$, and the constant term of the polynomial $X^{\psi((\bar{\mathbf{a}},\bar{b}))} \cdot \mathsf{testv}$ is in

$$\mathcal{M}_{m_{\text{in}}} := \{\mu_{\lceil (m_{\text{in}} - \frac{1}{2})\frac{N}{\tau} \rceil}, \ldots, \mu_{\lfloor (m_{\text{in}} + \frac{1}{2})\frac{N}{\tau} \rfloor}\}.$$

  We define the value of all $\mu \in \mathcal{M}_{m_{\text{in}}}$ as $\mu := \frac{f(m_{\text{in}})}{2\tau} \in \mathbb{T}$. Similarly, when $m_{\text{in}} \in \{-(\tau - 1), \ldots, -1\}$, we have $-N < \left(m_{\text{in}} - \frac{1}{2}\right)\frac{N}{\tau} < \psi((\bar{\mathbf{a}}, \bar{b})) < \left(m_{\text{in}} + \frac{1}{2}\right)\frac{N}{\tau} < -\frac{N}{2\tau}$. Thus, $\psi((\bar{\mathbf{a}}, \bar{b})) \in \left\{\left\lceil \left(m_{\text{in}} - \frac{1}{2}\right)\frac{N}{\tau} \right\rceil, \ldots, \left\lfloor \left(m_{\text{in}} + \frac{1}{2}\right)\frac{N}{\tau} \right\rfloor\right\}$, and the constant term of the polynomial $X^{\psi((\bar{\mathbf{a}},\bar{b}))} \cdot \mathsf{testv}$ is in $\{-\mu_{N+\lceil (m_{\text{in}} - \frac{1}{2})\frac{N}{\tau} \rceil}, \ldots, -\mu_{N+\lfloor (m_{\text{in}} + \frac{1}{2})\frac{N}{\tau} \rfloor}\}$. These values are the sign inversions of the previously defined $\mu \in \mathcal{M}_{m_{\text{in}}}$ for $m_{\text{in}} \in \{1, \ldots, \tau - 1\}$.

**Algorithm 4:** Our setting of the test vector coefficients defined in (9).

**Input:** A constant function $f : \{0, \ldots, \tau - 1\} \to \{-\tau, \ldots, \tau - 1\}$
**Output:** The coefficients $\{\mu_0, \mu_1, \ldots, \mu_{N-1}\}$ of the test vector
1 For all $\mu \in \{-\mu_{N-\lfloor \frac{N}{2\tau} \rfloor}, \ldots, -\mu_{N-1}, \mu_0, \mu_1, \ldots, \mu_{\lfloor \frac{N}{2\tau} \rfloor}\}$, define $\mu := \frac{f(0)}{2\tau} \in \mathbb{T}$.
2 **for** $m_{\text{in}} \in \{1, \ldots, \tau - 1\}$ **do**
3 For all $\mu \in \{\mu_{\lceil (m_{\text{in}} - \frac{1}{2}) \frac{N}{\tau} \rceil}, \ldots, \mu_{\lfloor (m_{\text{in}} + \frac{1}{2}) \frac{N}{\tau} \rfloor}\}$, define $\mu := \frac{f(m_{\text{in}})}{2\tau} \in \mathbb{T}$.
4 **end**

#### 3.1.4. Extract (Line 5)

The message of the output $\mathbf{u} := \mathsf{SampleExtract}(\mathsf{ACC})$ in line 5, denoted as $\mathsf{msg}((\mathbf{a}', b'))$, is the constant term of $(X^{\bar{b} - \bar{\mathbf{a}}\mathbf{s}} \cdot \mathsf{testv})$. By our construction of BlindRotate, it is the coefficient $\mu = \frac{m_{\text{out}}}{2\tau} = \frac{f'(m_{\text{in}})}{2\tau} \in \mathbb{T}$, where $f'$ is defined in (8).

#### 3.1.5. KeySwitch (Line 7)

Our scheme uses the same KeySwitchprocedure as the original TFHE, which is mentioned in Section 2.3.5. The output $\mathsf{KeySwitch}_{\mathsf{KS}}(\mathbf{u})$ of line 6 is a TLWE sample denoted as $(\mathbf{a}, b) \in \mathbb{T}^n \times \mathbb{T}$. The sample is the encryption of the message $\frac{m_{\text{out}}}{2\tau} \in \mathbb{T}$ with the secret key $\mathbf{s}$. We denote this TLWE ciphertext as $\mathsf{LWE}_{\mathbf{s}}(m_{\text{out}})$ for simplicity of the notation. Only the difference from the original bit-wise TFHE is the upper bound of (5): From (7), we need to choose the parameters that satisfy the upper bound

$$2n(k+1)lN\beta\alpha_{\mathsf{BK}} + n(1+kN)\epsilon + Nt\gamma + N2^{-(t+1)} < 1/4\tau, \tag{12}$$

so that we can obtain a "fresh" LWE sample after the bootstrapping with overwhelming probability.

### *3.2. Security*

The security of our scheme largely relies on the original TFHE scheme because we only modify the test vector, and it changes only the bound of the size of noise given in (12). Although the noise bound of TFHE is fixed to $1/4$, as in (5), that of our scheme is $1/4\tau$. Therefore, when we use the larger plaintext space, the bound becomes lower. This means that we need to use the TLWE sample with smaller noise if the bound (12) does not hold. Conversely, we select the security parameters and $\tau$ that satisfy (12); then, the security of our scheme relies solely on the original TFHE scheme. We use the same security parameters and noise as those in the original TFHE scheme and choose a value of $\tau$ that satisfies (12) in our experiments in Section 5. Notably, the bound in (12) is independent of the input function $f$; therefore, our scheme can run arbitrary functions with the same parameters and running time.

## 4. Applications

We present several basic applications of our integer-wise general TFHE bootstrapping procedure (Algorithm 3):

- $C_{\text{sign}(a)} = \mathsf{Sign}(C_a)$: Homomorphic evaluation of the sign function over a ciphertext (Algorithm 5)
- $C_{(a=b)} = \mathsf{Eq}(C_a, C_b)$: Homomorphic evaluation of the equality test (Algorithm 6)
- $C_{(a=b)} = \mathsf{ConstEq}(C_a, b)$: Homomorphic evaluation of the equality test with a plaintext (Algorithm 7)
- $C_{a \cdot b} = \mathsf{MultbyBin}(C_a, C_b)$: Homomorphic evaluation of multiplication by a binary number (Algorithm 8)
- $C_{\lfloor a/d \rfloor} = \mathsf{DivbyConst}(C_a, d)$: Homomorphic evaluation of division by a plaintext (Algorithm 9)

- $C_{\lfloor a/d \rfloor} = \mathsf{Div}(C_a, C_d)$: Homomorphic evaluation of division (Algorithm 10)

The function $\mathsf{Sign}()$, which is the homomorphic algorithm of sign, was already presented by Bourse et al. in [14]. For clarity, we describe $\mathsf{Sign}()$ performed in our scheme to show that our general Bootstrapping schemes includes the sign function. Additionally, the integer-wise TFHE proposed by Bourse et al. [14] can evaluate only $\mathsf{Sign}()$ bootstrapping; it does not support the homomorphic multiplication of integer ciphertexts.

