IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 2 November 2023, accepted 28 December 2023, date of publication 9 January 2024,
date of current version 16 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3351738

METHODS

Overflow-Detectable Floating-Point Fully Homomorphic Encryption

SEUNGHWAN LEE AND DONG-JOON SHIN, (Senior Member, IEEE)
Department of Electronics and Computer Engineering, Hanyang University, Seoul 04673, South Korea

Corresponding author: Dong-Joon Shin (djshin@hanyang.ac.kr)

This work was supported by Institute of Information and communications Technology Planning and Evaluation (IITP) grant funded by the Korea Government (MSIT) under Grant 2021-0-00400, Development of Highly Efficient PQC Security and Performance Verification for Constrained Devices.

ABSTRACT A floating-point fully homomorphic encryption (FPFHE) is proposed, which is based on torus fully homomorphic encryption equipped with programmable bootstrapping. Specifically, FPFHE for 32-bit and 64-bit floating-point messages are implemented, the latter showing the state-of-the-art precision among FHEs. Also, a ciphertext is constructed for checking if an overflow has occurred or not while evaluating arithmetic circuits with the proposed FPFHE, which is useful when the message space or arithmetic circuit is too complex to estimate a bound of outputs such as some deep learning applications. Also, homomorphic algorithms, which are crucial components of overflow detectable (OD)-FPFHE, are constructed. First, a state-of-the-art bootstrapping method of TFHE is extended to bootstrap larger messages by using NTT-friendly integer modulus. Second, a subgaussian analysis method is proposed without assuming independent heuristic on AP/GINX-bootstrapping even if the deterministic gadget decomposition is used. Third, the blind rotation algorithm of TFHE is modified such that any secret key having finite non-zero values can be used while keeping the number of NTT operations the same as when the binary key is used. Fourth, various homomorphic algorithms are proposed such as evaluating min and max, lifting a constant message to the monomial exponent, counting the number of consecutive zeros from the most significant in the fraction, and performing carryover after homomorphic operation of floating-point numbers. Finally, 32-bit and 64-bit OD-FPFHEs are implemented and simulation results are provided to confirm that they work well even for extreme cases. Also, it is verified that homomorphic overflow detection is well-operated.

INDEX TERMS Fully homomorphic encryption, homomorphic floating-point arithmetic, homomorphic overflow detection, subgaussian error analysis.

I. INTRODUCTION

Since Gentry's seminal work on fully homomorphic encryption (FHE) [2], various FHEs such as BGV/FV [3], FHEW/TFHE [4], [5], and CKKS [6] have been proposed and intensively studied. FHE is a powerful methodology for evaluating arithmetic circuits while keeping the privacy of data. As applications of FHE with boolean circuits, private information retrieval (PIR) [7], [8] and private set intersection (PSI) [9] have been studied. Also, homomorphically evaluating deep learning models [10] have been mostly studied by using CKKS. Since CKKS deals with a ciphertext packed with a large number of messages, CKKS is the most suitable for evaluating circuits with extensive parallel data.

The associate editor coordinating the review of this manuscript and approving it for publication was Sedat Akleylek.

A. MOTIVATION OF FLOATING-POINT FHE AND OVERFLOW-DETECTABLE FHE

However, when floating-point numbers are encoded into the message space of CKKS (Minkowski space [11]), encoding errors cannot be avoided. Therefore, encoding in CKKS inevitably degrades the accuracy of floating-point arithmetic. Without solving such encoding error problems, CKKS may not guarantee acceptable results for the applications requiring

complex and accurate operations such as calculating satellite collision probabilities [12]. Therefore, an accurate floating-point FHE (FPFHE) is required, which can achieve almost similar results to the corresponding plaintext operations.

Furthermore, the efficiency of the floating-point and fixed-point number systems varies for each application. The Minkowski message space is a fixed-point number system which is an effective number system for computations when the range of input data is limited (e.g. when data are aligned between 0 and 1). However, in cases where the size of a user's input data is unknown, the floating-point number system is the efficient choice since it offers a broader range of represented numbers and relative error in the results. In other words, for achieving a general-purpose FHE system, research in floating-point FHE is essential.

Moreover, overflow occurrences are another problem For example, demands of verification and testing validation of streaming data have been increased [13]. Apparently, if the server homomorphically manipulates ciphertexts of data having an unbounded size, the results may contain overflows. As another example, evaluating a deep arithmetic circuit may return irrelevant results when any of the intermediate results take a value out of the message space.

However, in contrast to the evaluation with plaintexts, such overflow cannot be detected by a user in the ciphertext domain. If a range of circuit output is not bounded or an input dimension is too large, overflow frequently occurs and thus any input cannot be ensured whether it causes an overflow or not. Also, when the past data is used to update the circuit, e.g. privacy-preserving federated learning [8], inaccurate updating using meaningless values due to overflow ruins the circuit performance. However, to the best of our knowledge, an overflow detection method for FHE has not been proposed.

B. CONTRIBUTIONS

Our main contributions are divided into two parts. First, we propose FPFHE with homomorphic normalization for the first time, which effectively resolves the error problem from encoding floating-point numbers and makes every operation on ciphertexts with FPFHE synchronized to the corresponding operation on plaintexts from floating-point message space. Moreover, we implement FPFHE with single (32-bit) and double (64-bit) precision, which shows a much wider range of numbers (up to 2^{127} and 2^{1023}, respectively) and exact significant digits. Note that the message space and operations of CKKS only guarantee fixed-point precision.

Second, an effective overflow-detection (OD) method with the proposed FPFHE is constructed, which can check whether an overflow occurs or not during homomorphic operations. By properly combining these two schemes, we construct OD-FPFHE.

Also, we propose homomorphic algorithms to handle the following technical issues, which are important components of OD-FPFHE and can also be used for other FHEs.

- **Sequential bootstrapping on shared primes:** FHE requires bootstrapping for reducing the amplified error from homomorphic operations. We extend the bootstrapping method in [1] to bootstrap a large number of message bits by using integers modulo *shared primes* Q which is number theoretic transform(NTT)-friendly. This algorithm enables the proposed FPFHE to bootstrap more message bits by using NTT, which is a solution for removing errors generated from fast Fourier transform (FFT), listed as an open problem in [1].
- **Modified blind rotation for GINX-bootstrapping:** This algorithm keeps the number of NTT operations the same for any secret key having finite non-zero values and hence improves running time compared to the state-of-the-art GINX-bootstrapping [5].
- **Error analysis without independent heuristic:** This analysis is applicable even when deterministic gadget decomposition is used and makes it possible to choose small lattice parameters for enhancing running time.
- **Various non-linear homomorphic algorithms**: We propose various homomorphic algorithms such as evaluating min and max, lifting a constant message to the monomial exponent, counting the number of consecutive zeros from the most significant in the fraction of floating-point message until non-zero value occurs, and performing carryover after homomorphic operations. Note that they are run by using sequential bootstrapping.

C. RELATED WORKS

To the best of our knowledge, [14] suggests homomorphic operations on an approximated rational number, however, it is not the exact implementation of the floating-point number system. Reference [15] also implements a floating-point FHE. However, since they do not normalize the results after homomorphic operations (See Section II-F), the operation error may grow rapidly after consecutive homomorphic operations. Moreover, they suffer from slow operation time because they only directly add floating-point operations to the existing FHE schemes using gate operations such as TFHE and BGV/FV.

II. PRELIMINARIES

This section introduces mathematical backgrounds and some fully homomorphic encryption schemes. The main reference and notation of algebraic and statistical backgrounds are followed from [11], [16], and [17].

A. NOTATION

Let $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{C}$ denote the sets of natural, integer, rational, and complex numbers, respectively. Let $\mathbb{Z}_q \cong \mathbb{Z}/q\mathbb{Z}$ be an

integer ring $\mathbb{Z}$ modulo $q\mathbb{Z}$ for some $q \in \mathbb{N}$, and let $\mathbb{Z}_q[X]$ be a polynomial ring $\mathbb{Z}[X]$ modulo $q\mathbb{Z}[X]$. We use $[n]$ to denote an index set of $0, 1, \ldots, n-1$ for $n \in \mathbb{N}$. More generally, $[n_1, n_2, \ldots, n_m] = \prod_{i=1}^{m}[n_i]$ is used as a product index set for $n_1, \ldots, n_m \in \mathbb{N}$.

We use boldface $\boldsymbol{a}= (a_i)_{i\in[n]}$ to denote row vector where a_i is the i-th element of $\boldsymbol{a}$. Analogously for any polynomial $a(X) \in \mathbb{Z}[X]$, we use a_i to denote the coefficient of X^i in $a(X)$. For a polynomial vector $\boldsymbol{a}(X)$, $a_i(X)$ denotes the i-th polynomial of $\boldsymbol{a}(X)$ and $a_{i,j}$ is the j-th coefficient of $a_i(X)$.

As a magnitude of an element, we always use l_1 metric $|\cdot|$ for $x \in \mathbb{R}$. For a vector $\mathbf{x}\in \mathbb{R}^n$, $|x|$ is the maximum value of $|x_i|$ for all $i \in [n]$. If $x \in \mathbb{Z}_q$ is given, $|x|$ is defined as $|\bar{x}|$ where $\bar{x} = x \mod q$ with $-q/2 \le \bar{x} < q/2$. Analogously, if $a(X) \in \mathbb{Z}_q[X]$, $|a(X)|$ is defined as the maximum among $|a_i|$ for all i.

We use the following index function θ for mapping 2-D indices of the lower triangle part of a matrix into integers such as

$$\theta : \{(i,j) \in \mathbb{N}^2 | i \ge j\} \to \mathbb{N}, (i,j) \mapsto i(i+1)/2 + j, \quad (1)$$

which is used for indexing tensor product keys. Note that θ^{-1} denotes its inverse function, and θ_1^{-1} and θ_2^{-1} are the coordinate functions of θ^{-1} such that $\theta^{-1}(x) = (\theta_1^{-1}(x), \theta_2^{-1}(x))$. We represent a non-negative rational number x by $(x_n.x_{n-1}\ldots x_0)_{(\beta)}$ for a given $\beta \ge 2$ if

$$x = x_n + x_{n-1}\beta^{-1} + \ldots + x_0\beta^{-n},$$

where $0 \le x_0, \ldots, x_n < \beta$ are non-negative integers and β is called a radix. We use Big Oh and Omega notation as $O(\cdot)$ and $\Omega(\cdot)$, respectively.

B. ALGEBRAIC BACKGROUND

Let $\Phi_{2N}(X) \in \mathbb{Q}[X]$ be the cyclotomic polynomial of order $2N \in \mathbb{N}$. If N is a power of two, then $\Phi_{2N}(X)$ is $X^N + 1$. Let $R_N \triangleq \mathbb{Z}[X]/(X^N + 1)$ be the quotient polynomial ring with ideal $(X^N + 1)$ and let $R_{N,q} \triangleq \mathbb{Z}[X]/(X^N + 1, q) \cong \mathbb{Z}_q[X]/(X^N + 1)$ be the quotient polynomial ring with ideal $(X^N + 1, q)$ for a positive integer $q \in \mathbb{Z}$. It is clear that the product of $a(X), b(X) \in R_{N,q}$ is

$$a(X)b(X) = \sum_{j\in[N]} \left[\sum_{i=0}^{j} a_i b_{j-i} - \sum_{i=j+1}^{N-1} a_i b_{N+j-i} \right] X^j \quad (2)$$

having the property that a_i and b_i for all $i \in [N]$ appear only once in the expression of each coefficient of $a(X)b(X)$.

For the above algebraic structure, there are two types of Chinese remainder theorem(CRT). First, if Q is a prime number satisfying $2N|(Q-1)$, then $\mathbb{Z}_Q$ has a primitive $2N$-th root of unity $\zeta \in \mathbb{Z}_Q$ such that there is an isomorphism by the CRT as follows:

$$\psi : R_{N,Q} \to \mathbb{Z}_Q^N, \quad a(X) \mapsto \left(a(\zeta^1), a(\zeta^3), \ldots, a(\zeta^{2N-1})\right).$$

We call Q as NTT-friendly if $2N|(Q-1)$.

Second, if $Q = Q_0Q_1 \ldots Q_{n-1} \in \mathbb{N}$ with relatively prime integers Q_i, there exist the following canonical isomorphism:

$$\phi : \mathbb{Z}_Q \to \prod_{i\in[n]} \mathbb{Z}_{Q_i}, \ a \mapsto \left(a \bmod Q_i\right)_{i\in[n]}. \quad (3)$$

In this paper, a particular pair of primes are used. Let two NTT-friendly primes Q_0 and Q_1 be called *shared primes* if they share the same scaling factor $\nu \in \mathbb{N}$ such that $Q_0 = \nu 2^{\eta_0} + 1$ and $Q_1 = \nu 2^{\eta_1} + 1$. Note that, the product of *shared primes* is used as an integer modulus Q as in Section III-B.

C. STATISTICAL BACKGROUND

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be an ambient probability space and $X : \Omega \to \mathbb{R}$ be a random variable. When the co-domain of X is defined over $\mathbb{Z}_Q$, we always define X as a function $\Omega \to \{-\lfloor Q/2 \rfloor, \ldots, \lfloor Q/2 \rfloor\}$. A random variable X is called B-bounded if $|X| \le B$ almost surely. Next, we briefly introduce subgaussian random variable and its properties.

Definition 1 (Subgaussian [16]). *A random variable X is called a subgaussian random variable with a standard parameter $\sigma \ge 0$, denoted as $X \sim subG(\sigma)$, if $\mathbb{E}[X] = 0$ and the moment generating function $M_X(t)$ is bounded as follows*:

$$M_X(t) \triangleq \mathbb{E}[\exp(tX)] \le \exp(\sigma^2 t^2/2).$$

Proposition 1 (Error bound [16]). *For any random variable $X \sim subG(\sigma)$ and $v > 0$, X is $O(\sigma\sqrt{v})$-bounded except with $2^{-\Omega(v)}$ probability.*

It is also known that the space of subgaussians forms an $\mathbb{R}$-vector space with properties: (*i*) If $X \sim subG(\sigma_X)$ and $Y \sim subG(\sigma_Y)$, then $X + Y \sim subG(\sigma_X + \sigma_Y)$; (*ii*) For any scaling $c \in \mathbb{R}$, $cX \sim subG(|c|\sigma_X)$; (*iii*) Moreover, if X and Y are independent, $X + Y \sim subG((\sigma_X^2 + \sigma_Y^2)^{1/2})$.

Since (*iii*) is the best analytical result for the sum of two subgaussian random variables in terms of minimizing variance, we will focus on the conditions for $X + Y \sim subG((\sigma_X^2 + \sigma_Y^2)^{1/2})$ without assuming independence. Random variables $X_i \sim subG(\sigma_i)$ for $i \in [n]$ are called to have *Pythagorean additivity* if $\sum_{i\in[n]} X_i \sim subG((\sum_{i\in[n]} \sigma_i^2)^{1/2})$ and the conditions for satisfying *Pythagorean additivity* have been investigated as follows.

Lemma 1 (Sum of dependent subgaussians [18]). *If $(Z_i|Z_0, \ldots, Z_{i-1}) \sim subG(\sigma_i)$, $\mathbb{E}[Z_i] = 0$ for all $i \in [n]$, and σ_i is free of $Z_0, \ldots, Z_{i-1}$, then the random variables Z_i, $i \in [n]$, have Pythagorean additivity.*

Corollary 1. *Let $Y_i \sim subG(B_Y)$ be mutually independent B_Y-bounded random variables for all $i \in [n]$, and let $X_i \sim subG(B_X)$ be B_X-bounded random variables for all $i \in [n]$ where X_i depends only on X_j and Y_j for all $j < i$. Then X_iY_i for all $i \in [n]$ have Pythagorean additivity.*

Corollary 1 ensures that although all X_iY_i's are dependent on each other, they can have *Pythagorean additivity* similar to the case when X_iY_i's are mutually independent. This is useful for analyzing a sum of large numbers of dependent

random variables, e.g. analyzing an error bound after bootstrapping in FHE. However, Corollary 1 implies that X_i and Y_i must be bounded with zero mean since X_i and Y_i are subgaussian. A similar analysis under more relaxed conditions is performed in Section IV-A.

D. LWE/MLWE SYMMETRIC ENCRYPTION AND GADGET DECOMPOSITION

In this section, widely used lattice-based crytosystems are introduced. First, we recall LWE symmetric encryption [19]. For given $q, n, t \in \mathbb{N}$, t-bit message space $\mathbb{Z}_{2^t}$, a scaling factor $1 \le \Delta \le \lfloor q/2^t \rceil$, and a secret key $\boldsymbol{s} = (-s_1, \ldots, -s_n, 1) \in \mathbb{Z}_q^{n+1}$, a ciphertext for the message $m \in \mathbb{Z}_{2^t}$ is obtained as follows:

$$\mathbf{ct}_s\big[\Delta m\big] \triangleq \big(a, b = \sum_{i=1}^{n} a_i s_i + \Delta m + e\big)^T \in \mathbb{Z}_q^{(n+1)\times 1}, \quad (4)$$

where m is chosen from $\{0, \ldots, 2^t - 1\}$, T denotes the transpose of matrix, $\boldsymbol{a} \in \mathbb{Z}_q^n$ is chosen uniformly at random, e is sampled from a centered discrete Gaussian distribution χ_e on $\mathbb{Z}$ with standard deviation σ, and $\mathbf{s}$ is sampled from a distribution χ_s. We adopt a ternary secret key $\boldsymbol{s}$, which is widely used for FHE [20]. In addition, $\boldsymbol{s}$ is called h-sparse if the number of non-zero elements of $\boldsymbol{s}$ is h.