We show Eqand ConstEqin Section 4.2, MultbyBinin Section 4.3, and DivbyConstin Section 4.4. We propose Divin Section 4.5 using DivbyConst, ConstEq, and MultbyBinas building blocks. We also generalize the Divfunction to the homomorphic algorithm that evaluates any 2-variable function in Section 4.5.

**Algorithm 5:** $\mathsf{Sign}(C_{m_{\mathrm{in}}})$: Homomorphic sign evaluation in our bootstrapping scheme.

**Input:** A ciphertext $C_{m_{\mathrm{in}}}$, where $m_{\mathrm{in}} \in \{-\tau, \ldots, \tau-1\}$.

**Output:** $C_{m_{\mathrm{out}}} := \mathsf{Sign}(C_{m_{\mathrm{in}}})$, where $m_{\mathrm{out}} = \begin{cases} -1 & (m_{\mathrm{in}} \in \{-\tau+1, \ldots, -1\}), \\ 0 & (m_{\mathrm{in}} \in \{-\tau, 0\}), \\ 1 & (m_{\mathrm{in}} \in \{1, \ldots, \tau-1\}). \end{cases}$

1 **return** $\mathsf{Bootstrap}(C_{m_{\mathrm{in}}}, f_{\mathrm{sign}})$, where $f_{\mathrm{sign}}(x) := 1$ if $x \in \{1, \ldots, \tau-1\}$, $0$ if $x = 0$.

**Algorithm 6:** $\mathsf{Eq}(C_{m_1}, C_{m_2})$: Equality test.

**Input:** Two ciphertexts $C_{m_1}$ and $C_{m_2}$, where $m_1, m_2 \in \{0, \ldots, \tau-1\}$.

**Output:** $C_{m_{\mathrm{out}}} := \mathsf{Eq}(C_{m_1}, C_{m_2})$, where $m_{\mathrm{out}} = \begin{cases} \tau\ (\texttt{true}) & \text{if } m_1 = m_2, \\ 0\ (\texttt{false}) & \text{otherwise.} \end{cases}$

1 **return** $\mathsf{Bootstrap}(C_{m_1} - C_{m_2}, f_{\mathrm{ztest}})$, where $f_{\mathrm{ztest}}(x) := \tau$ if $x = 0$, $f_{\mathrm{ztest}}(x) := 0$ otherwise.

**Algorithm 7:** $\mathsf{ConstEq}(C_{m_1}, m_2)$: Equality test with a constant

**Input:** A ciphertext $C_{m_1}$ and plaintext $m_2$, where $m_1, m_2 \in \{0, \ldots, \tau-1\}$.

**Output:** $C_{m_{\mathrm{out}}} := \mathsf{ConstEq}(C_{m_1}, m_2)$, where $m_{\mathrm{out}} = \begin{cases} \tau\ (\texttt{true}) & \text{if } m_1 = m_2, \\ 0\ (\texttt{false}) & \text{otherwise.} \end{cases}$

1 Encode $m_2 \in \{0, \ldots, \tau-1\}$ to $\nu_{m_2} := \frac{m_2}{2\tau} \in \mathbb{T}$.

2 **return** $\mathsf{Bootstrap}(C_{m_1} - (\mathbf{0}, \nu_{m_2}), f_{\mathrm{ztest}})$, where $f_{\mathrm{ztest}}(x) := \tau$ if $x = 0$, $f_{\mathrm{ztest}}(x) := 0$ otherwise.

### *4.1. Homomorphic Evaluation of the Sign Function:* $\mathsf{Sign}()$

Algorithm 5 shows that our algorithms homomorphically evaluate the sign function. This algorithm is constructed using Bootstrap with an input function $f_{\mathrm{sign}}$. Specifically, we "express" $f_{\mathrm{sign}}$ by defining the coefficients $\mu_0, \ldots, \mu_{N-1} \in \mathbb{T}$ of test vector using Algorithm 4, as follows:

$$\begin{cases} \mu_0, \ldots, \mu_{\lfloor \frac{N}{2\tau} \rfloor} & := 0, \\ \mu_{\lfloor \frac{N}{2\tau} \rfloor + 1}, \ldots, \mu_{N - \lfloor \frac{N}{2\tau} \rfloor - 1} & := \frac{1}{2\tau}, \\ \mu_{N - \lfloor \frac{N}{2\tau} \rfloor}, \ldots, \mu_{N-1} & := 0. \end{cases}$$

We illustrate the setting of the test vector coefficients in Figure 2. We can confirm from (8) that $m_{\mathrm{out}} = f'(m_{\mathrm{in}}) = f_{\mathrm{sign}}(m_{\mathrm{in}}) = 1$ for $m_{\mathrm{in}} \in \{1, \ldots, \tau-1\}$, $m_{\mathrm{out}} = f'(m_{\mathrm{in}}) = -f_{\mathrm{sign}}(B + m_{\mathrm{in}}) = -1$ for $m_{\mathrm{in}} \in \{-\tau+1, \ldots, -1\}$, $m_{\mathrm{out}} = f'(m_{\mathrm{in}}) = 0$ for $m_{\mathrm{in}} = 0$, and $m_{\mathrm{out}} = f'(m_{\mathrm{in}}) = -f_{\mathrm{sign}}(B + m_{\mathrm{in}}) = 0$ for $m_{\mathrm{in}} = -\tau$. Note that the plaintext of the output with $m_{\mathrm{in}} = -\tau$ is the same as that with $m_{\mathrm{in}} = 0$.

**Algorithm 8:** MultbyBin$(C_{m_{\text{int}}}, C_{m_{\text{bin}}})$: Homomorphic multiplication by binary number

**Input:** Ciphertexts $C_{m_{\text{int}}}, C_{m_{\text{bin}}}$, where $m_{\text{int}} \in \{0, \ldots, \tau-1\}$, $m_{\text{bin}} \in \{\tau\ (\texttt{true}), 0\ (\texttt{false})\}$.

**Output:** $C_{m_{\text{out}}} := \text{MultbyBin}(C_{m_{\text{int}}}, C_{m_{\text{bin}}}) = \begin{cases} C_{m_{\text{int}}} & (m_{\text{bin}} = \tau\ (\texttt{true})), \\ C_0 & (m_{\text{bin}} = 0\ (\texttt{false})). \end{cases}$

1 $\mathsf{C}_{\mathsf{tmp}} = \text{Bootstrap}(C_{m_{\text{int}}} + C_{m_{\text{bin}}} + (\mathbf{0}, \frac{\tau}{2\tau}), f_{\text{id}})$ // $\mathsf{C}_{\mathsf{tmp}} = \begin{cases} C_{m_{\text{int}}} & (m_{\text{bin}} = \tau\ (\texttt{true})), \\ C_{-m_{\text{int}}} & (m_{\text{bin}} = 0\ (\texttt{false})). \end{cases}$

2 **return** $\text{Bootstrap}(C_{m_{\text{int}}} + \mathsf{C}_{\mathsf{tmp}}, f_{\text{half}})$ // $C_{m_{\text{int}}} + \mathsf{C}_{\mathsf{tmp}} = \begin{cases} 2 \cdot C_{m_{\text{int}}} & (m_{\text{bin}} = \tau\ (\texttt{true})), \\ C_0 & (m_{\text{bin}} = 0\ (\texttt{false})). \end{cases}$

**Algorithm 9:** DivbyConst$(C_{m_{\text{in}}}, m_d)$: Homomorphic division by a constant

**Input:** A ciphertext $C_{m_{\text{in}}}$, where $m_{\text{in}} \in \{0, \ldots, \tau-1\}$, and a constant plaintext $m_d \in \{1, \ldots, \tau-1\}$.

**Output:** $C_{m_{\text{out}}} := \text{DivbyConst}(C_{m_{\text{in}}}, m_d) := C_{\lfloor m_{\text{in}}/m_d \rfloor}$

1 **return** $\text{Bootstrap}(C_{m_{\text{in}}}, f_{\text{div},m_d})$

**Algorithm 10:** Div$(C_{m_{\text{in}}}, C_{m_d})$: Homomorphic division.

**Input:** Ciphertexts $C_{m_{\text{in}}}$ and $C_{m_d}$, where $m_{\text{in}} \in \{0, \ldots, \tau-1\}$ and $m_d \in \{1, \ldots, \tau-1\}$.

**Output:** $C_{m_{\text{out}}} := \text{Div}(C_{m_{\text{in}}}, C_{m_d}) := C_{\lfloor m_{\text{in}}/m_d \rfloor}$

1 $\mathsf{C}_{\mathsf{SUM}} := C_0$ (noiseless TLWEsample of 0)
2 **for** $i = 1$ to $(\tau - 1)$ **do**
3 $\quad C_{\lfloor m_{\text{in}}/i \rfloor} = \text{DivbyConst}(C_{m_{\text{in}}}, i)$,
4 $\quad C_{(m_d=i)} = \text{ConstEq}(C_{m_d}, i)$
5 $\quad \mathsf{C}_{\mathsf{SUM}} = \text{Bootstrap}(\mathsf{C}_{\mathsf{SUM}} + \text{MultbyBin}(C_{\lfloor m_{\text{in}}/i \rfloor}, C_{(m_d=i)}), f_{\text{id}})$
$\qquad$ // $\text{MultbyBin}(C_{\lfloor m_{\text{in}}/i \rfloor}, C_{(i=m_d)}) = C_{\lfloor m_{\text{in}}/m_d \rfloor}$ if $i = m_d$, $C_0$ otherwise
6 **end**
7 **return** $\mathsf{C}_{\mathsf{SUM}}$

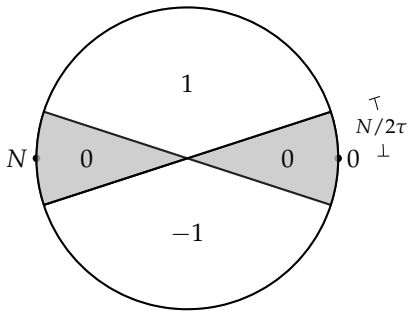


**Figure 2.** Illustration of the slices for the test vector coefficients of sign bootstrapping (when $\tau = 5$).

### *4.2. Homomorphic Equality Test: Eq() and ConstEq()*

For clarity, we present that our bootstrapping scheme includes the homomorphic equality test. This concept is similar to the homomorphic comparison on TFHE proposed by Bourse et al. in [34]. Note that we require the input plaintext space to be in $\{0, \ldots, \tau-1\}$ (or in $\{-\tau, \ldots, -1\}$), for our equality tests Algorithms 6 and 7. This restriction is required since, if we allowed $m_1$ and $m_2$ to be in $\{-\tau, \ldots, \tau-1\}$, we would have $m_2 - m_1 = \tau$ and would obtain the `true`output when $m_2 = \tau + m_1$, although $m_1 \neq m_2$.

Some previous integer-wise homomorphic equality tests were performed based on Fermat's little theorem [35] in other FHE schemes, such as HElib. For example, in HElib

(i.e., in the BGV FHE scheme [31]), the plaintext space was $\mathbb{Z}_p$ for some prime number modulus $p$, and the equality test was constructed using the fact

$$(x-y)^{p-1} \equiv \begin{cases} 1 \bmod p & (x-y \not\equiv 0) \\ 0 \bmod p & (x-y \equiv 0). \end{cases}$$

However, this approach is not efficient because it calls for homomorphic multiplication many ($\approx \log(p)$) times, and thus, some additional bootstrapping are required in order to reduce the increase in noise caused by the homomorphic multiplication. Our homomorphic equality test is efficient because it calls only one bootstrapping procedure.

Algorithm 6 shows $\mathsf{Eq}(\cdot,\cdot)$, the equality test between two ciphertexts. The algorithm is based on the integer-wise "zero test" bootstrapping with the function $f_{\text{ztest}}$, where $f_{\text{ztest}}(x) := \tau$ if $x = 0$, 0 otherwise. Concretely, we define the test vector coefficients $\mu_0, \ldots, \mu_{N-1} \in \mathbb{T}$ that correspond to $f_{\text{ztest}}$, based on Algorithm 4, as follows:

$$\begin{cases} \mu_0, \ldots, \mu_{\lfloor \frac{N}{2\tau} \rfloor} & := \frac{\tau}{2\tau} = \frac{1}{2}, \\ \mu_{\lfloor \frac{N}{2\tau} \rfloor + 1}, \ldots, \mu_{N - \lfloor \frac{N}{2\tau} \rfloor - 1} & := 0, \\ \mu_{N - \lfloor \frac{N}{2\tau} \rfloor}, \ldots, \mu_{N-1} & := \frac{-\tau}{2\tau} = -\frac{1}{2}. \end{cases}$$

We show the illustration of the setting of the coefficients of the test vector in Figure 3.