The phase function φ_s and the decryption function $\bar{\varphi}_s$ of $\mathbf{ct}_s$ are defined as:

$$\varphi_s\Big(\mathbf{ct}_s\big[\Delta m\big]\Big) \triangleq b - \sum_{i=1}^{n} a_i s_i = \Delta m + e,$$
$$\bar{\varphi}_s\Big(\mathbf{ct}_s[\Delta m]\Big) \triangleq \Big\lfloor \big(\varphi_s(\mathbf{ct}_s[\Delta m]) + \lfloor \Delta/2 \rfloor\big)/\Delta \Big\rfloor.$$

Therefore, if $|e| < \Delta/2$, then the message m is correctly extracted by the decryption function $\bar{\varphi}_s$ and we call such $\mathbf{ct}_s[\Delta m]$ a *valid* ciphertext.

If the structure $\mathbb{Z}_q^{(n+1)\times 1}$ of LWE ciphertext in (4) is replaced by $R_{N,q}$, such ciphertext is called a module-LWE (MLWE) ciphertext $\mathrm{CT}_{s(X)}[\Delta m(X)]$ for the message $m(X) \in R_{N,q}$ encrypted by the secret key $\boldsymbol{s}(X) \in R_{N,q}^K$. In addition, we use a generalized MLWE of sample extraction [1], [4], which generates LWE ciphertexts $\mathbf{ct}[\Delta m_i]$ from MLWE ciphertext $\mathrm{CT}[\Delta m(X)]$ as follows:

$$\begin{aligned}&\mathbf{SampleExtract}(\mathbf{CT}_{s(X)}[\Delta m(X)], i) = \mathbf{ct}_{s'}[\Delta m_i]\\ &\triangleq (a'_{0,0}, \ldots, a'_{0,N-1}, a'_{1,0}, \ldots, a'_{K-1,N-1}, b_i), \end{aligned} \quad (5)$$

where $a'_{k,j} = a_{k,i-j}$ for $0 \le j \le i$ and $a'_{k,j} = -a_{i,N-j}$ for $i < j \le N-1$, for all $k \in [K]$. Then, $\mathbf{ct}_{s'}[\Delta m_i]$ is a *valid* LWE ciphertext for the secret key $\boldsymbol{s}' = (s_{0,0}, \ldots, s_{0,N-1}, s_{1,0}, \ldots, s_{K-1,N-1})$. In this paper, LWE and MLWE ciphertexts are denoted as $\mathbf{ct}$ and CT if they are clear in the context.

For a given N, a ciphertext $\mathbf{ct}$ is called *squashed* if the integer modulus used for $\mathbf{ct}$ is $2N$. A *squashed* $\mathbf{ct}$ is used to explain FHEW/TFHE in Section II-E.

1) GADGET DECOMPOSITION

Next, an approximated gadget for decomposing LWE and MLWE ciphertext is introduced, which is used for constructing GSW cryptosystem.

Definition 2 (Gadget [21]). *For any finite additive group $\mathfrak{R}$, an $\mathfrak{R}$-gadget of size l, quality ρ, and precision ϵ is a vector $\boldsymbol{g} \in \mathfrak{R}^l$ such that any element $u \in \mathfrak{R}$ can be written as an approximated integer combination $\sum_i g_i \cdot x_i$ such that $\max|x_i| \le \rho$ for all $x_i \in \mathbb{Z}$ and the gadget error $A = u - \sum_i g_i \cdot x_i \in \mathfrak{R}$ has the magnitude $|A| \le \epsilon$.*

Proposition 2 (Deterministic (signed) gadget decomposition [4], [5]). *Assume that two finite additive groups $R_{N,Q}$ and $\mathbb{Z}_Q$, and gadget parameters $B \in \mathbb{N}$ and $\bar{l} \in \mathbb{N}$ are given such that $B^{\bar{l}-2} \le Q < B^{\bar{l}-1}$. Then, there exists a gadget $\boldsymbol{g} = (B^{\bar{l}-l}, \ldots, B^{\bar{l}-1})$ and the deterministic gadget decomposition $\mathrm{G}_1^{-1} : R_{N,Q} \to R_{N,Q}^{1\times l}$ and $\mathrm{G}_2^{-1} : \mathbb{Z}_Q \to \mathbb{Z}_Q^{1\times l}$ with size $l = \bar{l} - 1$, quality $\rho = \lceil B/2 \rceil$ and precision $\epsilon = \lceil \sum_{i=1}^{\bar{l}-l} B^i/2 \rceil$.*

Note that if $\epsilon = 0$, i.e., $l = \bar{l}$, it is an exact gadget decomposition and if $l < \bar{l}$, it is an approximated gadget decomposition. Next, module GSW (MGSW) cryptosystem [22] and external product of MGSW and MLWE ciphertexts are introduced based on gadget decomposition.

E. FULLY HOMOMORPHIC ENCRYPTIONS USED FOR CONSTRUCTING OD-FPFHE: (M)GSW, FHEW, AND TFHE

First, (M)GSW cryptosystem is briefly introduced. Let $I_n \in R_{N,Q}^{n\times n}$ be the identity matrix and $\otimes$ denote Kronecker product. Then for a message $m \in \{0, 1\}$, MLWE secret key $\boldsymbol{s}(X)$, and $R_{N,Q}$-gadget $\boldsymbol{g}$ in Proposition 2, an MGSW ciphertext $\mathrm{GCT}_{s(X)}[m] \in R_{N,Q}^{l(K+1)\times(K+1)}$ in matrix form is obtained by using MLWE ciphertexts as follows [5], [22]:

$$\mathrm{GCT}_{s(X)}[m] \triangleq \Big(\mathrm{CT}_{s(X)}\Big[m\boldsymbol{S}'_i(X)\Big]\Big)_{i\in[l(K+1)]}^T$$

where $\boldsymbol{S}'(X) = (I_{K+1} \otimes g)\boldsymbol{s}(X)$. The external product $\boxdot$ for any given two ciphertexts $\mathrm{CT}_{s(X)}[m_1(X)]$ and $\mathrm{GCT}_{s(X)}[m_2]$ is defined as:

$$\boxdot : \mathrm{CT} \times \mathrm{GCT} \to \mathrm{CT}, \quad (\mathrm{CT}, \mathrm{GCT}) \mapsto \mathrm{G}^{-1}(\mathrm{CT})\mathrm{GCT}, \quad (6)$$

where G^{-1} is a gadget decomposition algorithm given in Proposition 2. The phase result of CT $\boxdot$ GCT is known as follows [4]:

$$\begin{aligned}&\varphi_{s(X)}(\mathrm{CT}_{s(X)} \boxdot \mathrm{GCT}_{s(X)})\\ &= m_2 m_1(X) + m_2 e'(X) + \sum_{i\in[l(K+1)]} e_i(X)\mathrm{G}^{-1}(\mathrm{CT})_i\\ &\quad + \sum_{i\in[K+1]} s_i(X)A_i(X) \end{aligned} \quad (7)$$

where $\boldsymbol{e}(X) \in R_{N,Q}^{l(K+1)}$ is the noise in GCT, $e'(X)$ is the noise in CT, and $\boldsymbol{A}(X)$ is the gadget error from Definition 2.

Since $|e_i(X)|$ and $|s_i(X)|$ are small, the validity of CT $\boxdot$ GCT depends on ρ and ϵ of gadget decomposition.

For a given LWE ciphertext **ct** and an evaluating function f, bootstrapping in FHEW and TFHE performs the following operations: (*i*) the modulus of **ct** is reduced to $2N$ so that the *valid squashed* $\mathbf{ct}'[\Delta m]$ for $m = (m_{t-1}, \ldots, m_0)_{(2)}$ is obtained [5]; (*ii*) **BlindRotate** [4] is run with the accumulate polynomial $\mathrm{ACCPoly}(X) \in R_{N,Q}$ which is converted from the look-up table f ((14) in Section IV-B), and then the following MLWE ciphertext CT is obtained

$$\mathrm{CT}\left[(-1)^{m_{t-1}}\mathrm{ACCPoly}(X)X^{\varphi_s(\mathbf{ct})}\right]; \tag{8}$$

(*iii*) the CT in (8) is extracted to obtain a *valid* LWE ciphertext $\mathbf{ct}''$ for the constant message of CT by using **SampleExtract**(CT, 0); (*iv*) the secret key of $\mathbf{ct}''$ is switched to the secret key of **ct** by using **KeySwitch** [4]. Therefore, if m_{t-1} controls the sign of the look-up table, or if m_{t-1} is zero, then (8) is correct.

Recently, a programmable bootstrapping (PBS) is proposed [23], which can evaluate any look-up tables. Moreover, without padding PBS (WoP-PBS) is proposed to make the above bootstrapping correct even when $m_{t-1} = 1$. In this paper, WoP-PBS is applied to $R_{N,Q}$ when $Q = Q_0Q_1$ with *shared primes* Q_0 and Q_1, as explained in Section IV-C.

F. INTRODUCTION TO FLOATING-POINT NUMBER SYSTEMS

A floating-point number system can be defined as follows:

Definition 3 (Floating point number system [24])**.** *A floating-point number system is defined by four integers: (i) a radix $\beta \geq 2$; (ii) a precision $p \geq 2$; (iii) two extreme exponents $e_{\min}$ and $e_{\max}$ with $e_{\min} < e_{\max}$. Then a real number x is called a floating-point number if x has at least one representation (s, m, e) such that*

$$x = s \cdot m \cdot \beta^e, \tag{9}$$

where $s \in \{1, -1\}$ is the sign of x, $m = (m_{p-1}.m_{p-2} \ldots m_0)_{(\beta)}$ is a rational number satisfying $0 \leq m < \beta$, and e is an integer satisfying $e_{min} \leq e \leq e_{max}$. It is denoted as $(\beta, p, e_{\min}, e_{\max})$ floating-point number system.

We call m and e in (9) as fraction and exponent, respectively. Since Definition 3 does not guarantee uniqueness of (s, m, e), *normal form*[1] is defined as:

Definition 4 (Normal form [24])**.** *For the $(\beta, p, e_{\min}, e_{\max})$-floating-point number system, (s, m, e) of $x \in \mathbb{R}$ is called a normal form if $1 \leq m < \beta$ or if $e = e_{\min}$ with $0 \leq m < 1$.*

If $x \geq \beta^{e_{\min}}$, we can choose the unique fraction $(m_{p-1}.m_{p-2} \ldots m_1m_0)_{(\beta)}$ with $m_{p-1} \geq 1$, and otherwise, we can uniquely choose m by setting $e = e_{\min}$.

The result of floating-point operation should be rounded to a nearer floating-point number and various rounding algorithms have been studied [24]. In this paper, the rounding zero (RZ) method is used [24] because it is easily implemented by rounding down when the calculated number is represented in *normal form*. Let RZ be a function from any real number x to a floating-point number such that $\mathrm{RZ}(x)$ rounds down if $x \geq 0$ and rounds up if $x < 0$ [24]. Then a tight bound of rounding error is known as follows:

Proposition 3 (Error bound [24])**.** *Let $\top \in \{+, -, \cdot, /\}$ be an arithmetic operation. If $x, y \in \mathbb{R}$ satisfying $\beta^{e_{\min}} \leq |x \top y| \leq (\beta - \beta^{1-p})\beta^{e_{max}}$, then the following inequality holds*

$$|x \top y - \mathrm{RZ}(x \top y)| \leq \beta^{1-p} \min\left(x \top y, \mathrm{RZ}(x \top y)\right). \tag{10}$$

If there is no overflow, Proposition 3 gives an error bound after floating-point operation $\top$ and RZ. However if *normal form* is not used, RZ cannot be implemented by rounding down at the p-digit point of fraction m, and hence a precision degradation occurs meaning that Proposition 3 cannot be applied. Consider $x = 0.1 \cdot 10^1$ in the $(10, 2, 0, 2)$ floating-point number system. If $x^2 = 0.01 \cdot 10^2$ is rounded down on the second digit, the result is 0 but $\mathrm{RZ}(x^2) = 1$ and hence the error is 1 which is larger than 2^{-1} calculated by (10).

The IEEE Standard [25] introduces $(2, 24, -2^7+2, 2^7-1)$ and $(2, 53, -2^{10}+2, 2^{10}-1)$ floating-point number systems, which are called single and double precision, respectively. In this paper, analogues of them are homomorphically implemented in Section V and simulated in Section VI.

III. FLOATING-POINT FULLY HOMOMORPHIC ENCRYPTION

In this section, FPFHE and OD-FHE are defined and a cryptosystem dealing with floating-point messages is proposed, which is used for constructing FPFHE.

A. DEFINITIONS

We formally define FPFHE and OD-FHE by extending the definitions in the homomorphic encryption standard [20], and we will focus on constructing a cryptosystem called OD-FPFHE using the operations $\mathcal{O} = \{+, -, \cdot\}$.

Definition 5 (FPFHE)**.** *A floating-point fully homomorphic encryption is* FHE = (KeyGen, Enc, Dec, Eval) *such that (i) it uses $(\beta, p, e_{\min}, e_{\max})$-floating-point numbers as its message space; (ii) it has an operation set $\mathcal{O} \subseteq \{+, -, \cdot, /\}$ (iii); it uses a rounding function R,* i.e. *for $\top \in \mathcal{O}$ and any x, y with $\beta^{e_{\min}} \leq |x \top y| \leq (\beta - \beta^{1-p})\beta^{e_{max}}$, the ciphertexts* $\mathrm{ct}[x]$ *and* $\mathrm{ct}[y]$ *satisfy the inequality*

$$\left|\mathrm{Dec}\big(\mathrm{Eval}(\mathrm{ct}[x], \mathrm{ct}[y], \top)\big) - \mathrm{R}(x \top y)\right| \leq \beta^{1-p} \min\left(x \top y, \mathrm{R}(x \top y)\right),$$

except negligible probability.

Definition 6 (ξ-Overflow)**.** *For a message space $\mathcal{M}$ with norm $|\cdot|$ and $\xi \in \mathbb{R}$, $x \in \mathcal{M}$ is called ξ-overflow if $|x| \geq \xi$. For a depth-bounded circuit f, x is called ξ-overflow over f if any intermediate result is ξ-overflow while evaluating $f(x)$.*

[1] We do not distinguish the *normal form* and the *subnormal form* given in [24].

Definition 7 (OD-FHE)**.** *For a message space $\mathcal{M}$ with norm $|\cdot|$ and a depth-bounded circuit family $\mathcal{C}$, a $(\xi, \mathcal{C})$-overflow-detectable fully homomorphic encryption is an* FHE *with a homomorphic algorithm such that for any ciphertext* ct *of $m \in \mathcal{M}$ and any $f \in \mathcal{C}$, it returns a ciphertext for the message $m' = 1$ if m is ξ-overflow over f, and $m' = 0$ otherwise, except negligible probability.*

For CKKS with message space $\mathcal{M}_{\mathbb{C}} \subset \mathbb{C}^N$ and $\xi = \max_{x\in\mathcal{M}_{\mathbb{C}}} |x|/2$, it clearly has ξ-overflow numbers over f for any depth-bounded circuit f because $\mathcal{M}_{\mathbb{C}}$ is finite. If the message space $\mathcal{M}$ does not have a norm such as a ring with positive characteristic, e.g. BGV/FV and TFHE, ξ-overflow number is hard to define. However, if BGV/FV are used to encrypt a subset of $\mathbb{Z}$, then ξ-overflow numbers can be defined and exist as well as CKKS.

Generating a proof for overflow in CKKS and BGV/FV is a complicated problem because fixed-point operations are used and extra precision may be required to check if an overflow occurs. In the proposed FPFHE, however, only by inspecting the exponent of homomorphic operation result, we can easily check if an overflow occurs or not when $\xi = \beta^l$ for some $l \in \mathbb{N}$ and just one extra bit in the exponent is required to do that.

Note that the security considered in this paper is Chosen-Plaintext Attacks (CPA)-security. From that viewpoint, the overflow-detecting ciphertext does not give further information to the attacker having the encryption oracle.

B. FLOATING-POINT ENCODING AND DECODING

To implement FPFHE, we must have proper encoding and decoding algorithms between floating-point numbers and the corresponding message polynomials. We choose $q \in \mathbb{N}$ and *shared primes* $Q_0 = \nu 2^{\eta_0} + 1$ and $Q_1 = \nu 2^{\eta_1} + 1$ with $Q_0 > Q_1$, and use the scaling factors $\Delta = Q_1$ and $\Delta' = q\nu$ and β-evaluation map $\Psi_\beta : \mathbb{N}[X] \to \mathbb{Q}$, $a(X) \mapsto a(\beta)\beta^{1-p}$. Then, for a *normal form* (s, m, e) of floating-point number x, **Encode** and **Decode** are defined as follows:

- Encode(s, $m = (m_{p-1}.m_{p-2}\ldots.m_0)_{(\beta)}$, e)
  - Set polynomials $M^{\mathrm{s}}(X) \triangleq \Delta s$, $M^{\mathrm{f}}(X) \triangleq \Delta \sum_{i=0}^{p-1} m_i X^i$, and $M^{\mathrm{e}}(X) \triangleq \Delta' e$.
  - Return $(M^{\mathrm{s}}(X), M^{\mathrm{f}}(X), M^{\mathrm{e}}(X))$.
- Decode($\bar{M}^{\mathrm{s}}(X)$, $\bar{M}^{\mathrm{f}}(X)$, $\bar{M}^{\mathrm{e}}(X)$), where $\bar{M}^i(X) = \bar{\varphi}(\mathrm{CT}^i[M^i(X)])$, $i \in \{\mathrm{s}, \mathrm{f}, \mathrm{e}\}$
  - Set $m = \Psi_\beta(\bar{M}^{\mathrm{s}}(X))$, $e = \bar{M}^{\mathrm{e}}(0)$, and s as the sign of $\bar{M}^{\mathrm{s}}(0)$.
  - Return (s, m, e).