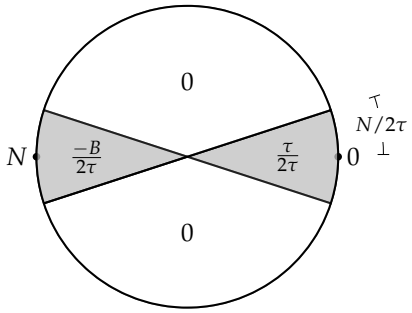


**Figure 3.** Illustration of the slices for the test vector coefficients of equality test bootstrapping (when $\tau = 5$).

Algorithm 7 shows the variant that is performed between a ciphertext and a plaintext. Notice that the second argument of the algorithm is a plaintext $m_1$, which is then encoded as a trivial $(0, \nu_{m_1})$ LWE ciphertext in line 1.

### *4.3. Homomorphic Multiplication by a Binary Number: MultbyBin()*

Algorithm 8 shows the algorithm used to homomorphically evaluate the multiplication by a binary number (MultbyBin). As mentioned before, integer-wise TFHE cannot perform homomorphic multiplication between integer ciphertexts. MultbyBinhomomorphically multiplies an integer ciphertext by a ciphertext of a binary message. MultbyBinis called as a module in the homomorphic division Div, which we describe later in Algorithm 10. Figure 4 illustrates how the MultbyBinworks. Note that we need restrictions for MultbyBin: (1) $\tau$ to be an odd number and (2) $m_{\text{in}} \in \{0, \ldots, \tau - 1\}$. We explained these restrictions later in this subsection.

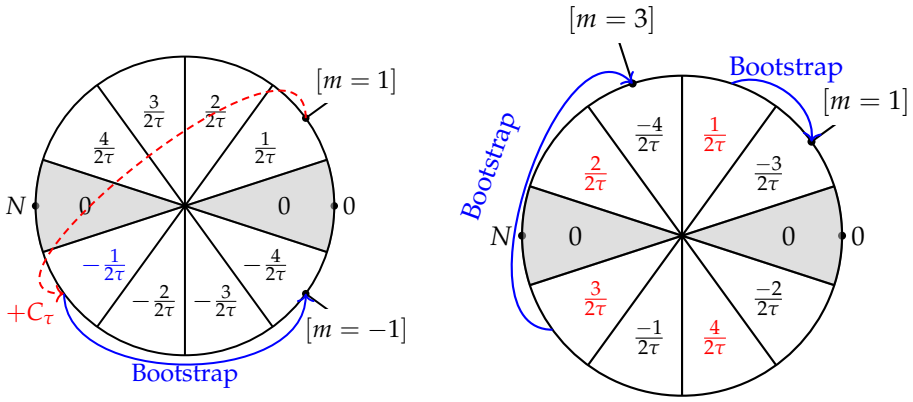


**Figure 4.** Illustration of MultbyBinwhen $(m_{\text{int}}, m_{\text{bin}}) = (1, 0)$. The left figure illustrates the procedure in line 1, and the right figure illustrates the procedure in line 2.

We define two functions $f_{\text{id}}$ and $f_{\text{half}}$ to be used as inputs of Bootstrapto construct MultbyBin. The function $f_{\text{id}} : \{0, \ldots, \tau - 1\} \to \{0, \ldots, \tau - 1\}$ is an identity function defined as $f_{\text{id}}(x) := x$. Based on Algorithm 4, we define the test vector coefficients $\mu_0, \ldots, \mu_{N-1} \in \mathbb{T}$ for $f_{\text{id}}$ as follows:

$$\begin{cases} \mu_0, \ldots, \mu_{\lfloor \frac{N}{2\tau} \rfloor} & := \frac{f_{\text{id}}(0)}{2\tau} = 0, \\ \mu_{\lfloor \frac{(2i-1)N}{2\tau} \rfloor + 1}, \ldots, \mu_{\lfloor \frac{(2i+1)N}{2\tau} \rfloor} & := \frac{f_{\text{id}}(i)}{2\tau} = \frac{i}{2\tau}, \text{ for } i = 1, \cdots, \tau - 1, \\ \mu_{N - \lfloor \frac{N}{2\tau} \rfloor}, \ldots, \mu_{N-1} & := \frac{-f_{\text{id}}(0)}{2\tau} = 0. \end{cases}$$

The function $f_{\text{half}}$ is defined as

$$f_{\text{half}}(x) := \begin{cases} \frac{x}{2} & (x \text{ is even}), \\ -\frac{x+1}{2} - \frac{\tau - 1}{2} & (x \text{ is odd}). \end{cases}$$

For $f_{\text{half}}$, we define the test vector coefficients $\mu_0, \ldots, \mu_{N-1} \in \mathbb{T}$ based on Algorithm 4, as follows:

$$\begin{cases} \mu_0, \ldots, \mu_{\lfloor \frac{N}{2\tau} \rfloor} & := \frac{f_{\text{half}}(0)}{2\tau} = 0, \\ \mu_{\lfloor \frac{(2i-1)N}{2\tau} \rfloor + 1}, \ldots, \mu_{\lfloor \frac{(2i+1)N}{2\tau} \rfloor} & := \frac{f_{\text{half}}(i)}{2\tau}, \text{ for } i = 1, \cdots, \tau - 1, \\ \mu_{N - \lfloor \frac{N}{2\tau} \rfloor}, \ldots, \mu_{N-1} & := \frac{-f_{\text{half}}(0)}{2\tau} = 0. \end{cases}$$

We now explain Algorithm 8. First, in line 1, we homomorphically multiply a ciphertext of a binary message in $\{\tau, 0\}$, $C_{m_{\text{bin}}}$ and a constant $(\mathbf{0}, \frac{\tau}{2\tau})$ to $C_{m_{\text{int}}}$. The output of the bootstrapping of the sum with $f_{\text{id}}$ is stored as $C_{\text{tmp}}$.

- If $C_{m_{\text{bin}}}$ is a ciphertext of 0 (which means `false`), then $C_{m_{\text{bin}}} + (\mathbf{0}, \frac{\tau}{2\tau})$ is $C_\tau$ (ciphertext of the constant $\tau$). Thus, the phase of $C_{m_{\text{int}}}$ is rotated to a position symmetrical about the origin, as illustrated in Figure 4 by the dashed arrow in the left image. Then, the phase of the ciphertext is rotated to a position symmetrical about the $x$-axis. After bootstrapping with $f_{\text{id}}$, $C_{\text{tmp}}$ becomes a ciphertext of $-m_{\text{int}}$.
- If $C_{m_{\text{bin}}}$ is a ciphertext of $\tau$ (which means `true`), then $C_{m_{\text{bin}}} + (\mathbf{0}, \frac{\tau}{2\tau})$ is $C_0$ (ciphertext of 0). Thus, $C_{\text{tmp}}$ remains a ciphertext of $m_{\text{int}}$.