In this paper, $M^{\mathrm{s}}(X)$, $M^{\mathrm{f}}(X)$, and $M^{\mathrm{e}}(X)$ are called sign, fraction, exponent polynomials, respectively, and the following facts will be derived after analyzing bootstrapping error in Section IV-B: (*i*) The size of Q_1 controls the bootstrapping error; (*ii*) The rounding error after tensor product of two MLWE ciphertexts are relatively small when Q_0 and Q_1 are *shared primes*; (*iii*) Moreover, the size of Q_0 should be large enough to take messages increased by a carry and should be determined by depending on β, p, and the carry system which is analyzed in Section V-B. q is also used as integer modulus of LWE ciphertext and *shared primes* Q_0 and Q_1 are found by exhaustive search.

C. CONSTRUCTION OF FLOATING-POINT FULLY HOMOMORPHIC ENCRYPTION

In this section, a formal FPFHE is proposed and its essential homomorphic operations will be introduced in Section V. We adopt the state-of-the-art improved TFHE cryptosystem with tensor product [1] but we change its torus modulus to *shared primes* (See details in [1]).

Our proposed evaluation keys are almost similar to those in [1], but for the proposed OD-FPFHE, these keys have been renamed for better understanding of their roles. The evaluation keys **P**, **BL**, and **Ten** correspond to the key-switching key KSK, bootstrapping key BSK, and relinearization key RLK used in [1], respectively. Note that in [1] the term KSK is used for reverting to canonical LWE ciphertexts, in the proposed FPFHE, **P**, is used for denoting the operation of packing ciphertexts into a single RLWE ciphertext after bootstrapping.

We use additional key-switching key **KS** encrypted with h-sparse secret key to reduce the lattice dimension and error magnitude. Therefore, the improved TFHE [1] takes the key-chain of BSK and KSK, and the proposed FPFHE takes the key-chain of **BL**, **P**, and **KS**. Fig. 1 compares between the improved TFHE [1] and the proposed FPFHE in terms of key-chain of evaluation keys and bootstrapping procedure. In addition, an example of encoding, encryption, decoding, and decryption procedures of the proposed FPFHE is shown in Fig. 2.

Note that generating evaluation keys in FHEs assumes circular security and key-dependent message (KDM) security [26] and we also construct FPFHE under these assumptions. Then, an overall procedure of the proposed FPFHE is given as follows and the detailed procedures will be explained in the next sections.

- Setup(1^λ)
  - Choose q and *shared primes* Q_0 and Q_1. Determine the dimensions N_{gct}, K_{gct}, N_{ct}, K_{ct}, and n. Choose t for the message space $\mathbb{Z}_{2^t}$ of LWE ciphertext. Choose h for the sparsity of secret key.
  - Choose the gadget parameters B_* with $B_*/2$-quality and $B_*/2$-precision, $l_* = \lceil \log Q / \log B_* \rceil - 1$ for all $* \in$ {bl, pack, ten, ks} (See Definition 2).
- KeyGen(1^λ)

  Set the evaluation key as $\boldsymbol{ev}$= (**P**, **BL**, **KS**, **Ten**) and a secret key as $\boldsymbol{sk}$.
  - Generate two ternary MLWE keys $\boldsymbol{sk\text{-}bl}$, $\boldsymbol{sk}$ and a h-sparse LWE key $\boldsymbol{sk\text{-}ks}$.
  - $\mathrm{KS}_{i,j,k} = \mathbf{ct}_{\boldsymbol{sk\text{-}ks}}[sk_{i,j}B_{\mathrm{ks}}^{k+1}]$,$(i, j, k) \in [K_{\mathrm{ct}}, n, l_{\mathrm{ks}}]$.
  - Let $\mathcal{S}$ be the set of non-zero values of $\boldsymbol{sk}$. Then, generate $\mathrm{BL}_i^{(s)}$=$\mathrm{GCT}_{\boldsymbol{sk\text{-}bl}}[\, m_i^{(s)}]$, $i \in [n]$, and

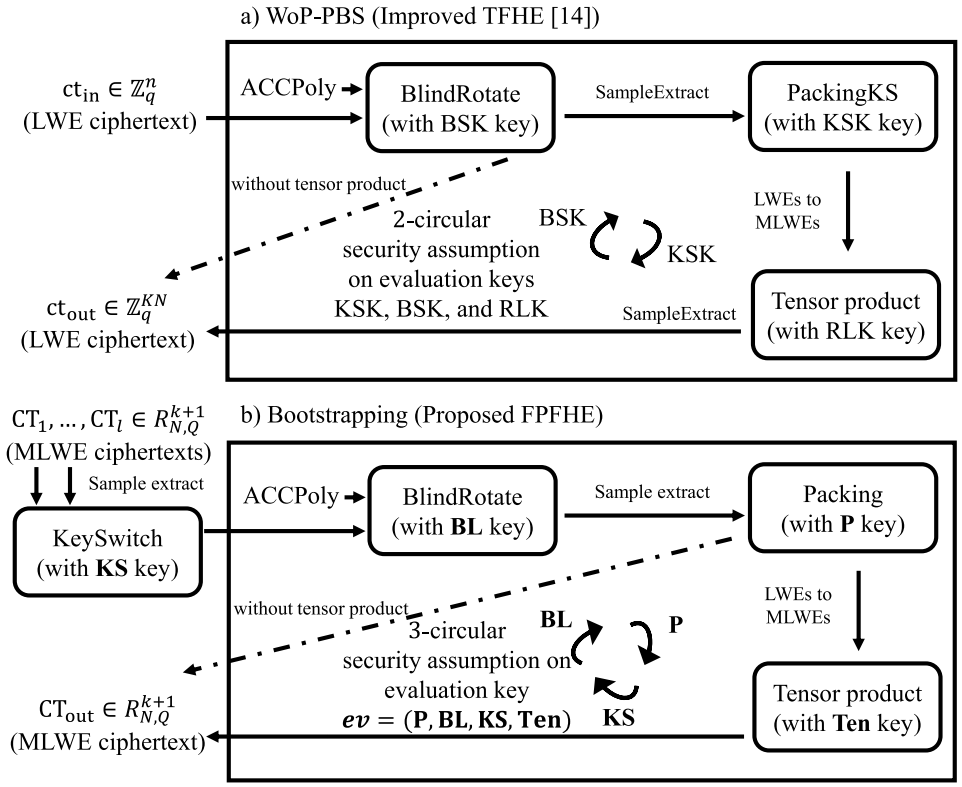

FIGURE 1. Comparison of key-chain and bootstrapping between improved TFHE and the proposed FPFHE.

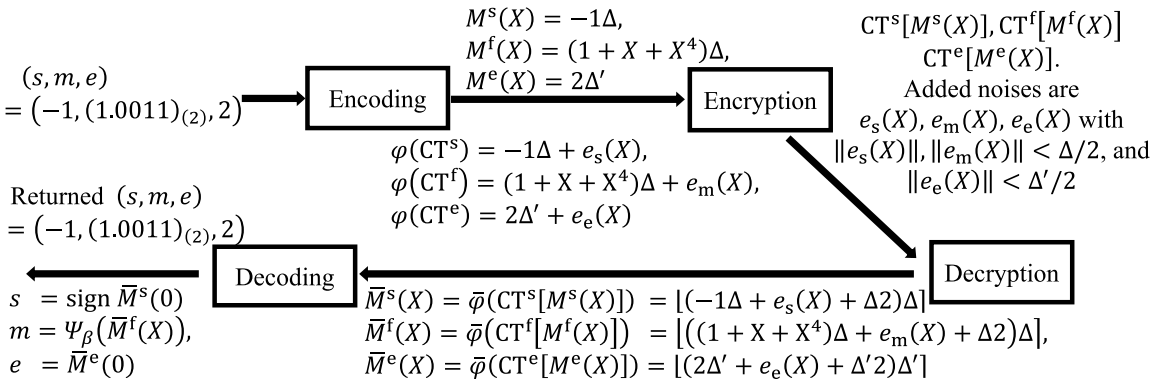

FIGURE 2. Overview of encoding and encryption (resp. decoding and decryption) when (2, 5, 0, 3)-floating point number system, $x = -4.75$, and its *normal form* $(-1, (1.0011)_{(2)}, 2)$ are used.

$s \in \mathcal{S}$ where $m_i^{(s)}$ is 1 if $sk - ks_i = s$, and 0 otherwise.
- $\mathrm{P}_{i,j,k} = \mathrm{CT}_{\boldsymbol{sk}}[sk - bl_{i,j}B_{\mathrm{pack}}^{k+1}]$, $(i, j, k) \in [K_{\mathrm{gct}}, N_{\mathrm{gct}}, l_{\mathrm{pack}}]$.
- $\mathrm{Ten}_{i,j} = \mathrm{CT}_{\boldsymbol{sk}}[sk_{i_1}(X)sk_{i_2}(X)B_{\mathrm{ten}}^{k+1}]$ for all $(i, j) \in [(K_{\mathrm{ct}} + 1)K_{\mathrm{ct}}/2, l_{\mathrm{ten}}]$ where $i_1 = \theta_1^{-1}(i)$ and $i_2 = \theta_2^{-1}(i)$ (θ_1^{-1} and θ_2^{-1} are defined in Section II-A).

- $\mathrm{Enc}_{\boldsymbol{sk}}(x)$
  - Choose the *normal form* (s, m, e) of x and run **Encode**(s, m, e).
  - Return $\mathrm{FCT}_{\boldsymbol{sk}} \triangleq (\mathrm{CT}_{\boldsymbol{sk}}[M^{\mathrm{s}}(X)], \mathrm{CT}_{\boldsymbol{sk}}[M^{\mathrm{f}}(X)], \mathrm{CT}_{\boldsymbol{sk}}[M^{\mathrm{e}}(X)])$.
- $\mathrm{Dec}_{\boldsymbol{sk}}(\mathrm{FCT}_{\boldsymbol{sk}})$
  - Run $\bar{\varphi}_{\boldsymbol{sk}}$ for all inputs to obtain $\bar{M}^{\mathrm{s}}(X)$, $\bar{M}^{\mathrm{f}}(X)$, and $\bar{M}^{\mathrm{e}}(X)$.
  - Return **Decode**$(\bar{M}^{\mathrm{s}}(X), \bar{M}^{\mathrm{f}}(X), \bar{M}^{\mathrm{e}}(X))$.
- $\mathrm{Eval}_{ev}(\mathrm{FCT}^1, \mathrm{FCT}^2, \top)$
  - If $\top \in \{+, -\}$, run **ADD** and if $\top = \cdot$, run **MULT** proposed in Section V.

In Section VI, Q_0 and Q_1 are chosen near 40-bit *shared primes* and every element of $\mathbb{Z}_Q$ are saved in CRT form $(a, b) \in \mathbb{Z}_{Q_0} \times \mathbb{Z}_{Q_1}$ to circumvent 128-bit operations. However, to perform the original gadget decomposition G^{-1}, $\phi^{-1}(a, b)$ should be performed, which is the inverse function of (3) and requires 128-bit operation.

Algorithm 1 $\mathrm{G}_{\mathrm{crt}}^{-1}$, gadget decomposition on CRT

Input: $Q_0 = \nu 2^{\eta_0} + 1, Q_1 = \nu 2^{\eta_1} + 1, \boldsymbol{d} = (a_0, b_0) \times \ldots \times (a_{n-1}, b_{n-1}) \in (\mathbb{Z}_{Q_0} \times \mathbb{Z}_{Q_1})^n$

Output: $c = (c_{0,0}, c_{0,1}, \ldots, c_{0,l-1}, c_{1,0}, \ldots, c_{n-1,l-1}) \in \mathbb{Z}_Q^{1\times nl}$

(Pre-)calculate $\hat{Q}_0 = Q_1^{-1} \mod Q_0$ ▷ Without loss of generality, $Q_1 < Q_0$

for $i \in [n]$ **do**

$x = (a_i - b_i)\hat{Q}_0 \mod Q_1$

$y = x + b_i \mod 2^{\eta_1}$

$z = x\nu + \lfloor (x + b_i)/2^{\eta_1} \rfloor$

$c_i = (c_{i,0}, \ldots, c_{i,l-1}) := \mathrm{G}^{-1}(y + 2^{\eta_1} z)$ ▷ Guarantee that $0 \le y < 2^{\eta_1}$

end for

return $c = (c_1, \ldots, c_n)$

Therefore, we introduce an accelerating gadget decomposition $\mathrm{G}_{\mathrm{crt}}^{-1}$ using *shared primes*, which requires only 64-bit operation when νQ_0 and νQ_1 are less then 2^{64}.

The correctness of $\mathrm{G}_{\mathrm{crt}}^{-1}$ in Algorithm 1 comes from the following property. For any $(a, b) \in \mathbb{Z}_{Q_0} \times \mathbb{Z}_{Q_1}$ with $0 \le a < Q_0$, $0 \le b < Q_1$, and $c = \phi^{-1}(a, b) \in \mathbb{Z}_Q$, there exists $x < Q_0$ such that $c = b + xQ_1 = (b + x) + (x\nu 2^{\eta_1})$ by Euclid division algorithm. Since $Q_1 < Q_0$ and $\phi^{-1}(b, b) = b$, we obtain

$$xQ_1 = \phi^{-1}(a, b) - \phi^{-1}(b, b) = \phi^{-1}(a - b, 0) = (a - b)Q_1\hat{Q}_0,$$

where the equalities hold in $\mathbb{Z}_Q$, which means $x = (a - b)\hat{Q}_0 \mod Q_1$. Then we can split $c = y + 2^{\eta_1} z$ into lower significant η_1-bit number y and more significant η_0-bit number z without calculating the exact value of c, and run gadget decomposition twice by using 64-bit integer operations.

In the next section, error amplification after bootstrapping of the proposed FPFHE is analyzed.

IV. ERROR ANALYSIS AND SEQUENTIAL BOOTSTRAPPING

In this section, we revisit error analysis of bootstrapping in Fig. 1. Section IV-A proves a more generalized result of [18], which is used to analyze the error amplification without independent heuristic even when deterministic gadget decomposition is used for bootstrapping. Section IV-B performs an error analysis for the product of two fraction ciphertexts, which is the worst-case error amplification among homomorphic operations in Section V. Therefore, for a *valid* MLWE ciphertext from **TensorProd**, the server runs **KeySwtich**, **BlindRotate**, and **Packing** sequentially and returns a *valid* MLWE ciphertext which can be multiplied using **TensorProd** again, as in Fig. 1. Therefore, a server can run polynomial number of times due to this bootstrapping. In detail, the following algorithms are run: (*i*) A **BlindRotate** in Algorithm 2 runs to reduce error and raise modulus from q to Q; (*ii*) A **Packing** in Algorithm 4 runs to pack outputs of **BlindRotate** into two MLWE ciphertexts CT_1 and CT_2; (*iii*) A **TensorProd** in Algorithm 3 runs to product CT_1 and CT_2, and p LWE ciphertexts $(\mathbf{ct}'_i)_{i\in[p]}$ are obtained by using SampleExtract. (*iv*) A **KeySwitch** in Algorithm 5 runs to generate *squashed* LWE ciphertext from $(\mathbf{ct}'_i)_{i\in[p]}$. However, after multiplying two fraction polynomials, the message in each coefficient may exceed 2^t. To solve this problem, Section IV-C introduces a sequential bootstrapping with *shared primes*.

A. INVESTIGATION OF SUBGAUSSIAN RANDOM VARIABLES HAVING PYTHAGOREAN ADDITIVITY

For subgaussian random variables X and Y, Corollary 1 requires boundedness of both X and Y to show that XY is subgaussian. However, we will show that it is enough to require that one is bounded and the other is subgaussian.

Lemma 2. *If X is a B-bounded random variable, Y is σ-subgaussian random variable, and X and Y are uncorrelated, then $XY \sim subG(\sqrt{8}B\sigma)$.*

Proof: Since X and Y are uncorrelated, $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y] = 0$. Due to the following inequality

$$\mathbb{P}(|XY| > t) \le \mathbb{P}(|Y| > t/B) \le 2\exp\left(-\frac{t^2}{\sigma^2 B^2}\right), \quad (11)$$

it can be shown that (*i*) the k-th moment is bounded as $\mathbb{E}[|XY|^k] \le (2B^2\sigma^2)^{k/2}\, k\Gamma(k/2)$ where $\Gamma(\cdot)$ is a gamma function, and (*ii*) the moment generating function is bounded as $M_{XY}(s) \le \exp(4B^2\sigma^2 s^2)$ [17]. Therefore $XY \sim subG(\sqrt{8}B\sigma)$. □

However, the factor $\sqrt{8}$ in Lemma 2 is undesirable and by putting additional condition on Y, $\sqrt{8}$ can be removed as follows:

Lemma 3. *Let X be a B-bounded random variable and Y be σ-subgaussian with symmetric distribution,* i.e. $\mathbb{E}[Y^{2n-1}] = 0$ *for all $n \in \mathbb{N}$. If X and Y are independent, then $XY \sim subG(B\sigma)$.*

Proof: Since X and Y are independent, $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y] = 0$. By Lemma 2, there exists a measurable function point-wisely larger than the moment generating function $M_{XY}(s)$ for all $s \in \mathbb{R}$. Then, for any $s \in \mathbb{R}$,

$$\begin{aligned} M_{XY}(s) &= \int_\Omega \mathrm{e}^{sXY} d\mathbb{P} = \int_\Omega \lim_{m\to\infty} \sum_{n=1}^{m} \frac{(sXY)^n}{n!} d\mathbb{P} \\ &\overset{(a)}{=} \lim_{m\to\infty} \sum_{n=1}^{m} \int_\Omega \frac{(sXY)^n}{n!} d\mathbb{P} \\ &\overset{(b)}{\le} \lim_{m\to\infty} \sum_{n=1}^{\lfloor m/2 \rfloor} \int_\Omega \frac{(sB)^{2n} Y^{2n}}{(2n)!} d\mathbb{P} \overset{(c)}{=} M_{BY}(s) \\ &\overset{(d)}{\le} \exp\left(\frac{(\sigma B)^2 s^2}{2}\right), \end{aligned} \quad (12)$$

where (a) holds from monotone convergence of measurable functions with boundedness of $M_{XY}(s)$ [16], (c) holds since Y has a symmetric distribution, and (d) holds from the

property of B-scaled subgaussian. In addition, (b) holds since $\mathbb{E}[(XY)^{2n+1}] = \mathbb{E}[X^{2n+1}] \cdot \mathbb{E}[Y^{2n+1}] = 0$ for all $n \in \mathbb{N}$, and even moment of X is bounded by Lebesgue integral property with $X^{2n} \leq B^{2n}$ for all $n \in \mathbb{N}$ [16]. Therefore, $XY \sim subG(B\sigma)$. □

Corollary 2. *Let $Y_i \sim subG(\sigma)$ be mutually independent random variables having symmetric distribution for all $i \in [n]$. Let $X_1, \ldots, X_n$ be B-bounded random variables where X_i is dependent only on X_0, $Y_0, \ldots, X_{i-1}$, and Y_{i-1}. Then $X_0Y_0, \ldots, X_{n-1}Y_{n-1}$ have Pythagorean additivity.*

Proof: By Lemma 3, $Z_i = X_iY_i \sim subG(\sigma B)$ for all $i \in [n]$. Since Y_i is independent of $Z_1, \ldots, Z_{i-1}$ and X_i is still B-bounded even if $Z_1, \ldots, Z_{i-1}$ are given, $X_0Y_0, \ldots, X_{n-1}Y_{n-1}$ have *Pythagorean additivity* by Lemma 1. □

Note that Corollary 2 does not require that X_i's are subgaussian, meaning that $\mathbb{E}[X]$ does not have to be zero contrary to Corollary 1. Most of additional errors after bootstrapping are the product form of gadget decomposition and errors in ciphertext or secret key. Since the output of deterministic gadget decomposition relies on its input, the mean value of output may be non-zero, to which Corollary 2 can be applied.

B. ERROR ANALYSIS OF THE BOOTSTRAPPING ON A SHARED PRIMES

In this section, **BlindRotate**, **Packing**, **TensorProd**, and **KeySwitch** are sequentially run for bootstrapping as in Fig. 1 and its error analysis is performed. Since **Packing** and **TensorProd** have been widely studied in FHE with GINX-bootstrapping [1], [4], their detailed algorithms are provided in Appendix, and **BlindRotate** and **KeySwitch** [1], [4] are modified to properly operate the proposed OD-FPFHE. Moreover, overall error analysis is performed and the detailed proofs are provided in Appendix.

1) SETUP BEFORE RUNNING BLINDROTATE

We have *valid squashed* LWE ciphertexts $\mathbf{ct}_0, \mathbf{ct}_1, \ldots, \mathbf{ct}_{2p-1}$, where p denotes the precision of floating-point number system, and the accumulate polynomial is given as

$$\text{ACCPoly}(X) = \Delta'' \sum_{i \in [n]} m_i^{out} X^i \in R_{Q,N_{\text{gct}}}, \tag{13}$$

where the scaling factor $\Delta'' \geq \Delta$ and the coefficients m_i^{out} are chosen by the server according to the target look-up table. In this paper, three options of Δ'' are considered: $(i)\Delta'' = \Delta$ if the output of **BlindRotate** is used to run **TensorProd** or to generate a ciphertext of fraction polynomial; $(ii)\Delta'' = \Delta^2$ if the output of **BlindRotate** is used to operate with the output of **TensorProd**, which has the scaling factor Δ^2; $(iii)\Delta'' = \Delta'$ if the output of **BlindRotate** is a ciphertext of exponent message. For a given $\mathbf{ct}_i[\Delta m]$, let $c \in \mathbb{N}$ be the largest number satisfying $2^c | a_j$ of $\mathbf{ct}_i$ and $2^c | b$ of $\mathbf{ct}_i$ for all $j \in [n]$, and let $s \in \mathbb{N}$ be the bit length of m such that $s + c \leq t$. In addition, we adopt multi-output PBS technique with 2^c multi-output in [1]: for given server's target look-up tables $g_j : \mathbb{Z}_{2^s} \to \mathbb{Z}_{2^t}$ for all $j \in [2^c]$, we can convert them into polynomials ACCPoly(X) in (13) with the coefficients

$$m_i^{out} = \sum_{j \in [2^c]} \sum_{i = j \bmod 2^c} g_j\left(\lfloor i 2^s / N_{\text{gct}} \rfloor\right). \tag{14}$$

Then by using **BlindRotate** with ACCPoly(X) and **SampleExtract**$(\cdot, i)$, for all $i \in [2^c]$, the server can obtain 2^c LWE ciphertexts each of which has the message $g_i(m)$, $i \in [2^c]$. Since one term in the summation of m_i^{out} in (14) becomes the message $g_j(m)$ in the output of **BlindRotate** as in (8), the server can get the information about message bit-length of output because of publicity of ACCPoly(X).

2) RUNNING BLINDROTATE

First, we propose Algorithm 2 which is obtained by modifying the blind rotation in [5].

Algorithm 2 BlindRotate

Input: $\mathbf{ct} = (a_1, \ldots, a_n, b) \in \mathbb{Z}_{2N_{\text{gct}}}^{n+1}$, ACCPoly(X) $\in R_{N_{\text{gct}},Q}$

Output: $\mathbf{out} \in \mathbb{Z}_Q^{(N_{\text{gct}}K_{\text{gct}}+1) \times 2^c}$

1: ACC $:= (0(X), \ldots, 0(X), X^{-b}\text{ACCPoly}(X)) \in R_Q^{k+1}$
2: **for** $i \in [n]$ **do**
3: ACC$+= \sum_{j \in \mathcal{S}} (X^{a_i j} - 1)[\text{ACC} \boxdot \text{BL}_i^{(j)}]$
4: **end for** ▷ $\mathcal{S}$ is the set of non-zero values of ***sk-ks***
5: **return out** = (**SampleExtract**(ACC, α))$_{\alpha \in [2^c]}$

The difference between Algorithm 2 and the blind rotation in [5] is Line 3. When $\mathcal{S} = \{-1, 1\}$, i.e. using ternary secret key in blind rotation, the algorithm [5] runs with ACC $+= [(X^{a_i} - 1)\text{ACC} \boxdot \text{BL}_i^{(1)}] + [(X^{-a_i} - 1)\text{ACC} \boxdot \text{BL}_i^{(-1)}]$. To calculate the external product $\boxdot$, gadget decomposition should be run with $\text{G}_{\text{crt}}^{-1}((X^{a_i} - 1)\text{ACC})$ and $\text{G}_{\text{crt}}^{-1}((X^{-a_i} - 1)\text{ACC})$, for evaluating (6). However, in Algorithm 2, only $\text{G}_{\text{crt}}^{-1}(\text{ACC})$ is required to evaluate (6), meaning that the proposed method does not increase the number of NTT operations even if the size of $\mathcal{S}$ increases. Therefore, the number of (I)NTT operations $(K_{\text{gct}} + 1)l_{bl} + 1$ are required compared to the number of (I)NTT operations $|\mathcal{S}|(K_{\text{gct}} + 1)l_{bl} + 1$ in [5].

The error analysis for the output of **BlindRotate** is given as Proposition 5 in Appendix.

3) RUNNING PACKING AND TENSORPROD

After $\mathbf{ct}_i$'s are run by **BlindRotate** for $i \in [p]$, every p resulting ciphertexts generate one MLWE ciphertext of fraction polynomial by using **Packing** (see Algorithm 4). If another MLWE ciphertext from **Packing** is given, these two ciphertexts are multiplied by using **TensorProd** (see Algorithm 3) and an MLWE ciphertext containing product of two fraction polynomials is obtained. Note that **Packing** and **TensorPrd** are widely used in many FHEs [1], [6] and error analysis for the outputs of **Packing** and **TensorProd** in our case is given in Propositions 6 and 7 in Appendix.

Algorithm 3 TensorProd

Input $\mathrm{CT}_1[\Delta m_1(X)] = (a_0, \ldots, a_{K_{\mathrm{ct}}-1}, b)$, $\mathrm{CT}_2[\Delta m_2(X)] = (\mathfrak{a}_0, \ldots, \mathfrak{a}_{K_{\mathrm{ct}}-1}, \mathfrak{b}) \in R_{n,Q}^{K_{\mathrm{ct}}+1}$

Output $\mathrm{OUT}[\Delta^2 m_1(X)m_2(X)] \in R_{n,Q}^{K_{\mathrm{ct}}+1}$

OUT = $\mathfrak{b}\mathrm{CT}_1 + b\mathrm{CT}_2 - (0, \ldots, 0, b\mathfrak{b})$
for $i \in [K_{\mathrm{ct}}]$, $j \le i$ **do**
  Set $k = \theta(i,j)$ and set $\gamma_k = 1/2$ if $i = j$, $\gamma_k = 1$ otherwise
  $v(X) = \mathrm{G}_{\mathrm{crt}}^{-1}\left(\gamma_k\left(a_i\mathfrak{a}_j + a_j\mathfrak{a}_i\right)\right)$
  OUT += $\sum_x v_x(X)\mathrm{Ten}_{k,x}$ ▷ $\mathrm{Ten}_{k,x}$ is the one of evaluation key generated by the user and l_{ten} is its gadget parameter (See Section III-C)
end for
return OUT

Algorithm 4 Packing

Input: $(\mathbf{ct}_i[\Delta m_i])_{i\in[p]} \in \prod_{i\in[p]} \mathbb{Z}_Q^{N_{\mathrm{gct}}K_{\mathrm{gct}}+1}$,
Output: $\mathrm{OUT}\left[\Delta \sum_{i\in[p]} m_i X^i\right] \in R_{N_{\mathrm{ct}},Q}^{K_{\mathrm{ct}}+1}$

Set OUT = $(0(X), \ldots, 0(X), \sum_{i\in[p]} b_i X^i)$ ▷ b_i is the b of $\mathbf{ct}_i$
for $(i,j,x) \in [p, K_{\mathrm{gct}}, N_{\mathrm{gct}}]$ **do**
  $v = \mathrm{G}_{\mathrm{crt}}^{-1}\left(\mathrm{CT}_{i,j,x}\right)$
  OUT += $\sum_{y\in[l_{\mathrm{pack}}]} v_y \mathrm{P}_{j,x,y} X^i$ ▷ $\mathrm{P}_{j,x,y}$ is the evaluation key generated by the user and l_{pack} is its gadget parameter (See Section III-C)
end for
return OUT

However, the resulting ciphertext should be re-run by bootstrapping for two reasons. First, a large error is added after **TensorProd** so that if **TensorProd** is run with this error again, the output may not be *valid*. Second, some coefficient of message polynomial may contain a message larger than the radix β or the degree of message polynomial is greater than $p-1$ (Recall that fraction polynomial should have p message coefficients), that requires homomorphic RZ operation. Therefore, **SampleExtract** is applied to the output of **TensorProd** to generate p LWE ciphertexts again and calculate carryovers to adjust fraction polynomial homomorphically. This process is explained in Section V-B and **KeySwitch** is again requied to make *valid squashed* LWE ciphertexts. Since **Packing** and **TensorProd** have been widely used and studied in FHEs [1], [4], [5], [6], they are listed as Algorithms 4 and 3 in Appendix.

4) RUNNING KEYSWITCH

For p ciphertexts from **TensorProd**, **KeySwitch** is run to reduce the modulus and dimension such that a *valid squashed* LWE ciphertext is obtained.

Let $\mathbf{ct}_i[\Delta^2 m]$ be the LWE ciphertext extracted from the i-th coefficient message by **SampleExtract**$(\cdot, i)$, and let d be the number of zeros from LSB such that $m_{d-1} = \ldots = m_1 = m_0 = 0$ for $m = (m_{t'-1} \ldots m_1 m_0)_{(2)}$. Then **KeySwitch** plays an important role in homomorphically generating the ciphertext for s-bit message $(m_{d+s-1} \ldots m_{d+1} \ldots m_d)_{(2)}$ from the ciphertext for the message m. Since c is chosen by the server, the server can select s to satisfy $s + c \le t$, and re-run **BlindRotate**.

Algorithm 5 KeySwitch

Input $\mathbf{ct}[m\Delta^2] \in \mathbb{Z}_Q^{K_{\mathrm{ct}}N_{\mathrm{ct}}+1}$, the start index d, the number of desirable bootstrapping bits s, and the number of multi-output 2^c.

Output $\mathbf{out}\left[(m_{d+s-1} \ldots m_d)_{(2)} 2^{\log N_{\mathrm{gct}} - s}\right] \in \mathbb{Z}_{2N_{\mathrm{gct}}}^{n+1}$

$\mathbf{ct} \leftarrow \lfloor ct/Q_1 \rfloor \bmod Q_0$
Calculate the bias $= 2^{d+\eta_0-1}v$ and add it to b of $\mathbf{ct}$
$\mathbf{ct} \leftarrow \lfloor ct/v2^{\eta_1+s+c+1+d-q} \rceil \bmod 2^q$
Set $\mathbf{out} = (0, 0, \ldots, 0, b_{\mathrm{ct}}) \in \mathbb{Z}_{2^q}^{K_{\mathrm{ct}}N_{\mathrm{ct}}+1}$ ▷ b_{ct} is b in $\mathbf{ct}$
for $(j,x) \in [K_{\mathrm{ct}}, N_{\mathrm{ct}}]$ **do**
  $v = \mathrm{G}^{-1}(a_{\mathrm{ct},j})$ ▷ $a_{\mathrm{ct},j}$ is the j-th coefficient of a in $\mathbf{ct}$
  $\mathbf{out}$ += $\sum_k v_k \mathrm{KS}_{j,x,k}$
end for
return $\mathbf{out} \leftarrow \lfloor out/2^{q-1-\log N_{\mathrm{gct}}} \rceil 2^c \bmod 2N_{\mathrm{gct}}$

The overall error amplification of bootstrapping is given in Lemma 4 by combining the errors from **BlindRotate**, **Packing**, **TensorProd**, and **KeySwitch**.

Lemma 4. *Assume that* **KeySwitch**(Algorithm 5) *runs with a ciphertext* $\mathbf{ct}[\Delta^2 (m_{t'-1} \ldots m_0)_{(2)}]$ *generated by* **BlindRotate**, **Packing**, **TensorProd**(Algorithms 2, 4, 3), *and* **SampleExtract** *with valid squashed* $(\mathbf{ct}_j)_{j\in[p]}$ *and* $(\mathbf{ct}'_j)_{j\in[p]}$. *Let* d *be the largest number satisfying* $m_{d-1} = \ldots = m_0 = 0$ *and* $\Delta = \Omega(N_{\mathrm{ct}}|\mathcal{E}_{\mathrm{pack}}|)$. *Then the error* $\mathcal{E}_{\mathrm{tot}}$ *of* **KeySwitch** *output with the message* $(m_{d+s-1} \ldots m_d)_{(2)}\ 2^{\log N_{\mathrm{gct}}-s}$ *is bounded except with the probability* $2^{-\Omega(v)}$ *as*

$$O\left(\frac{|\mathcal{E}_{\mathrm{ten}}^{(p-1)}| + \Delta v p \beta^2}{\Delta v 2^{d+\eta_1+s-\log N_{\mathrm{gct}}}} + \frac{\sigma B_{\mathrm{ks}}\sqrt{l_{\mathrm{ks}}N_{\mathrm{ct}}K_{\mathrm{ct}}v}}{2^{q-1-\log N_{\mathrm{gct}}}} + \sqrt{hv}\right), \quad (15)$$

where $\mathcal{E}_{\mathrm{pack}}$ *and* $\mathcal{E}_{\mathrm{ten}}^{(p-1)}$ *are derived in* Propositions 6 *and* 7.

The proof of Lemma 4 is provided in Appendix.

If the overall error amplification $\mathcal{E}_{\mathrm{tot}}$ in (15) is less than or equal to $2N_{\mathrm{gct}}/2^t$, the output of **KeySwitch** is a *valid* ciphertext and hence **BlindRotate** can be applied again. We call that FPFHE is *valid* if its parameters satisfy $|\mathcal{E}_{\mathrm{tot}}| \le 2N_{\mathrm{gct}}/2^t$. Intuitively, a *valid* FPFHE can bootstrap ciphertext for the message $(m_{t-1}m_{t-2} \ldots m_0)_{(2)}\Delta^2$.