Therefore, in line 2, $C_{m_{\text{int}}} + C_{\text{tmp}} = C_{m_{\text{int}}} + C_{m_{\text{int}}}$ when $m_{\text{bin}} = \tau$, and $C_{m_{\text{int}}} + C_{\text{tmp}} = C_0$ when $m_{\text{bin}} = 0$. After bootstrapping with $f_{\text{half}}$, $C_{m_{\text{int}}} + C_{m_{\text{int}}}$ is converted to $C_{m_{\text{int}}}$, and $C_0$ remains $C_0$, as illustrated in Figure 4. Finally, we obtain $C_{m_{\text{out}}} = C_{m_{\text{int}}}$ when $m_{\text{bin}} = \tau$ and $C_0$ when $m_{\text{bin}} = 0$.

As noted before, we need to restrict $\tau$ to odd values. If $\tau$ is even, when $m_{\text{int}} = \frac{\tau}{2}$ and $m_{\text{bin}} = \tau$, the ciphertext in line 2 $C_{m_{\text{int}}} + C_{\text{tmp}} = C_{m_{\text{int}}} + C_{m_{\text{int}}}$ is converted to $C_0$, but we

require it to remain as $C_{\frac{\tau}{2}}$. Therefore, $\tau$ needs to be an odd number to homomorphically evaluate the division by 2 with the function $\mathsf{Bootstrap}(\cdot, f_{\mathrm{half}})$. Additionally, we require the input $m_{\mathrm{int}}$ to be in $\{0, \ldots, \tau-1\}$ because we cannot perform multiplication with binary values on negative plaintext $m_{\mathrm{int}} \in \{-\tau+1, \ldots, -1\}$ with MultbyBin. (Although we can perform binary multiplication on $m_{\mathrm{int}} = -\tau$ or $\tau$, which is equivalent to $m_{\mathrm{int}} = 0$, we omit $-\tau$ from the input plaintext space, for simplicity.)For $m_{\mathrm{int}} \in \{-\tau+1, \ldots, -1\}$,

- if $m_{\mathrm{bin}} = \tau$ (which means `true`), the message of $C_{\mathsf{tmp}}$ in line 1 is $-(\tau + m_{\mathrm{int}})$, and the output becomes $m_{\mathrm{out}} = 0$ because $C_{\mathsf{tmp}} + C_{m_{\mathrm{bin}}} = C_{\tau}$ for all $m_{\mathrm{int}} \in \{-B+1, \ldots, -1\}$ while $m_{\mathrm{bin}} = \tau(\mathtt{true})$.
- If $m_{\mathrm{bin}} = 0$ (which means `false`), in line 1, the message of $C_{\mathsf{tmp}}$ is $\tau - m_{\mathrm{int}}$ and the output is $m_{\mathrm{out}} = -\tau + m_{\mathrm{int}}$.

Therefore, for negative inputs $m_{\mathrm{int}} \in \{-\tau+1, \ldots, -1\}$, MultbyBin does not work as multiplication with binary values; thus, we need the restriction.

### *4.4. Homomorphic Division by a Constant: DivbyConst*

Algorithm 9 shows that DivbyConsthomomorphically evaluates "division by a constant". DivbyConsttakes a ciphertext $C_{m_{\mathrm{in}}}$ and a plaintext $m_d$ as inputs, and it outputs a ciphertext of $C_{\lfloor m_{\mathrm{in}}/m_d \rfloor}$. Note that we need to restrict $m_{\mathrm{in}} \in \{0, \ldots, \tau-1\}$ and $m_d \in \{1, \ldots, \tau-1\}$.

Bit-wise integer division by 2 on TFHE was proposed in [7]. The algorithm was constructed based on a right shift over the bits; therefore, it only supports a power of two dividend. Homomorphic division by a (not only 2) constant was proposed in [23] based on HElib. This algorithm uses polynomial interpolation, which needs to homomorphically calculate all powers of the ciphertexts $\{C_a, C_a^2, C_a^3, \ldots, C_a^{p-1}\}$, where $p$ is the modulus of the plaintext space $\mathbb{Z}_p$. Thus, it is not efficient since many $(p-1)$ homomorphic multiplications are needed for a homomorphic polynomial interpolation. Contrary to that, our homomorphic division by a constant on the integer-wise TFHE is efficient because it needs only one bootstrapping.

The function $f_{\mathrm{div},d} : \{0, \ldots, \tau-1\} \to \{0, \ldots, \tau-1\}$ is defined as $f_{\mathrm{div},d}(x) := \lfloor x/d \rfloor$ for a constant $d \in \{1, \ldots, \tau-1\}$. We define the test vector coefficients for $f_{\mathrm{div},d}$ based on Algorithm 4, as follows:

$$\begin{cases} \mu_0, \ldots, \mu_{\lfloor \frac{N}{2\tau} \rfloor} & := \frac{f_{\mathrm{div},d}(0)}{2\tau} = 0, \\ \mu_{\lfloor \frac{(2i-1)N}{2\tau} \rfloor + 1}, \cdots, \mu_{\lfloor \frac{(2i+1)N}{2\tau} \rfloor} & := \frac{f_{\mathrm{div},d}(i)}{2\tau} = \frac{\lfloor i/d \rfloor}{2\tau}, \text{ for } i = 1, \cdots, \tau-1, \\ \mu_{N-\lfloor \frac{N}{2\tau} \rfloor}, \ldots, \mu_{N-1} & := \frac{-f_{\mathrm{div},d}(0)}{2\tau} = 0. \end{cases} \tag{13}$$

Note that we need to have a set of $\{\mu_0, \ldots, \mu_{N-1}\}$ for all $d \in \{0, \ldots, \tau-1\}$: we need to store $NB$ values in $\mathbb{T}$ before starting the algorithm. Additionally, we need the restriction $m_{\mathrm{in}} \in \{0, \ldots, \tau-1\}$ since we cannot perform divisions for $m_{\mathrm{in}} \in \{-\tau, \ldots, -1\}$, as the output $m_{\mathrm{out}}$ becomes $-\lfloor (\tau - m_{\mathrm{in}})/m_d \rfloor$.