5) IMPORTANCE OF USING SHARED PRIME

The reason for using shared primes is to mitigate the distortion in the messages that occur when the modulus is changed from Q_0 to 2^n for the purpose of performing bootstrapping. However, if we change the modulus Q_0 to 2^n, $\lfloor ct \cdot 2^n/Q_0 \rfloor$ is calculated and hence the scaling factor becomes $\lfloor \Delta \cdot 2^n/Q_0 \rfloor$. Therefore, the inherent drawback of this method is that the scaling factor becomes distorted, while

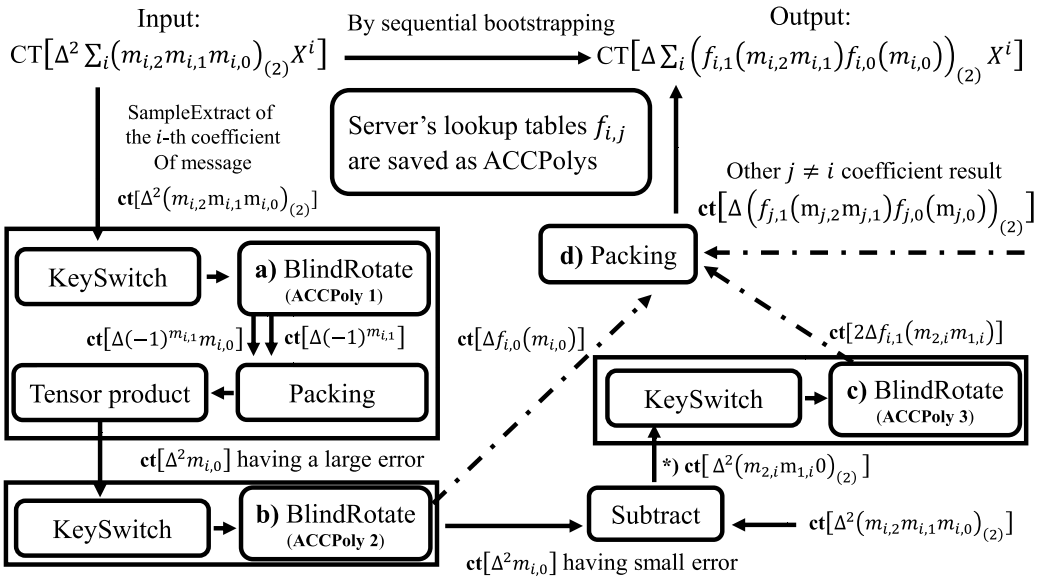

FIGURE 3. An example of sequential bootstrapping to evaluate look-up tables ($f_{i,j}$) when $t = 2$.

the bootstrapping algorithm requires a power of two scaling factor.

However, if *shared primes* are used and the modulus reduction is performed as in Line 1 of Algorithm 5, the approximated scaling factor $\Delta \approx \nu 2^{\eta_1}$ of the message and the approximated modulus $Q_0 \approx \nu 2^{\eta_0}$ share the same parameter ν with negligible error amplification (which is shown in the proof of Lemma 4). Therefore, by dividing **ct** by $\nu 2^i$ for some $i < \eta_1 < \eta_0$ as in Line 3 of Algorithm 5, Δ and Q_0 are easily reduced to the power of two.

In addition, if a non-sparse ternary secret key is used for encrypting $\mathrm{KS}_{j,x,k}$, the third error term of (15) is changed to $O(\sqrt{nv})$. Although the first and the second error terms of (15) can be reduced by increasing $Q_1 = \nu 2^{\eta_1} + 1$ and q, $O(\sqrt{nv})$ cannot be controlled and hence it makes an error floor. However, if a sparse ternary secret key is used for encrypting $\mathrm{KS}_{j,x,k}$, the third error term can be controlled by the sparsity h.

C. SEQUENTIAL BOOTSTRAPPING ON SHARED PRIMES FOR ACCOMMODATING LARGE NUMBERS AND EVALUATING LOOK-UP TABLES

We discuss how to bootstrap ciphertexts of large messages and how to evaluate arbitrary look-up tables (LUT) with the proposed *valid* FPFHE with $t \geq 2$. Note that TFHE/FHEW-based bootstrapping can evaluate arbitrary LUTs having input with t bits and output with t bits, typically $t = 4$ in a practical manner. Apparently, this does not imply the ability to perform operations on arbitrary n-bit input-output LUTs. Instead of arbitrary n-bit input and n-bit output LUTs, if a such LUTs can be decomposed into product forms of t-bit input and t-bit output, it can be implemented using n/t rounds of bootstrapping in a parallel manner.

However, more challenges still exist. If each n-bit messages are encoded as numbers in the range $[0, 2^n - 1]$ and encrypted in each message coefficient of $m(X)$ of ciphertext **CT**, it is difficult to perform bootstrapping on them all at once due to obtaining sign-reversed bootstrapped when most significant bit is 1, as explained in (8)

Note that an attempt to overcome this problem is explained in [1]. Analogously, we explain how to bootstrap when the LUT is decomposed into t-bit input and t-bit output LUTs in this section, that is called sequential bootstrapping. In comparison to [1], the sequential bootstrapping performs over integers modulo $Q = Q_0 Q_1$, which is greater than 2^{64} and hence admits a lot of room for message space, while [1] is limited to a small modulus. In addition, sequential bootstrapping enables to bootstrap carry-over circuit after calculating arithmetic operations.

To explain sequential bootstrapping dealing with worst errors, consider an output MLWE ciphertext of **TensorProd** as an input to sequential bootstrapping as in Fig. 3. First, the operation of sequential bootstrapping is explained by using a simple example.

Suppose that LWE ciphertexts **ct** encrypting each message m_i are given (the expression $m_i = (m_{i2} m_{i1} m_{i0})_{(2)}$ is a binary representation). the precision of each message coefficient is 3 bits, and the server wants to evaluate look-up tables $f_{i,0} : \mathbb{Z}_{2^s} \to \mathbb{Z}_{2^t}$ and $f_{i,1} : \mathbb{Z}_{2^{s'}} \to \mathbb{Z}_{2^t}$ with $s = 1$ and $s' = 2$, where the input to $f_{i,0}$ is an LSB of the message, and the input to $f_{i,1}$ is the remaining 2 bits of the message. Then, the server can evaluate each $f_{i,j}$ on the ciphertext by sequential bootstrapping with **ACCPoly 1**, **2**, and **3**, and its equivalent look-up tables $g_0, \ldots, g_4$ as follows:

output \ location	0	$1\ (=2^s-1)$
$g_0\ (\times\Delta)$	0	1
$g_1\ (\times\Delta)$	1	1
$g_2\ (\times\Delta)$	$f_{i,0}(0)$	$f_{i,0}(1)$
$g_3\ (\times\Delta^2)$	0	1

output \ location	0	1	2	$3\ (=2^{s'}-1)$
$g_4\ (\times 2\Delta)$	$f_{i,1}(0)$	$f_{i,1}(1)$	$f_{i,1}(2)$	$f_{i,1}(3)$

The sequential bootstrapping is performed as follows: (*i*) At Fig. 3 **a)**, two ciphertexts are generated by using **ACCPoly 1** and (8); (*ii*) These two ciphertexts are multiplied and $\mathbf{ct}[\Delta^2 m_{i,0}]$ is generated. Since it contains a large error, the server runs **KeySwitch** and **BlindRotate** with **ACCPoly 2** at Fig. 3 **b)** to optain a partial result $\mathbf{ct}[\Delta f_{i,0}(m_{i,0})]$ and $\mathbf{ct}[\Delta^2 m_{i,0}]$ with small error; (*iii*) $\mathbf{ct}[\Delta^2 m_{i,0}]$ is subtracted from the input $\mathbf{ct}[\Delta^2(m_{i,2}m_{i,1}m_{i,0})_{(2)}]$ and runs **KeySwitch** and **BlindRotate** with **ACCPoly 3** at Fig. 3 **c)** to obtain $\mathbf{ct}[2\Delta f_{i,1}(m_{i,2}m_{i,1})]$; (*iv*) **Packing** is run to collect every partial ciphertext at Fig. 3 **d)**.

The correctness of sequential bootstrapping in Fig 3 relies on two facts. First, the subtraction $*$) part generates negligible error amplification because the error amplification of **BlindRotate** is $O(2^{\eta_0})$-times less than the error amplification of **TensorProd**, which are shown in Propositions 5 and 7 in Appendix. Therefore, the output of **BlindRotate** can be added to and subtracted from the output of **TensorProd** $\mathrm{poly}(\eta_0)$-times, with negligible error amplification. Second, every output ciphertext in Fig. 3 is generated by using the proposed sequential bootstrapping in Fig. 1 (or subroutine), except the output at $*$) part. Therefore, if the proposed FPFHE is *valid*, every output ciphertext is *valid*.

Finally, we formally propose a sequential bootstrapping, which is the special case of WoP-PBS on our *shared prime* as follows.

Lemma 5 (Sequential Bootstrapping on Shared Primes, Analogous of Lemma 5 in [1])**.** *Suppose that a valid FPFHE with* LWE *message space* $\mathbb{Z}_{2^t}$ *for* $t \geq 2$ *and look-up tables* $f_0,\ldots,f_{t'-1}$ *are given. Then, for each bit of the message* $m_i = (m_{i,t'-1}\ldots m_{i,0})_{(2)}$ *of the ciphertext* $\mathbf{ct}[\Delta^2 m_i]$ *obtained by* **BlindRotate**, **Pack**, **TensorProd**, *and* **SampleExtract**, *a valid ciphertext* $\mathrm{CT}[\Delta'' f_{i,j}(m_{i,j})]$ *is generated for any scaling factor* $\Delta'' \geq \Delta$.

Although Lemma 5 is similar to Lemma 5 in [1], NTT can be used in the proposed FPFHE and hence a large message can be processed without generating extra noise, contrary to the improved TFHE using FFT [1]. In addition, the proposed FPFHE allows to pack up to p LWE ciphertexts for multiplying two packed ciphertexts and to bootstrap every coefficient bit by bit, by applying sequential bootstrapping, which is essential for constructing homomorphic floating-point operations.

In the next section, floating-point homomorphic addition and multiplication are proposed for constructing FPFHE.

V. OVERFLOW-DETECTABLE FLOATING-POINT FHE

In Section V-A, floating-point homomorphic addition and multiplication denoted as **ADD** and **MULT** are proposed. Section V-B constructs various homomorphic (sub)algorithms for constructing **ADD** and **MULT**. Section V-C proposes a homomorphic method for normalizing floating-point outputs. Finally, Section V-D introduces a homomorphic algorithms for generating ciphertext of the message indicating overflow occurrence.

Due to Lemma 5, various floating-point homomorphic algorithms can be constructed and we will implement (4,27,-511,511) and (4,12,-127,127) OD-FPFHEs as examples, which achieve double and single precision, respectively. Also, we choose LWE message space with $t = 6$, meaning that each least significant 5-bit messages can be sequentially bootstrapped. Note that, various ACC initial polynomials for implementing each pseudo-code of homomorphic algorithms are listed in Appendix.

A. OVERVIEW OF HOMOMORPHIC OPERATIONS FOR OD-FPFHE: ADDITION, MULTIPLICATION, AND OVERFLOW-DETECTION

For a better understanding of homomorphic operations, we briefly explain how to do floating-point homomorphic addition and multiplication of ciphertexts by using examples in Figs. 4 and 5. Also, their pseudo-codes are given in Algorithms 6 and 7, denoted as **ADD** and **MULT**. Note that all the homomorphic algorithms used for construction of **ADD** and **MULT** are proposed and explained in detail in the following sections (Therefore, you may read the following sections and then come back to this section if needed.)

Addition is performed as follows (e.g. See Fig. 4 **A**) Addition). (*i*) Takes two floating-point ciphertexts; (*ii*) Calculate the maximum of two exponents; (*iii*) For each exponent, subtract exponent values from the max value and calculate the minimum of p and the subtracted value homomorphically; (*iv*) The difference values are sign-reversed and lifted to the monomial exponent multiplied with its sign message by **ConstToExp**; (*v*) The outputs at (*iv*) are multiplied with each fraction polynomial; (*vi*) **CarryAdd** bootstraps each coefficient of the output ciphertext at (*v*) to make it less than the precision β and move its carry to higher-degree coefficients.

In Fig. 4 **A**), there are two noticeable points. The first one is that the max of two exponents is added by 1. because, after moving carry to higher-degree coefficients, the p-th coefficient of fraction polynomial may become non-zero. For this case only, the resulting fraction polynomial should be divided by X and the exponent should be increased by 1 to make it a *normal form*. Note that, it may require a lot of computation at the server to homomorphically check the value of the p-th coefficient. However, if the server adds 1 to the max value in advance, and regards the most significant position in fraction polynomial as p, normalizing process can be easily implemented.

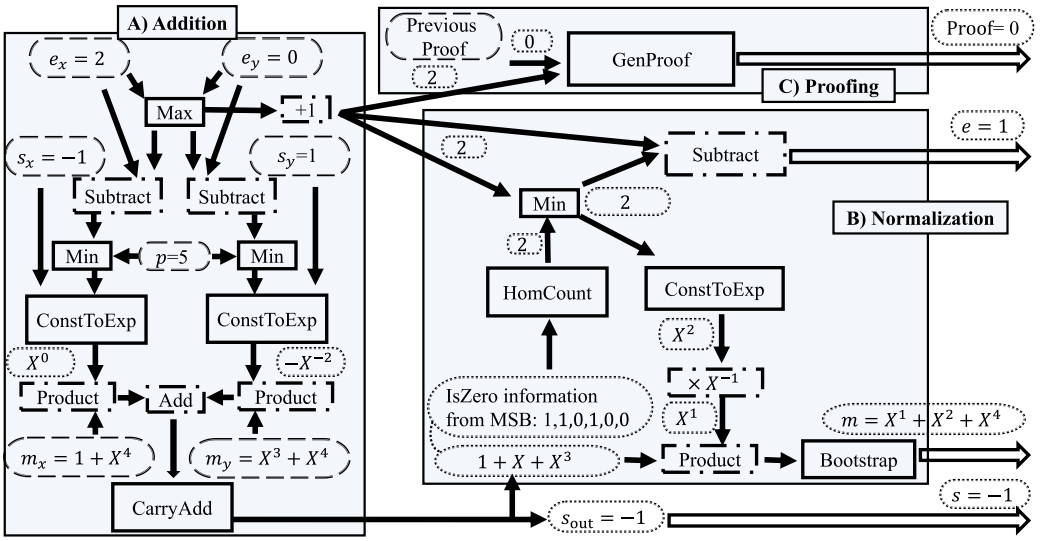

FIGURE 4. Addition: (2,5,0,3)-floating point number system with $x = -4.25$, y=1.5. Normal representation of $x = (s_x, e_x, m_x)$ and $y = (s_y, e_y, m_y)$ is $(-1, (1.0001)_{(2)}, 2)$ and $(1, (1.1)_{(2)}, 0)$, where $(1.0001)_{(2)} = X^4 + 1$ and $(1.1)_{(2)} = X^4 + X^3$. Dashed-circle is the input, dotted-circle is the input value, solid-box is the homomorphic operation requiring sequential bootstrapping, dash-single dotted box is the homomorphic operation without sequential bootstrapping.

Algorithm 6 ADD

Input $\mathrm{FCT}^i = (\mathrm{CT}^i_{\mathrm{sign}}, \mathrm{CT}^i_{\mathrm{frac}}, \mathrm{CT}^i_{\mathrm{exp}})$ for $i = 1, 2$, $\mathrm{CT}_{\mathrm{proof}}$
Output $\mathrm{FCT}_{\mathrm{out}} = (\mathrm{Out}_{\mathrm{sign}}, \mathrm{Out}_{\mathrm{sign}}, \mathrm{Out}_{\mathrm{sign}})$, $\mathrm{CT}'_{\mathrm{proof}}$

1: $\mathrm{CT}^{\max}_{\mathrm{exp}}[\max(m_1, m_2)\Delta'] = \mathbf{Max}\big(\mathrm{CT}^1_{\mathrm{exp}}[m_1\Delta'], \mathrm{CT}^2_{\mathrm{exp}}[m_2\Delta']\big)$
2: **for** i=1:2 **do**
3: $\mathrm{CT}^{\mathrm{diff}}_{\mathrm{exp}}\Big[\min\big(\max(m_1, m_2) - m_i, p\big)\Delta'\Big] = \mathbf{Min}(\mathrm{CT}^{\max}_{\mathrm{exp}}$ - $\mathrm{CT}^i_{\mathrm{exp}}, \mathrm{CT}[p\Delta']))$
4: $\mathrm{tmpCT}_i = \mathbf{TensorProd}\Big(\mathbf{ConstToExp}(\mathrm{CT}^{\mathrm{diff}}_{\mathrm{exp}}, \mathrm{CT}^i_{\mathrm{sign}}), \mathrm{CT}^i_{\mathrm{frac}}\Big)$
5: **end for**
6: $(\mathrm{Out}_{\mathrm{sign}}, \mathrm{Out}_{\mathrm{frac}}, (\mathrm{IsZero}_i)_{i\in[p]}) = \mathbf{CarryAdd}(\mathrm{tmpCT}_1 + \mathrm{tmpCT}_2)$
7: $\mathrm{CT}'_{\mathrm{proof}} := \mathbf{GenProof}(\mathrm{CT}^{\max}_{\mathrm{exp}} + 1, \mathrm{CT}_{\mathrm{proof}})$
8: $(\mathrm{Out}_{\mathrm{frac}}, \mathrm{Out}_{\mathrm{exp}}) = \mathbf{Normalize}((\mathrm{IsZero}_i)_{i\in[p]}, \mathrm{CT}^{\max}_{\mathrm{exp}} + 1, \mathrm{Out}_{\mathrm{frac}})$
9: **return** $(\mathrm{Out}_{\mathrm{sign}}, \mathrm{Out}_{\mathrm{frac}}, \mathrm{Out}_{\mathrm{exp}}, \mathrm{CT}_{\mathrm{proof}'})$

The second one is that the input to **ConstToExp** is forced to take a value between 0 and p by applying **Min** at (*iii*). If the monomial obtained at (*iv*) has the degree less than or equal to $-p$, multiplication of this monomial and each fraction polynomial, as performed at (*v*), generates a new fraction polynomial with coefficient 0 for the term X^i, $0 \le i < p$. However, time consumption of the proposed **ConstToExp** depends on the maximum input size, and therefore forcing its input less than or equal p enhances the overall speed of **ADD**.

For multiplication as in Fig.5 **A**), (*i*) Take two floating-point ciphertexts; (*ii*) Multiply two fraction polynomials; (*iii*) **CarryMul** bootstraps each coefficient of the output at (*ii*) to make it less than the precision β and moves its carry to higher-degree coefficients; (*iv*) Add two exponents; (*v*) Multiply two signs.

Algorithm 7 MULT

Input $\mathrm{FCT}^i = (\mathrm{CT}^i_{\mathrm{sign}}, \mathrm{CT}^i_{\mathrm{frac}}, \mathrm{CT}^i_{\mathrm{exp}})$ for $i = 1, 2$, $\mathrm{CT}'_{\mathrm{proof}}$
Output $\mathrm{FCT}_{\mathrm{out}} = (\mathrm{Out}_{\mathrm{sign}}, \mathrm{Out}_{\mathrm{sign}}, \mathrm{Out}_{\mathrm{sign}})$, $\mathrm{CT}'_{\mathrm{proof}}$

1: $(\mathrm{Tmp}_{\mathrm{frac}}, (\mathrm{IsZero}_i)_{i\in[p]}) = \mathbf{CarryMul}(\mathbf{TensorProd}(\mathrm{CT}^1_{\mathrm{frac}}, \mathrm{CT}^2_{\mathrm{frac}}))$
2: $\mathrm{Tmp}_{\mathrm{exp}} = \mathrm{CT}^1_{\mathrm{exp}} + \mathrm{CT}^2_{\mathrm{exp}}$
3: $\mathrm{Out}_{\mathrm{sign}} = \mathit{Packing} \circ \mathit{BlindRotate} \circ \mathit{KeySwitch} \circ \mathbf{SampleExtract}(\mathbf{TensorProd}(\mathrm{CT}^1_{\mathrm{sign}}, \mathrm{CT}^2_{\mathrm{sign}}), 0)$
4: $\mathrm{CT}'_{\mathrm{proof}} = \mathbf{GenProof}(\mathrm{Out}_{\mathrm{exp}}, \mathrm{CT}_{\mathrm{proof}})$
5: $(\mathrm{Out}_{\mathrm{frac}}, \mathrm{Out}_{\mathrm{exp}}) = \mathbf{Normal}((\mathrm{IsZero}_i)_{i\in[p]}, \mathrm{FCT}^{\max}_{\mathrm{exp}}, \mathrm{Out}_{\mathrm{frac}})$
6: **return** $(\mathrm{Out}_{\mathrm{frac}}, \mathrm{Out}_{\mathrm{sign}}, \mathrm{Out}_{\mathrm{exp}}, \mathrm{CT}'_{\mathrm{proof}})$

For both **ADD** and **MULT**, the homomorphic operation output is converted into a *normal form* by applying **Normalize** as explained in Section V-C. The resulting exponent after performing **ADD** or **MULT** is examined to check if an overflow occurs or not by **GenProof** as in Figs. 4 C) and 5 C), and a ciphertext $\mathrm{CT}'_{\mathrm{proof}}$ containing overflow information is generated, as explained in Section V-D.

B. VARIOUS NON-LINEAR HOMOMORPHIC ALGORITHMS FOR FLOATING-POINT HOMOMORPHIC ADDITION AND MULTIPLICATION

In this section, various non-linear homomorphic algorithms are introduced which are necessary for **ADD** and **MULT**.

First, we propose **Max** in Algorithm 8 to calculate the maximum of two exponent values. The correctness follows

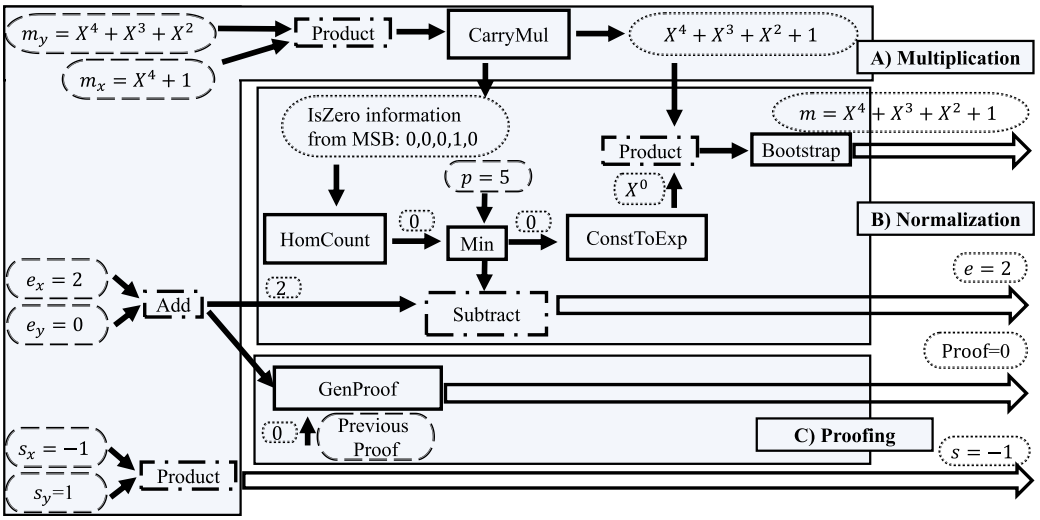

FIGURE 5. Multiplication: (2,5,0,3)-floating point number system with $x = -4.25$, $y = 1.75$. Normal representation of x and y is $(-1, (1.0001)_{(2)}, 2)$ and $(1, (1.11)_{(2)}, 0)$, where $(1.0001)_{(2)} = X^4 + 1$ and $(1.11)_{(2)} = X^4 + X^3 + X^2$.

Algorithm 8 Max

Input $\mathrm{CT}^{(1)}_{\exp}[m_1\Delta']$, $\mathrm{CT}^{(2)}_{\exp}[m_2\Delta']$
Output $\mathrm{Out}_{\exp}[\max(m_1, m_2)\Delta']$

1: $\mathrm{CT}_{\exp}[(m_e m_{e-1} \dots m_0)_{(2)}\Delta'] \leftarrow \mathrm{CT}^{(1)}_{\exp} - \mathrm{CT}^{(2)}_{\exp} + 2^e\Delta'$
▷ e is the smallest integer satisfying $e_{\max} - e_{\min} < 2^e$
2: Generate MLWE ciphertext $\mathrm{CTtmp}_i[m_i\Delta]$ by sequential bootstrapping for all $i \in [e+1]$
3: **for** $i \in [e]$ **do**
4: $\mathrm{CT}[m_e m_i 2^i\Delta'] = \mathit{Packing} \circ \mathit{BlindRotate} \circ \mathit{KeySwitch} \circ$ **SampleExtract** (**TensorProd** (CTtmp_e, CTtmp_i), 0)
5: $\mathrm{Out} \leftarrow \mathrm{Out} + \mathrm{CT}[m_e m_i 2^i\Delta']$
6: **end for**
7: **return** $\mathrm{Out} + \mathrm{CT}^{(2)}_{\exp} - 2^e\Delta'$

Algorithm 9 ConstToExp

Input $\mathrm{CT}_{\exp}[m\Delta']$ where $m = (m_e \dots m_0)_{(2)}$, $\mathrm{CT}_{\mathrm{sign}}[m_s\Delta]$, $f \in \{0, 1\}$
Output $\mathrm{Out}[\Delta X^m]$ if $f = 1$, $\mathrm{Out}[m_s\Delta X^{-m}]$ otherwise

1: Generate $\mathrm{CT}_i[m_i\Delta]$ by using sequential bootstrapping and **Packing**
2: $\mathrm{Out}[(1-f)m_s + f\Delta] = (1-f)\mathrm{CT}_{\mathrm{sign}} + f\Delta$
3: **for** $i \in [e+1]$ **do**
4: $\mathrm{Out} = \mathbf{TensorProd}((X^{(2f-1)2^i} - 1)\mathrm{Out}, \mathrm{CT}_i) + \Delta\mathrm{Out}$
5: **for** $j \in [2^i]$ **do**
6: Out = Run **Packing** with **BlindRotate** $\circ \mathit{KeySwitch} \circ$ **SampleExtract**(Out, j)
7: **end for**
8: **end for**
9: **return** Out

from $\max(x, y) = \mathrm{ReLU}(x - y) + y$ where $\mathrm{ReLU}(x)$ return 0 if $x < 0$ and x otherwise. To examine the sign of message in $\mathrm{CT}^{(1)}_{\exp} - \mathrm{CT}^{(2)}_{\exp}$, assume that both messages take the value between $e_{\min}$ and $e_{\max}$. Since the magnitude of message in $\mathrm{CT}^{(1)}_{\exp} - \mathrm{CT}^{(2)}_{\exp}$ is less than 2^e, the server can add $2^e\Delta'$ to it and check whether m_e is still one or not by using sequenctial bootstrapping in Line 2. Then, a ciphertext $\mathrm{CTtmp}_e[m_e\Delta]$ can mask other ciphertexts after processing the loop Line 3-5, which is equivalent to ReLU.

Next, we propose a homomorphic algorithm for lifting a constant message $m\Delta'$ to the monomial exponent message as ΔX^m in **ConstToExp** (Algorithm 9), which is used for equalizing exponent values before doing addition and for normalizing the resulting ciphertext after doing addition or multiplication. For the former case, **ConstToExp** returns a ciphertext containing the message $m_s\Delta X^{-m}$ for given sign message m_s, and for the latter case, returns a ciphertext containing the message ΔX^m.

The correctness of **ConstToExp** in Algorithm 9 is similar to the correctness of **BlindRotate**. Suppose that $m = (m_e m_{e-1} \dots m_0)_{(2)}$ and $\mathrm{CT}_{\mathrm{sign}}[m_s\Delta]$. When $i = 0$ in Line 4, the message $\Delta^2((X-1)m_0 + 1) = \Delta^2 X^{m_0}$ is assigned to Out. By induction on i, we can show that the message $\Delta^2 X^{m_i \dots m_0{}_{(2)}}$ is assigned to Out if the previous message is $\Delta^2 X^{m_{i-1} \dots m_0{}_{(2)}}$. In addition, we know that Out in Line 4 can be bootstrapped with sufficiently large Q_1 since the error added to Out is relatively small by Propositions 6 and 7.

Next, a homomorphic carryover algorithm for addition is proposed in Algorithm 10, denoted as **CarryAdd**, which is a core part of performing carryover during addition of two fractions. Let $\pi : \mathbb{Z} \to \mathbb{Z}_\beta$, $\pi(x) = x \mod \beta$ be a message-extraction function. After two ciphertexts are added, message in each coefficient should be adjusted by using π and if a carry appears, it should be added to higher-degree coefficients.

There are many ways to move carry, and first we give abstract definition of carry system. Let $\mathfrak{c}_{i\to j} : \mathbb{Z} \to \mathbb{Z}$ be a carry function for all $i, j \in \mathbb{N}$ with $i < j$. After defining a carry collection $\mathfrak{C}_j$ by using $\mathfrak{c}_{i\to j}$ and $\mathfrak{C}_0,\ldots,\mathfrak{C}_{j-1}$, recursively, we define a carry system $\mathfrak{C}$ over polynomial ring $\mathbb{Z}[X]$ as follows:

$$\mathfrak{C}_j\Big(\sum_{i=0}^{n}\alpha_i X^i\Big) = \Big[\alpha_j + \sum_{i=0}^{j-1}(\mathfrak{c}_{i\to j}\circ\mathfrak{C}_i)(\alpha(X))\Big] \in \mathbb{Z},$$
$$\mathfrak{C}\Big(\sum_{j=0}^{n}\alpha_j X^j\Big) = \sum_{j=0}^{n}(\pi\circ\mathfrak{C}_j)(\alpha(X))X^j \in \mathbb{Z}[X], \qquad (16)$$

where $\mathfrak{C}_0\big(\sum_{i=0}^{n}\alpha_i X^i\big) = \alpha_0$, i.e. the constant coefficient does not take a carry. Therefore, the carry system $\mathfrak{C}$ is dependent only on carry functions $\mathfrak{c}_{i\to j}$.

Intuitively, the carry collection $\mathfrak{C}_j$ adds every carry $(\mathfrak{c}_{i\to j}\circ\mathfrak{C}_i)(\alpha(X))$ to the j coefficient α_j from all coefficients strictly less than j. In addition, we will call $\mathfrak{C}$ a *valid* carry system if $\Psi_\beta(\alpha(X)) = (\Psi_\beta\circ\mathfrak{C})(\alpha(X)) \in \mathbb{Q}$ for all $\alpha(X) \in \mathbb{Z}[X]$, i.e. sharing the same value when evaluating β. For **ADD**, $\mathfrak{c}_{i\to i+1}(x) = (x - \pi(x))/\beta$ is used for all $i \in \mathbb{N}$.

Algorithm 10 CarryAdd

Input $\mathrm{CT}_{\mathrm{frac}}[m(X)\Delta^2]$
Output $(\mathrm{Out}_{\mathrm{sign}}, \mathrm{Out}_{\mathrm{frac}}, \mathrm{IsZero}_{i\in[p+1]})$
1: Set $\mathbf{ct}^c = 0$ for $\mathbf{ct}^c \in \mathbb{Z}_Q^{K_{\mathrm{ct}}N_{\mathrm{ct}}+1}$ and $\mathrm{CT}_{\mathrm{frac}} \leftarrow \mathrm{CT}_{\mathrm{frac}}X$
2: **for** $i \in [p+2]$ **do**
3: $\mathrm{Tmp}[\Delta^2\mathfrak{C}_i(m(X))] = \mathbf{SampleExtract}(\mathrm{CT}_{\mathrm{frac}}, i) + \mathbf{ct}^c$
4: $\mathbf{ct}'_i\big[\Delta(\pi\circ\mathfrak{C}_i)(m(X))\big], \mathbf{ct}^c\big[\Delta^2(\mathfrak{c}_{i\to i+1}\circ\mathfrak{C}_i)(m(X))\big] \leftarrow$ Run **SampleExtract**$(\cdot, j)$ with (**BlindRotate** ∘ **KeySwitch**) (Tmp) and $j \in [2]$
5: **end for**
6: $\mathrm{CT}'_{\mathrm{sign}} = (\mathbf{Packing}\circ \mathit{BlindRotate}\circ \mathbf{KeySwitch})(\mathbf{ct}^c)$ ▷ extracting the sign of last carry ciphertext
7: $\mathrm{CT}' = \mathbf{TensorProd}(\mathrm{CT}'_{\mathrm{sign}}, \mathbf{Packing}(\mathbf{ct}'_0, \ldots, \mathbf{ct}'_{p+1}))$ and set $\mathbf{ct}^c = 0$
8: **for** $i \in [p+2]$ **do**
9: $\mathrm{Tmp} = \mathbf{SampleExtract}(\mathrm{CT}', i) + \mathbf{ct}^c$
10: $(\mathbf{ct}''_i, \mathbf{ct}^c, \mathrm{IsZero}_i) = \mathrm{Run}\ \mathbf{SampleExtract}(\cdot, j)$ with (**BlindRotate** ∘ **KeySwitch**) (Tmp) and $j \in [3]$ ▷ IsZero_i contains a message $m\Delta$ with $m = 1$ if the message of $\mathbf{ct}''_i$ is zero, and $m = 0$ otherwise.
11: **end for**
12: **return** $\Big(\mathrm{CT}'_{\mathrm{sign}}, \mathbf{Packing}\big((\mathbf{ct}''_i)_{i\in[p+2]}\big), (\mathrm{IsZero}_i)_{i\in[p+1]}\Big)$

The correctness of **CarryAdd** is as follows: The fraction polynomial is multiplied by X so that the least significant coefficient can take a carry from the previous coefficient. Note that the position of most significant coefficient is $p+1$, not $p-1$, since the fraction polynomial is multiplied by X, and the exponent message is increased by 1 in advance. After **CarryAdd** runs at every iteration on Line 2, a last carry $\mathbf{ct}^c$ appears at $p+1$ coefficient and its sign becomes the sign of the output of adding two ciphertexts. However, if this sign is negative, a packed message from $\mathbf{ct}'_0, \ldots, \mathbf{ct}'_{p+1}$ is sign-reversed, and to fix this problem, $\mathrm{CT}'_{\mathrm{sign}}$ in Line 6 is multiplied to the packed ciphertext to CT' in Line 7. While adjusting and moving carries in Line 8, **CarryAdd** also checks whether each coefficient of $\mathfrak{C}(m(X))$ is zero or not, and generates a ciphertext IsZero_i. Note that IsZero_i is used to make the message in *normal form* in Section V-C.

Similar to **CarryAdd**, a homomorphic carryover algorithm for multiplication **CarryMul** is proposed in Algorithm 11 using a *valid* $\mathfrak{C}$ and carry functions $\mathfrak{c}_{i\to j}$.