### *4.5. Homomorphic Division: Div()*

As a final application, we construct a homomorphic division Divin Algorithm 10. It calls DivbyConst, ConstEq, and MultbyBinas subalgorithms. As these subalgorithms need the restriction $m_{\mathrm{in}} \in \{0, \ldots, \tau-1\}$, Divalso follows this restriction.

We now explain Div. First, in line 1, we set a $C_{\mathsf{SUM}} = C_0 = (\mathbf{0}, 0)$ as a trivial TLWE ciphertext of a message 0. In the iteration on line 2 and 6, we obtain a ciphertext of $\lceil a/i \rfloor$, denoted as $C_{\lceil a/i \rfloor}$ with DivbyConst, and then, we test if $i = d$ with ConstEq, where $d$ is the plaintext of the input $C_d$. The output is the ciphertext of the Boolean value of the test result $C_{(d=i)}$. Second, in line 5, we have

$$\mathsf{MultbyBin}(C_{\lfloor a/i \rfloor}, C_{(i=d)}) = \begin{cases} C_{\lfloor a/d \rfloor} & (i = d), \\ C_0 & (\text{otherwise}). \end{cases}$$

Therefore, after this loop, we have $C_{\mathsf{SUM}} = C_0 + \cdots + C_{\lfloor a/d \rfloor} + \cdots + C_0 = C_{\lfloor a/d \rfloor}$.

Generalization to a Two-Variable Function: $\mathsf{Func}()$

We now generalize Divto any two-variable function $\mathsf{Func}(C_{m_1}, C_{m_2})$, in Algorithm 11. The algorithm takes the following inputs: two ciphertexts $C_{m_1}$ and $C_{m_2}$; a two-variable function $f(x, y)$; and sets of test vector coefficients $\{\mu_{0,y}, \ldots, \mu_{N-1,y}\}$, which corresponds to a one-variable function $f_y(x) := f(x, y)$ for all fixed $y \in \{0, \ldots, \tau - 1\}$. The output of the algorithm is a ciphertext $C_{f(m_1,m_2)}$. This algorithm generalizes Divby generalizing the DivbyConstto $\mathsf{Bootstrap}(C_{m_{\mathrm{in}}}, f_y)$, which is the homomorphic evaluation of a general one-variable function $f_y(x) := f(x, y)$ over a ciphertext $C_x$.

## 5. Results of Homomorphic Division

In this section, we present the implementation results of Div. We implemented our bootstrapping procedure and Divby modifying the TFHE library [33]. The parameter values are the same as the defaults of the TFHE library:

- The degree of the polynomials in the ring: $N = 1024$.
- The dimensions of the LWE and TLWE: $n = 500$ and $k = 1$.
- Decomposition basis and length of TGSWciphertexts: $B_g = 2^{10}$ and $l = 2$.
- Decomposition basis and length of KeySwitch: $2^l$ and $t = 8$.
- Standard deviation of the noise of the key-switching keys KS: $\sigma_{\mathsf{KS}} = 2.44 \cdot 10^{-5}$.
- Standard deviation of the noise of the bootstrapping keys BK: $\sigma_{\mathsf{BK}} = 7.18 \cdot 10^{-9}$.

These choices of parameters achieve at least 128-bit security [6]. A single bootstrapping procedure takes approximately 10 msec on our PC with a 3.4-GHz Intel Core i5 CPU. We implemented Divfor a 4-bit integer, which is the same target as that of existing work on homomorphic division algorithms [16–18,23]. Thus, we set $B = 17$ to encrypt 4-bit integers.

Table 2 shows our results, including the values taken from the existing works on homomorphic division. We ran Div (Algorithm 10) 1000 times and took the average. Our method is approximately 3.4 times faster than the fastest method ([23]) shown in the table while achieving a higher level of security ($\lambda > 128$) than that of existing methods. For a fair comparison, we also implemented the "non-restorative division algorithm" on the original (bit-wise) TFHE library with the same parameters. The non-restorative division algorithm is a classic bit-wise algorithm for integer division that is used in previous works [16–18]. We can confirm from Table 2 that our method is approximately 2.2 times faster than non-restorative division. In the asymptotic analysis, as shown in the "Complexity" column of the table, our division algorithm and that of [23] take exponential time. Thus, our algorithm and that of [23] are asymptotically slower than other works based on the bit-wise non-restorative division algorithm. The main bottle necks of our algorithm and that in [23] are the exhaustive loop of line 2–6 in Algorithm 10, similar to the part in [23]. However, while the method of [23] needs an $O(2^{2l})$ calculation, our algorithm is quadratically faster: it takes a smaller $O(2^l)$ calculation. This is due to that our bootstrapping being able to evaluate any one-variable function without extra computational costs over the original TFHE, and in contrast, the bootstrapping in [23] needs an $O(2^l)$ calculation for homomorphic evaluation of the general one-variable function.

**Algorithm 11:** $\mathsf{Func}(C_{m_1}, C_{m_2}, f(\cdot,\cdot))$: Homomorphic evaluation of a 2-variable function $f(x,y)$.

**Input:** Ciphertexts $C_{m_1}$ and $C_{m_2}$, where $m_1, m_2 \in \{0,\dots,\tau-1\}$, a 2-variable function $f(x,y) : \{0,\dots,\tau-1\} \times \{0,\dots,\tau-1\} \to \{0,\dots,\tau-1\}$, and sets of test vector coefficients $\{\mu_{0,y},\dots,\mu_{N-1,y}\}$ which corresponds to 1-variable functions $f_y(x) := f(x,y)$, for all $y \in \{0,\dots,\tau-1\}$.

**Output:** $C_{m_{\text{out}}} := \mathsf{Func}(C_{m_1}, C_{m_2}) := C_{f(m_1,m_2)}$

1 $C_{\mathsf{SUM}} := C_0$ (noiseless TLWEsample of 0)

2 **for** $i = 0$ to $(\tau-1)$ **do**

3 $\quad C_{f_i(m_1)} = \mathsf{Bootstrap}(C_{m_{\text{in}}}, f_i)$,

4 $\quad C_{(m_2=i)} = \mathsf{ConstEq}(C_{m_2}, i)$

5 $\quad C_{\mathsf{SUM}} = \mathsf{Bootstrap}(C_{\mathsf{SUM}} + \mathsf{MultbyBin}(C_{f_i(m_1)}, C_{(m_2=i)}), f_{\text{id}})$
// $\mathsf{MultbyBin}(C_{\lfloor m_{\text{in}}/i \rfloor}, C_{(i=m_d)}) = C_{\lfloor m_{\text{in}}/m_d \rfloor}$ `if` $i = m_d$, $C_0$ `otherwise`

6 **end**

7 **return** $C_{\mathsf{SUM}}$

**Table 2.** Results: comparison with existing homomorphic division implementations. We abbreviate the non-restoring division method as NRD. The "Complexity" column shows the asymptotic complexity for a one-bit-length integer input.