Algorithm 11 CarryMul

Input $\mathrm{CT}_{\mathrm{frac}}[m(X)\Delta^2]$
Output $(\mathrm{CT}'_{\mathrm{frac}}, (\mathrm{IsZero}_i)_{i\in[p]})$
1: Set $\mathbf{ct}^c_i = 0$ for all $i \in [2p]$ where $\mathbf{ct}^c_i \in \mathbb{Z}_Q^{K_{\mathrm{ct}}N_{\mathrm{ct}}+1}$
2: **for** $i \in [2p]$ **do**
3: $\mathrm{Tmp}[\Delta^2\mathfrak{C}_i(m(X))] = \mathbf{SampleExtract}(\mathrm{CT}_{\mathrm{frac}}, i) + \mathbf{ct}^c_i$
4: $(\mathbf{ct}'_i\big[\Delta(\pi\circ\mathfrak{C}_i)(m(X))\big], \big(\mathbf{ct}^{cc}_j\big[\Delta^2(\mathfrak{c}_{i\to j}\circ\mathfrak{C}_i)(m(X))\big]\big)_j, \mathrm{IsZero}_i) \leftarrow$ Run sequential bootstrap to generate ciphertexts of i-th message $(\pi\circ\mathfrak{C}_i)(m(X))$ and carryovers $(\mathfrak{c}_{i\to j}\circ\mathfrak{C}_i)(m(X))$ for all $j > i$, from the ciphertext Tmp
5: Update carry $\mathbf{ct}^c_j \mathrel{+}= \mathbf{ct}^{cc}_j$ for all $j > i$.
6: **end for**
7: **return** $\big(\mathbf{Packing}(\mathbf{ct}'_i)_{i\in[2p]}, (\mathrm{IsZero}_i)_{i\in[2p]}\big)$

We use the following carry functions for multiplication. For the given i-th coefficient of l-bit message $m = (m_{l-1}\ldots m_0)_{(2)}$, set $\mathfrak{c}_{i\to i+1}(m\Delta^2) = (m_3m_2)_{(2)}$ and $\mathfrak{c}_{i\to i+2j}(m) = (m_{4j+3}m_{4j+2}m_{4j+1}m_{4j})_{(2)}$ for all $j \geq 1$. Note that it is not trivial to design look-up tables for generating ciphertexts in Line 4 and we list various look-up tables (ACCPoly(X)) for implementing above carry system in Appendix.

For accelerating **CarryMul**, an upper-bound of $\mathfrak{C}_i(m_1(X)m_2(X))$ is derived for any *valid* fraction polynomials $m_1(X)$ and $m_2(X)$ in Proposition 4.

Proposition 4. *Suppose that two polynomials $a(X), b(X) \in \mathbb{N}[X]$ are given where $a_i \leq b_i$, and carry functions $\mathfrak{c}_{i\to j} : \mathbb{N} \to \mathbb{N}$ are given for all $i, j \in \mathbb{N}$ such that $\mathfrak{c}_{i\to j}(x) \leq \mathfrak{c}_{i\to j}(y)$ if $x \leq y$ for all $x, y \in \mathbb{N}$. Then the inequality $\mathfrak{C}_j(\alpha(X)a(X)) \leq \mathfrak{C}_j(\alpha(X)b(X))$ holds for all j and any $\alpha(X) \in \mathbb{N}[X]$.*

Proof is provided in Appendix. By Proposition 4, $\mathfrak{C}_i(m_1(X)m_2(X)) \leq \mathfrak{C}_i(m_1(X)m_{\max}(X)) \leq \mathfrak{C}_i(m^2_{\max}(X))$, where $m_{\max}(X) = (\beta-1)\Delta\sum_{j=0}^{p-1}X^j$. Therefore, for any carry functions $\mathfrak{c}_{i\to j}$, the maximum value of each coefficient is less than or equal to $\mathfrak{C}_i(m^2_{\max}(X))$ while processing carrys. From Proposition 4, we can derive the condition for the modulus Q_0 (as discussed in Section III-B, (*iii*)) such that

$2^{\eta_0-\eta_1} > \log \max_i \mathfrak{C}_i(m^2_{\max}(X))$. Otherwise, messages may be deformed due to small Q_0. Moreover, this upper-bound can be determined in advance, and the range of index i in Line 2 of **CarryMul** can be reduced since the server knows the worst-case smallest index which affects the least significant position $p-1$ by using $\max_i \mathfrak{C}_i(m^2_{\max}(X))$.

C. HOMOMORPHIC ALGORITHMS FOR NORMALIZING FLOATING-POINT OUTPUTS

After doing **CarryAdd** or **CarryMul**, fraction and exponent should be adjusted to a *normal form*. The first step is to count the number of consecutive zeros from the most significant in the fraction until nonzero value appears and we propose **HomCount** in Algorithm 12 for that purpose.

Algorithm 12 HomCount

Input $(\text{IsZero})_{i\in[p']}$

Output $\text{Out}[m\Delta']$, where m is the number of consecutive zeros from the most significant until nonzero value appears

1: $\text{CT}_1[\Delta m_{p'-1}] \leftarrow \text{IsZero}_{p'-1}[\Delta m_{p'-1}]$
2: $\text{Out}[\Delta^2 m_{p'-1}] \leftarrow \Delta\text{IsZero}_{p'-1}[\Delta m_{p'-1}]$
3: **for** $i = p'-2$ to 0 **do**
4: $(\text{CT}_1[\Delta \prod_{j=i}^{p'-1} m_j], \text{CT}_2[\Delta^2 \prod_{j=i}^{p'-1} m_j]) \leftarrow$ **Packing** $\circ$*BlindRotate*$\circ$ **KeySwitch** (**SampleExtract**(**TensorProd**(CT_1, IsZero_i),0))
5: $\text{Out} \mathrel{+}= \text{CT}_2$
6: **end for**
7: **return** $\text{Out}[m\Delta'] \leftarrow$ Bootstrap Out

Note that the message of CT_1 and CT_2 are the same except the scaling factor. In Line 4 of Algorithm 12, **TensorProd**(CT_1, IsZero_i) works as AND gate meaning that the message of its output is Δ only if the messages of both CT_1 and IsZero_i are Δ. Therefore, every returned ciphertext CT_1 in Line 4 encrypts Δ until the message of IsZero_i is 0 at some index i for the first time, and encrypt 0 afterwards. Then, all the returned ciphertexts CT_2 are added to generate Out in Line 5, which has the message scaled by Δ^2. Finally, we bootstrap Out to obtain a ciphertext containing the message about the number of consecutive zeros starting from the most significant until nonzero significant value appears.

Algorithm 13 Normalize :

Input $((\text{IsZero}_i)_{i\in[p]}, \text{CT}_{\text{exp}}, \text{CT}_{\text{frac}})$

Output $\text{Out}_{\text{frac}}, \text{Out}_{\text{exp}}$

1: $\text{CT}^{\min}_{\text{exp}} =$ **Min**(**HomCount**($(\text{IsZero}_i)_{i\in[p]}$), CT_{exp})
2: $\text{CT}_{\text{tmp}} =$ **TensorProd**(**ConstToExp**($\text{CT}^{\min}_{\text{exp}}$), CT_{frac})
3: **for** $i \in [p]$ **do**
4: $\text{out}_i \leftarrow$ (**BlindRotate** $\circ$*KeySwitch*$\circ$ **SampleExtract**)(CT_{tmp}, i)
5: **end for**
6: **return** (**Packing**($\text{out}_0, \ldots, \text{out}_{p-1}$), $\text{CT}_{\text{exp}} - \text{CT}^{\min}_{\text{exp}}$)

By using **HomCount**, we construct **Normalize** in Algorithm 13 for normalizing fraction and exponent of the output ciphertext. Since the number of consecutive zeros in the fraction is counted by **HomCount**, **Normalize** can subtract this value from the exponent ciphertext. However, since the subtracted message may be less than $e_{\min}$, **Normalize** evaluates **Min** in Line 1 and then subtracts this min value from the input exponent ciphertext in Line 4. Note that $\text{CT}^{\min}_{\text{exp}}$ of constant message is converted to CT_{tmp} of monomial with this constant message as its exponent by **ConstToExp** in Line 2. Finally, all out ciphertext out_i are packed into one MLWE ciphertext.

D. GENERATING A PROOF TO DETECT OVERFLOW AND CONSTRUCTING OD-FPFHE

In this section, we propose **GenProof** in Algorithm 14 to generate a ciphertext, called as a proof containing the message indicating whether ξ-overflow occurs or not. Note that we will use the threshold $\xi = 2^{\beta_{\max}+1}$ and an auxiliary numbers $e' \geq \lfloor \log \max(|e_{\max}-1|, |e_{\max}-2e_{\min}+1|)\rfloor + 1$ in Algorithm 14 to examine the value of exponent, which is used for the proposed OD-FPFHE. The **GenProof** operates as follows:

Algorithm 14 GenProof

Input $\text{CT}_{\text{exp}}[m\Delta']$, $\text{CT}_{\text{proof}}[m_{pf}\Delta']$

Output $\text{proof}[(m_{pf}+\alpha)\Delta']$ with $\alpha = 1$ if $m > e_{\max}$, and $\alpha = 0$ otherwise

1: $\text{CT}[(m_{e'-1}\ldots m_0)_{(2)}\Delta'] = \text{CT}_{\text{exp}} + (2^{e'} - e_{\max} - 1)\Delta'$
2: $\text{CT}'_{\text{proof}}[(1-m_{e'-1}))\Delta'] \leftarrow$ ▷ Calculation can be performed via Line 1-5 of Max (Algorithm 8)
3: **return** $\text{CT}'_{\text{proof}} \mathrel{+}= \text{CT}_{\text{proof}}$

GenProof operates similar to **Max** as follows: If the previous proof has the message m_{pf} is 0, i.e., an overflow does not occur while performing the previous operations, then the message m is in $2e_{\min} \leq m \leq 2e_{\max}$. Since $m' \triangleq e_{\max} - m + 1$ is strictly positive if and only if $m \leq e_{\max}$, the e'-th bit in the binary representation of $2^{e'} - m'$ is one if and only if m' is strictly positive, meaning that $\text{CT}'_{\text{proof}}$ in Line 2 has the message about whether $m > e_{\max}$ or not. Otherwise, if the previous proof message m_{pf} has a non-zero message, then it indicates that an overflow has already occurred, hence $\text{CT}'_{\text{proof}}$ has a non-zero message.

Finally, a user can check whether an overflow occurs or not by decrypting the returning proof that is a sum of all the previous proofs. Therefore, by combining FPFHE with bootstrapping failure probability $2^{-\Omega(v)}$ given in Lemma 4 and an arithmetic circuit family $\mathcal{C}$ which contains a circuit f having poly(v) bounded operations, $(\beta^{e_{max}+1}, \mathcal{C})$-OD-FPFHE is constructed.

VI. SECURITY ANALYSIS AND SIMULATION RESULTS

As other FHEs, the proposed OD-FPFHE takes key-dependent message (KDM) and circular security assumptions

TABLE 1. Concrete parameters of OD-FPFHE for various security levels.

$\backslash$	N_{ct}	K_{ct}	N_{gct}	K_{gct}	n	Q_0-1	Q_1-1	q	B_{bl}	B_{pack}	B_{ev}	B_{ks}	h
D128	2^8	13	2^{11}	2	813	$521\cdot 2^{39}$	$521\cdot 2^{29}$	2^{21}	2^{12}	2^{16}	2^{18}	2	131
D160	2^8	16	2^{11}	2	1089	$521\cdot 2^{39}$	$521\cdot 2^{29}$	2^{21}	2^{10}	2^{14}	2^{18}	2	131
D192	2^8	19	2^{11}	3	1292	$521\cdot 2^{39}$	$521\cdot 2^{29}$	2^{22}	2^{10}	2^{13}	2^{18}	2	160

TABLE 2. Time consumption of ADD and Mult for various parameter values (second).

cryptosystem \ operation		ADD(single)	MULT(single)	ADD(double)	MULT(double)
[15]	TFHE-based	54,887	1,152	-	-
	CKKS-based	15,284	**39.3**	-	-
D128	Time (1 thread)	**541**	**452**	**875**	**824**
	Time (4 thread)	192	161	281	277
	Time (10 thread)	135	125	191	229
	Amortized time	53.88	42.19	90.7	87.7
D160	Time (1 thread)	823	674	1,303	1,230
	Time (4 thread)	357	322	565	604
	Time (10 thread)	254	236	382	435
	Amortized time	95.2	78.4	178	204
D192	Time (1 thread)	1,516	1,257	2,439	2,303
	Time (4 thread)	638	556	935	915
	Time (10 thread)	414	376	616	688
	Amortized time	144	129	260	300

to generate public keys [2], [22], [26]. To determine concrete parameter values of OD-FPFHE for achieving target security, we estimate the computational complexity of Primal uSVP and dual lattice attack using k-block BKZ with SVP oracle having the sieving cost $2^{0.292k+16.4}$ [27]. In addition, we apply hybrid primal and dual attack [28] to LWE key-switching key encrypted by h-sparse ***sk-ks***.

A. SIMULATION RESULTS

Every simulation is performed by running Ubuntu 20.04 LTS over Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz having 20 core 40 threads and 256 GB of RAM. PALISADE is compiled with the following CMake flags: WITH-NATIVEOPT = ON (machine-specific optimizations are applied by the compiler), WITH-INTEL-HEXL = ON (AVX-512 acceleration is used), and WITH-TCM = ON (Tcmalloc is used, which is suitable for multi-thread processing), by Clang++ 10.0.0.

In Table 1, the number after D (double precision) refers to the security level in bits. For instance, D128 guarantees 128-bit security of double precision OD-FPFHE under Primal, Dual, and hybrid attacks. Note that OD-FPFHE with D128 can deal with the ciphertexts for both double and single precision messages.

We implement (4,27,-511,511) and (4,12,-127,127) floating-point number systems by using PALISADE v1.11. We simulate **ADD** and **MULT** in Algorithms 6 and 7 by using 1 (single-core), 4, and 10 threads using the parameters in Table 1 and list the operation time in Table 2. In addition, we simulate addition and multiplication time per threads, which is also listed in Table 2 as Amortized time. Therefore, if many threads are available, run time is expected to be close to the amortized time if a circuit is evaluated parallel such as matrix multiplication. However, if a circuit is evaluated by sequential operations, run time is expected to be close to the Time (10 thread) in Table 2.

B. PERFORMANCE EVALUATION OF THE PROPOSED OD-FPFHE

The speed of **ADD** and **MULT** of the proposed OD-FPFHE with D128 is 100x and 2.6x times faster than those of TFHE-based implementation [15]. While the floating-point implementation in [15] inefficiently utilized the existing homomorphic encryption, the proposed approach redesigned homomorphic encryption to align with floating-point operations in the plaintext domain, resulting in a significant speed improvement. Also, the speed of **ADD** is 28.8x faster than that of CKKS-based implementation [15]. However, the speed of **MULT** of CKKS-based implementation [15] is faster than that of the proposed OD-FPFHE because the implementation in [15] does not normalize the result and hence Proposition 3 is not guaranteed.

Next, we arbitrarily choose double precision and single precision messages x without encoding error, i.e. $\mathbf{Decode}(\mathbf{Encode}(x)) = x$ as follows:

$$\begin{aligned}
x_d^1 &= -9.1763514236254290 * 10^{-32},\\
x_d^2 &= 6.2467247246375865 * 10^{-24},\\
x_d^3 &= 2.4523526872362373 * 10^{22},\\
x_d^4 &= -5.4324663335297274 * 10^{17},\\
x_s^1 &= -2.7914999921796382 * 10^{-15},\\
x_s^2 &= 8.3867001884896375 * 10^{-12},\\
x_s^3 &= 1.82634005135360 * 10^{14},\\
x_s^4 &= -6.278269952 * 10^{9},
\end{aligned}$$

where x_d^i denote double precision message and x_d^i denote single precision message. Then we evaluate $z_1 = x^1 + x^2$, $z^2 = x^3 - x^4$, $z^3 = z_1 \cdot z_2$, and $z_4 = z_3^2$ over the ciphertext domain, and the results together with the correct calculation values are listed in Table 3. Note that the correct calculation values are rounded down. It can be easily checked that the

TABLE 3. Homomorphic computation results For double precision(64-bit) and single precision(32-bit).

\	$x_d^1 + x_d^2$	$x_d^3 - x_d^4$	$(x_d^1 + x_d^2)(x_d^3 - x_d^4)$	$((x_d^1 + x_d^2)(x_d^3 - x_d^4))^2$
Correct value	$\mathbf{6.2467246328740732 \cdot 10^{-24}}$	$\mathbf{2.4524070118995769 \cdot 10^{22}}$	$\mathbf{0.153195112910661}$	$\mathbf{0.023468742619710}$
OD-FPFHE	$\mathbf{6.2467246328740717 \cdot 10^{-24}}$	$\mathbf{2.4524070118995730 \cdot 10^{22}}$	$\mathbf{0.153195112910657}$	$\mathbf{0.023468742619709}$
\	$x_s^1 + x_s^2$	$x_s^3 - x_s^4$	$(x_s^1 + x_s^2)(x_s^3 - x_s^4)$	$[(x_s^1 + x_s^2)(x_s^3 - x_s^4)]^2$
Correct value	$\mathbf{0.153195112910661}$	$\mathbf{0.023468742619710}$	$\mathbf{1531.23945}$	$\mathbf{2344694.28}$
OD-FPFHE	$\mathbf{0.153195112910657}$	$\mathbf{0.023468742619709}$	$\mathbf{1531.23937}$	$\mathbf{2344694.00}$

error between correct value and decryption result is bounded as expected from Proposition 3 if an overflow does not occur. All the simulation codes are open to the public.

C. CONCLUSIONS AND FUTURE WORKS

We proposed a floating-point fully homomorphic encryption with homomorphic normalization. Since floating-point number systems are is widely used in many areas such as high-precision deep learning and control systems, the proposed FPFHE can guarantee both privacy and accuracy for many precise applications. In addition, we proposed an OD-FPFHE which also has many applications. For instance, it is quite useful for solving satellite collision problem and performing accurate continual learning while keeping the privacy of training data because the encrypted training data causing an overflow can be excluded at the training stage to avoid the degradation in learning.