| Method | FHE lib. | Type | Bits ($l$) | Time [sec] | Complexity | Security $\lambda$ |
|---|---|---|---|---|---|---|
| [17] | HElib | Bit-wise | 4 | 67.94 | $O(l^2)$ | >128 |
| [18] | HElib | Bit-wise | 4 | 14.63 | $O(l^2)$ | >128 |
| [16] | HElib | Bit-wise | 4 | 7.74 | $O(l^2)$ | >80 |
| [23] | HElib | Integer-wise | 4 | 3.15 | $O(2^{2l})$ | >80 |
| Ours (Div) | TFHE | Integer-wise | 4 | 0.93 | $O(2^l)$ | >128 |
| NRD | TFHE | Bit-wise | 4 | 2.05 | $O(l^2)$ | >128 |

We show in Table 3 the breakdown of the run time of our Div, which was completed in 930 msec in total. The run time for each of MultbyBin, ConstEq, DivbyConst, and $\mathsf{Bootstrap}(\cdot, f_{\text{id}})$ in line 5 of Algorithm 10 are shown in the table. The run times of these subalgorithms account for almost the entire run time. As the single bootstrapping takes approximately 10 msec on our PC, the cost of these functions is dominated by the run time of bootstrapping, i.e., our functions entail minimal additional costs.

**Table 3.** A breakdown of the run time of our Div (Algorithm 10).

| Functions | # of Bootstrap | Time [msec] | # of Calls | Mean [msec] |
|---|---|---|---|---|
| MultbyBin (line 5) | 2 | 346.0 (37.2%) | $\tau - 1 = 16$ | 21.6 |
| ConstEq (line 4) | 1 | 174.8 (18.8%) | $\tau - 1 = 16$ | 10.9 |
| DivbyConst (line 3) | 1 | 175.8 (18.9%) | $\tau - 1 = 16$ | 11.0 |
| $\mathsf{Bootstrap}(\cdot, f_{\text{id}})$ (line 5) | 1 | 173.9 (18.7%) | $\tau - 1 = 16$ | 10.9 |

*Limitations of Correctness*

In the parameter setting of this section, the final standard deviation of the noise after bootstrapping is $\sigma = 0.00961$, as in [6]. The probability of decryption failure, i.e., the probability of the standard deviation of noise after bootstrapping is larger than $1/16$ and is bounded by $\mathrm{erf}(1/16\sqrt{2}\sigma) < 2^{-32}$. The condition suffices for the bit-wise TFHE scheme. However, for our scheme, decryption fails if the size of the noise after bootstrapping is larger than $1/4\tau$, as shown in (12). The probability of decryption error is bounded by $\mathrm{erf}(1/4\tau\sqrt{2}\sigma) < 2^{-4.06}$ for our scheme. This upper bound appears large, but we empirically observed in our experiment that the decryption error rate is approximately 0.08% (4 decryption errors in 5000 bootstrappings). In other words, there is a trade-off between the size of the plaintext space $\tau$ and the decryption error rate. For 5-bit integers,

i.e., when $B = 33$, the upper bound of the decryption failure probability rapidly increases to approximately 30%. Thus, for the parameter setting in this section, our scheme is practical only for $\leq$ 4-bit integer input. Notably, although the decryption failure rate of our scheme increases as $\tau$ increases, the security of the scheme is not affected because $\tau$ is independent of the standard deviation $\sigma$ of the noise [14].

## 6. Conclusions

We proposed a technique for performing arbitrary 1-variable functions with only one bootstrapping procedure using the integer-wise variant of TFHE. The core modification is to generalize the test vector used in the bootstrapping and the concrete setting of the values of the test vector coefficients. Based on this general functional bootstrapping procedure, we extended the functionality of the integer-wise TFHE scheme to construct several basic applications, such as homomorphic equality testing, multiplication by a binary number, and a division algorithm. We implemented our homomorphic division algorithm and showed that it can be performed in less than 1 second, which is approximately 3.4 times faster than the fastest division algorithm. However, as a limitation, our scheme is practical only for $\leq$4-bit integer inputs, when we use the default parameters of TFHE library. In order to use our scheme for larger integer inputs, we need to use a smaller error rate.

Efficient algorithms for basic arithmetic operations are needed to increase the options for optimizing high-level secure computation applications. We believe our bootstrapping scheme can be used to develop a wide variety of homomorphic calculation algorithms.

**Author Contributions:** Conceptualization, H.O.; methodology, H.O.; software, H.O.; validation, H.O.; formal analysis, H.O.; writing—original draft preparation, H.O.; writing—review and editing, H.O. and S.K.; visualization, H.O.; supervision, S.K. and C.C.; All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Acknowledgments:** We thank Benjamin Curtis and Rachel Player for their comments on the preliminary version of this paper [36].

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Gentry, C. A Fully Homomorphic Encryption Scheme. Ph.D. Thesis, Stanford University, Stanford, CA, USA, 2009.
2. Gentry, C. Fully Homomorphic Encryption Using Ideal Lattices. In *STOC 2009*; ACM: New York, NY, USA, 2009; pp. 169–178. [CrossRef]
3. Brakerski, Z.; Vaikuntanathan, V. Efficient Fully Homomorphic Encryption from (Standard) LWE. In *FOCS 2011*; IEEE: Piscataway, NJ, USA, 2011; pp. 97–106. [CrossRef]
4. Brakerski, Z.; Vaikuntanathan, V. Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In *CRYPTO 2011*; Rogaway, P., Ed.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 505–524. [CrossRef]
5. Brakerski, Z.; Vaikuntanathan, V. Lattice-based FHE As Secure As PKE. In *ITCS 2014*; ACM: New York, NY, USA, 2014; pp. 1–12. [CrossRef]
6. Chillotti, I.; Gama, N.; Georgieva, M.; Izabachène, M. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In *ASIACRYPT 2016*; Cheon, J.H., Takagi, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; pp. 3–33. [CrossRef]
7. Chillotti, I.; Gama, N.; Georgieva, M.; Izabachène, M. Faster Packed Homomorphic Operations and Efficient Circuit Bootstrapping for TFHE. In *ASIACRYPT 2017*; Takagi, T., Peyrin, T., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 377–408. [CrossRef]
8. Chillotti, I.; Gama, N.; Georgieva, M.; Izabachène, M. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *J. Cryptol.* **2020**, *33*, 34–91. [CrossRef]
9. Gentry, C.; Halevi, S.; Smart, N.P. Homomorphic Evaluation of the AES Circuit. In *CRYPTO 2012*; Safavi-Naini, R., Canetti, R., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 850–867. [CrossRef]
10. Gentry, C.; Sahai, A.; Waters, B. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *CRYPTO 2013*; Canetti, R.; Garay, J.A., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 75–92. [CrossRef]
11. Smart, N.P.; Vercauteren, F. Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. In *PKC 2010*; Nguyen, P.Q., Pointcheval, D., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 420–443. [CrossRef]

12. Stehlé, D.; Steinfeld, R. Faster Fully Homomorphic Encryption. In *ASIACRYPT 2010*; Abe, M., Ed.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 377–394.
13. van Dijk, M.; Gentry, C.; Halevi, S.; Vaikuntanathan, V. Fully Homomorphic Encryption over the Integers. In *EUROCRYPT 2010*; Gilbert, H., Ed.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 24–43. [CrossRef]
14. Bourse, F.; Minelli, M.; Minihold, M.; Paillier, P. Fast Homomorphic Evaluation of Deep Discretized Neural Networks. In *CRYPTO 2018*; Shacham, H., Boldyreva, A., Eds.; Springer International Publishing: Cham, Switzerland, 2018, pp. 483–512. [CrossRef]
15. Gilad-Bachrach, R.; Dowlin, N.; Laine, K.; Lauter, K.; Naehrig, M.; Wernsing, J. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; Volume 48, pp. 201–210. [CrossRef]
16. Chen, J.; Feng, Y.; Liu, Y.; Wu, W. Faster Binary Arithmetic Operations on Encrypted Integers. In *WCSE 2017*; SCIEI: Hong Kong, China, 2017; pp. 956–960. [CrossRef]
17. Chen, Y.; Gong, G. Integer arithmetic over ciphertext and homomorphic data aggregation. In Proceedings of the IEEE Conference on Communications and Network Security (CNS), Florence, Italy, 28–30 September 2015; pp. 628–632. [CrossRef]
18. Xu, C.; Chen, J.; Wu, W.; Feng, Y. Homomorphically Encrypted Arithmetic Operations Over the Integer Ring. In *ISPEC 2016*; Bao, F., Chen, L., Deng, R.H., Wang, G., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; pp. 167–181. [CrossRef]
19. Çetin, G.S.; Doröz, Y.; Sunar, B.; Savaş, E. Depth Optimized Efficient Homomorphic Sorting. In *LATINCRYPT 2015*; Lauter, K., Rodríguez-Henríquez, F., Eds.; Springer: Berlin/Heidelberg, Germany, 2015; pp. 61–80. [CrossRef]
20. Bost, R.; Popa, R.A.; Tu, S.; Goldwasser, S. Machine Learning Classification over Encrypted Data. In Proceedings of the Network and Distributed System Security (NDSS) Symposium, San Diego, CA, USA, 8–11 February 2015. [CrossRef]
21. Juvekar, C.; Vaikuntanathan, V.; Chandrakasan, A. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *USENIX Security 2018*; USENIX Association: Baltimore, MD, USA, 2018; pp. 1651–1669.
22. Narumanchi, H.; Goyal, D.; Emmadi, N.; Gauravaram, P. Performance Analysis of Sorting of FHE Data: Integer-Wise Comparison vs Bit-Wise Comparison. In *AINA 2017*; IEEE: Piscataway, NJ, USA, 2017; pp. 902–908. [CrossRef]
23. Okada, H.; Cid, C.; Hidano, S.; Kiyomoto, S. Linear Depth Integer-Wise Homomorphic Division. In *WISTP 2018*; Blazy, O., Yeun, C.Y., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 91–106. [CrossRef]
24. Halevi, S.; Shoup, V. HElib—An Implementation of Homomorphic Encryption. 2019. Available online: https://github.com/shaih/HElib/ (accessed on 26 July 2020).
25. Cheon, J.H.; Han, K.; Kim, A.; Kim, M.; Song, Y. Bootstrapping for Approximate Homomorphic Encryption. In *EUROCRYPT 2018*; Nielsen, J.B., Rijmen, V., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 360–384. [CrossRef]
26. New Jersey Institute of Technology. PALISADE. 2019. Available online: https://git.njit.edu/palisade/PALISADE (26 July 2020).
27. Chen, H.; Dai, W.; Kannepalli, S.; Khanpour, H.; Laine, K.; Lauter, K.; Singh, T.; Song, Y.; Tieman, J. Microsoft SEAL: Fast and Easy-to-Use Homomorphic Encryption Library. 2019. Available online: https://www.microsoft.com/en-us/research/project/microsoftseal/ (accessed on 26 July 2020).
28. Ducas, L.; Micciancio, D. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In *EUROCRYPT 2015*; Oswald, E., Fischlin, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2015; pp. 617–640. [CrossRef]
29. Regev, O. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. In *STOC '05*; ACM: New York, NY, USA, 2005, pp. 84–93. [CrossRef]
30. Cheon, J.H.; Stehlé, D. Fully Homomophic Encryption over the Integers Revisited. In *EUROCRYPT 2015*; Oswald, E., Fischlin, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2015; pp. 513–536. [CrossRef]
31. Brakerski, Z.; Gentry, C.; Vaikuntanathan, V. (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In *ITCS 2012*; ACM: New York, NY, USA, 2012; pp. 309–325. [CrossRef]
32. Lyubashevsky, V.; Peikert, C.; Regev, O. On Ideal Lattices and Learning with Errors over Rings. In *EUROCRYPT 2010*; Gilbert, H., Ed.; Springer: Berlin/Heidelberg, Germany, 2010, pp. 1–23. [CrossRef]
33. Carpov, S.; Chillotti, I.; Gama, N.; Georgieva, M.; Izabachene, M. TFHE: Fast Fully Homomorphic Encryption over the Torus. 2019. Available online: https://tfhe.github.io/tfhe/ (accessed on 26 July 2020).
34. Bourse, F.; Sanders, O.; Traoré, J. Improved Secure Integer Comparison via Homomorphic Encryption. In *CT-RSA 2020*; Jarecki, S., Ed.; Springer International Publishing: Cham, Switzerland, 2020; pp. 391–416. [CrossRef]
35. Çetin, G.S.; Doröz, Y.; Sunar, B.; Martin, W.J. Arithmetic Using Word-wise Homomorphic Encryption. Cryptology ePrint Archive, Report 2015/1195. 2015. Available online: https://eprint.iacr.org/2015/1195 (26 July 2020).
36. Okada, H.; Kiyomoto, S.; Cid, C. Integerwise Functional Bootstrapping on TFHE. In *Information Security*; Susilo, W., Deng, R.H., Guo, F., Li, Y., Intan, R., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 107–125. [CrossRef]