However, there are still many issues to be studied. First, we do not propose homomorphic floating-point division algorithm. Since algebraic structures of many FHEs are not in Euclidean domain, a simple and natural division algorithm is not trivial, and hence we have been searching an effective and fast division circuit suitable for the proposed FPFHE. In addition, floating-point homomorphic elementary functions such as exponential, logarithm, and N-th root functions are also desirable in privacy-preserving machine learning.

One of the critical disadvantages of current OD-FPFHE is slow operation time. However, the operation speed can be improved in many aspects as follows: Since a large Q affects bootstrapping time, reducing Q should be investigated. For instance, randomized gadget decomposition is reported that it reduces error amplification after running GSW-like multiplication [21]. Therefore, effective randomized gadget decomposition for OD-FPFHE and both rigorous and practical error analysis will improve the operation time.

Also, studies of accelerating speed of FHEs on GPUs have been performed [29] and in the near future, OD-FPFHE is expected to benefit from such hardware acceleration, potentially leading to the improved performance.

APPENDIX A
PROOF OF PROPOSITION 4

We prove Proposition 4 by induction on j of $\mathfrak{C}_j$. Assume that $\mathfrak{C}_0, \ldots, \mathfrak{C}_{j-1}$ satisfy Proposition 4. Since every coefficient of $b(X)$ is greater than or equal to the corresponding coefficient of $a(X)$, $\mathfrak{C}_0(a(X)\alpha(X)) = a_0\alpha_0 \le b_0\alpha_0 \le \mathfrak{C}_0(b(X)\alpha(X))$ holds. Then for the index j,

$$\begin{aligned}
&\mathfrak{C}_j(a(X)\alpha(X)) \\
&= \sum_{i\in[j+1]} a_i\alpha_{j-i} + \sum_{i\in[j]} (\mathfrak{c}_{i\to j} \circ \mathfrak{C}_i)(a(X)\alpha(X)) \\
&\le \sum_{i\in[j+1]} b_i\alpha_{j-i} + \sum_{i\in[j]} (\mathfrak{c}_{i\to j} \circ \mathfrak{C}_i)(b(X)\alpha(X)) \\
&= \mathfrak{C}_j(b(X)\alpha(X))
\end{aligned}$$

□

APPENDIX B
ERROR ANALYSIS OF BLINDROTATE

Proposition 5. *Assume that* **BlindRotate** *in* Algorithm 2 *runs with a valid squashed* **ct** *and returns* $\mathrm{out}_\alpha \in \mathbb{Z}_Q^{N_{gct}K_{gct}+1}$ *for the message* $(\mathrm{ACCPoly}(X)\, X^{-\varphi(\mathbf{ct})})_\alpha$. *Then for any* $\alpha \in [N_{gct}]$, *the error in* α*-coefficient, denoted as* $\mathcal{E}_{\mathrm{bl}}^{(\alpha)}$ *of* out_α, *is bounded except with probability* $2^{-\Omega(v)}$ *as follows*:

$$|\mathcal{E}_{\mathrm{bl}}^{(\alpha)}| = O\left(B_{\mathrm{bl}}\sqrt{\nu N_{\mathrm{gct}}K_{\mathrm{gct}}}\left(n + \sigma\sqrt{nl_{\mathrm{bl}}}\right)\right). \tag{17}$$

Proof: When Algorithm 2 runs with i on Line 2, by using (7) and CMux gate analysis in Section 3.4 of [4], the additive error is derived as follows:

$$\begin{aligned}
&\sum_{j\in[l_{\mathrm{bl}}]} (X^{a_i} - 1)\mathrm{G}_{\mathrm{crt}}^{-1}(\mathrm{ACC}^{(i)})E_j^1(X) \\
&\quad + \sum_{j\in[l_{\mathrm{bl}}]} (X^{-a_i} - 1)G_{\mathrm{crt}}^{-1}(\mathrm{ACC}^{(i)})E_j^{-1}(X) \\
&\quad + \sum_{j\in[\bar{l}_{\mathrm{bl}}]} \left[(X^{a_i} - 1)A_j^1(X) + (X^{-a_i} - 1)A_j^{-1}(X)\right]sk - bl(X)_j,
\end{aligned} \tag{18}$$

where $\mathrm{ACC}^{(i)}$ is the computed value after the $(i-1)$-st iteration on Line 3, $A_j^1(X)$ and $A_j^{-1}(X)$ are gadget error polynomials, and $E_j^1(X)$ and $E_j^{-1}(X)$ are j-column error polynomials of BL_i^1 and BL_i^{-1}, respectively.

Since the errors $E_j^1(X)$ and $E_j^{-1}(X)$, and the secret key $sk - bl_j(X)$ follow symmetric distribution and each of them is multiplied by independent and bounded random variable, then the summands in each summation in (18) have *Pythagorean additivity* by Corollary 2. By induction on i, we can obtain (17) by using (2) and Proposition 1. □

APPENDIX C
ERROR ANALYSIS OF PACKING

Proposition 6. *Assume that* **Packing** *in* Algorithm 4 *runs with p ciphertexts* $(\mathbf{ct}_i[\Delta m_i])_{i\in[p]}$ *where* $\mathbf{ct}_i \in \mathbb{Z}_Q^{N_{\mathrm{gct}}K_{\mathrm{gct}}+1}$ *are generated by running* **BlindRotate** *in* Algorithm 2 *with valid squashed* $(\mathbf{ct}'_j)_{j\in[p]}$*, and returns a ciphertext* $\mathrm{OUT}[\Delta\sum_{i\in[p]} m_iX^i] \in R_{N_{\mathrm{ct}},Q}^{K_{\mathrm{ct}}+1}$*. Then for any coefficient* $\alpha \in [2p]$*, the* α*-coefficient error* $\mathcal{E}_{Pack}^{(\alpha)}$ *of* OUT *is bounded except with probability* $2^{-\Omega(v)}$ *as follows*:

$$|\mathcal{E}_{\mathrm{pack}}^{(\alpha)}| = |\mathcal{E}_{\mathrm{bl}}^{(\alpha)}| + O\Big(B_{\mathrm{pack}}\sqrt{vN_{\mathrm{gct}}K_{\mathrm{gct}}}(p+\sigma\sqrt{l_{\mathrm{pack}}p})\Big). \tag{19}$$

Proof: The decryption result of the output of **Packing** OUT in Algorithm 4 is as follows:

$$\begin{aligned}
&\sum_{i,j,x,y} \mathrm{G}_{\mathrm{crt}}^{-1}(\mathrm{CT}_{i,j,x})_y(sk - bl_{j,x}B_{\mathrm{pack}}^{y+1} + E'_{j,x,y}(X))X^i \\
&\quad + \sum_{i\in[p]} b_iX^i \\
&= \sum_i \varphi(\mathrm{CT}_i)X^i + \sum_{j,x}\Big(\sum_i A'_{j,x,y}X^i\Big)sk - bl_{j,x} \\
&\quad + \sum_{i,j,x,y} \mathrm{G}_{\mathrm{crt}}^{-1}(\mathrm{CT}_{i,j,x})_yX^iE'_{j,x,y}(X),
\end{aligned} \tag{20}$$

where $A'_{j,x,y}$ is a gadget error and $E'_{j,x,y}(X)$ is the error polynomial of packing key $\mathrm{P}_{j,x,y}$, which is already analyzed (See details in [4]).

Since the errors $E'_{j,x,y}(X)$ and the secret key $sk - bl_{j,x}$ follow the symmetric distribution and each of them is multiplied by independent and bounded random variable, the second and third summands in (20) have *Pythagorean additivity* by using Corollary 2. Since the first summation in (20) is $\sum_i(\Delta m_iX^i + \mathcal{E}_{\mathrm{bl}}^{(i)})$, (19) holds. □

APPENDIX D
ERROR ANALYSIS OF TENSORPROD

Proposition 7. *Assume that* **TensorProd** *in* Algorithm 3 *runs with two ciphertexts* $\mathrm{CT}_1[m_1(X)]$ *and* $\mathrm{CT}_2[m_2(X)] \in R_{N_{\mathrm{ct}},Q}^{K_{\mathrm{ct}}+1}$ *which are generated by* Algorithms 2 *and* 4 *with valid squashed* $(\mathbf{ct}_j)_{j\in[p]}$ *and* $(\mathbf{ct}'_j)_{j\in[p]}$*, and returns* $\mathrm{OUT}[\Delta^2m_1(X)m_2(X)] \in R_{N_{\mathrm{ct}},Q}^{K_{\mathrm{ct}}+1}$*. If* $\Delta = \Omega(N_{\mathrm{ct}}|\mathcal{E}_{\mathrm{pack}}|)$ *is chosen and the coefficients of both* $m_1(X)$ *and* $m_2(X)$ *are bounded by* $\Delta(\beta-1)$*, then for any* $\alpha \in [2p]$*, the* α*-coefficient error* $\mathcal{E}_{Ten}^{(\alpha)}$ *of* OUT *is bounded except with probability* $2^{-\Omega(v)}$ *as follows*:

$$O\Big(\Delta p\beta\Big|\mathcal{E}_{\mathrm{pack}}^{(p-1)}\Big| + K_{\mathrm{ct}}^2N_{\mathrm{ct}}^2l_{\mathrm{ten}}B_{\mathrm{ten}} + \sigma B_{\mathrm{ten}}K_{\mathrm{gct}}\sqrt{l_{\mathrm{ten}}N_{\mathrm{gct}}v}\Big) \tag{21}$$

Proof: We perform decryption of OUT returned from Algorithm 3 as follows:

$$\begin{aligned}
&\sum_{k,x} \mathrm{G}_{\mathrm{crt}}^{-1}\Big(\gamma_k\big[a_j\mathfrak{a}_i + a_i\mathfrak{a}_j\varphi(\mathrm{CT}_1)\varphi(\mathrm{CT}_2) \\
&\quad + \sum_{k,x} A''_{k,x}(X)sk_i(X)sk_j(X) \\
&\quad + \sum_{k,x} \mathrm{G}_{\mathrm{crt}}^{-1}\Big(\gamma_k\big[a_j\mathfrak{a}_i + a_i\mathfrak{a}_j\big]\Big)_x E''_{k,x}(X),
\end{aligned} \tag{22}$$

where $A''_{k,x}$ are the gadget errors and $E''_{k,x}(X)$ are the errors in $\mathrm{Ten}_{k,x}$. Also, the noise $\varphi(\mathrm{CT}_1)\varphi(\mathrm{CT}_2) - m_1(X)m_2(X)$ is calculated as follows:

$$m_1(X)\mathcal{E}_{\mathrm{pack},2}(X) + m_2(X)\mathcal{E}_{\mathrm{pack},1}(X) + \mathcal{E}_{\mathrm{pack},1}(X)\mathcal{E}_{\mathrm{pack},2}(X), \tag{23}$$

where $\mathcal{E}_{\mathrm{pack},1}(X)$ and $\mathcal{E}_{\mathrm{pack},2}(X)$ are the packing errors in CT_1 and CT_2, respectively.

Since the maximum degree of both message polynomials $m_1(X)$ and $m_2(X)$ is $p-1$, the $(p-1)$-st coefficient of $m_1(X)\mathcal{E}_{\mathrm{pack},2}(X) + m_2(X)\mathcal{E}_{\mathrm{pack},1}(X)$ is expressed as

$$\sum_{i\in[p]}\Big(m_{1,i}\mathcal{E}_{\mathrm{pack},2}^{(p-1-i)} + m_{2,i}\mathcal{E}_{\mathrm{pack},1}^{(p-1-i)}\Big) \tag{24}$$

by (2). Since each coefficient is the sum of p product of message and packing error, without loss of generality, we analyze a worst-case error of $(p-1)$-st coefficient having messages $m_1(X) = m_2(X) = \sum_{i\in[p]}\Delta(\beta-1)X^i$. Since $|\mathcal{E}_{\mathrm{pack},1}(X)\mathcal{E}_{\mathrm{pack},2}(X)| = O(N_{\mathrm{ct}}|\mathcal{E}_{\mathrm{pack},1}^{(p-1)}|^2)$, by using the fact $\Delta = \Omega(N_{\mathrm{ct}}|\mathcal{E}_{\mathrm{pack}}|)$, (18), and (20), then (23) is bounded as $O(\Delta p\beta|\mathcal{E}_{\mathrm{pack}}^{(p-1)}|)$. Moreover for (22), the first summation is bounded as $O(K_{\mathrm{ct}}^2N_{\mathrm{ct}}^2l_{\mathrm{ten}}B_{ten})$ and the second summation is bounded as $O\big(\sigma B_{\mathrm{ten}}K_{\mathrm{gct}}\sqrt{l_{\mathrm{ten}}N_{\mathrm{gct}}v}\big)$ by using Corollary 2. □

APPENDIX E
PROOF OF LEMMA 4

After running Algorithm 5 with input **ct**, this **ct** is multiplied by three values Δ^{-1}, $(v2^{\eta_1+s+c+1+d-q})^{-1}$, and $2^{c+1+\log N_{\mathrm{gct}}-q}$, as listed in Lines 1, 3, and 9. Since $2^{c+\log N_{\mathrm{gct}}+1-q}/\Delta v2^{\eta_1+s+c+1+d-q}$ is multiplied to **ct**, the error in **ct** becomes the left of first term of (15). When Algorithm 5 runs on Line 1, $O(\sqrt{K_{\mathrm{ct}}N_{\mathrm{ct}}v})$-bounded flooring errors are added, which is negligible compared to $|\mathcal{E}_{\mathrm{ten}}^{(p-1)}|/\Delta$.

Next, we consider **ct** in Line 1 as the ciphertext modulo $v2^{\eta_0}$. If the modulus of **ct** is changed from $Q_0 = v2^{\eta_0}+1$ to $v2^{\eta_0}$, then $O(\sqrt{K_{\mathrm{ct}}N_{\mathrm{ct}}})$ errors are added by the following decryption equation on $\mathbb{Z}$:

$$b - a_is_i = m + e + \bar{h}Q_0 = m + e + \bar{h} + \bar{h}v2^{\eta_0} \in \mathbb{Z},$$

for some $\bar{h} \in \mathbb{Z}$ where $\bar{h} = O(\sqrt{K_{\mathrm{ct}}N_{\mathrm{ct}}})$ for ternary secret key [6], which is negligible compared to other noises. Moreover, we regard the message $mQ_1 = mv2^{\eta_1} + mv$ as a message $mv2^{\eta_1}$ with error mv. Therefore, the message of the ciphertext **out** becomes $(m_{f+s}\dots m_f)_{(2)}2^{\log N_{\mathrm{gct}}-s}$ after terminating Algorithm 5 and up to $mv \le p(\beta-1)^2v$ error is added. Therefore, extra noise $vp(\beta-1)^2$ is derived from the message part and becomes the rest of first term of (15).

After rounding in Line 3, $O(\sqrt{N_{\mathrm{ct}}K_{\mathrm{ct}}v})$-bounded rounding error is added. After running on Line 5, errors in $\mathrm{KS}_{j,x,k}$ are

added, which is $O(\sigma B_{\mathrm{ks}}\sqrt{l_{\mathrm{ks}}N_{\mathrm{ct}}K_{\mathrm{ct}}v})$-bounded random variable by Corollary 2. Both errors are divided by $2^{q-1-\log N_{\mathrm{gct}}}$ after running Line 9, which is the second term in (15).

Finally, the rounding errors after running Line 9 are added. However, we use h-sparse secret key for encrypting KS and only h rounding errors are added. By using subgaussian property with Corollary 2, this error is $O(\sqrt{hv})$-bounded, and hence the third term of (15) is derived. □

APPENDIX F
ACCUMULATE POLYNOMIALS FOR IMPLEMENTING HOMOMORPHIC ALGORITHMS MAX, MIN, CONSTTOEXP, CARRYADD, AND CARRYMUL

In this appendix, we propose accumulate polynomials for implementing various homomorphic algorithms necessary for constructing (4, 27, −511, 511) floating-point FHE.

A. MIN AND MAX

To implement and accelerate **Max** in the (4,27,-511,511) floating-point FHE with $e = 10$, we apply two sequential bootstrappings to generate ciphertext having 4-bit messages in Line 2 of **Max** in Algorithm 6. Since the exponential bit is 10, there is remaining significant 2bits, therefore **ACCPoly 4** is used once to bootstrap the remaining two message bits in Line 2 and we obtain two ciphertexts having the messages $m_{10}(m_9m_8 2^8 - 2^{10})\Delta'$ and $m_{10}\Delta$ at once. Note that **Min** can be implemented by using the equation $\min(x, y) = -\mathrm{ReLU}(x - y) + x$ in a similar way.

output \ location	0∼3	4∼15 (index as i)
g_0 outputs ($\times 2^8\Delta'$)	0	$i-4$
g_1 outputs ($\times\Delta$)	0	1

B. ConstToExp

To implement and accelerate **ConstToExp** in the (4,27,-511,511) floating-point FHE, **ACCPoly 5** and **ACCPoly 6** are used for bootstrapping in Line 2 in Algorithm 7. Since the message is cut and $m \le p = 27 < 2^5$ is less than 5 bits, 3 bits from the least significant bit are sequentially bootstrapped by using **ACCPoly 5** and hence CT[Δm_0], CT[Δm_1], and CT[Δm_2] are obtained. These ciphertexts generate CT[$\Delta X^{4m_2+2m_1+m_0}$] in Line 3 with $i = 0, 1$, and 2. Moreover, two most significant bits are bootstrapped by using **ACCPoly 6** and ciphertext CT[$\Delta X^{16m_4+8m_3}$] is constructed by using **Packing** with **ACCPoly 6** at once.

output \ location	0	1	2	3	4	5	6	7
g_0 outputs ($\times\Delta$)	0	1	0	1	0	1	0	1
g_1 outputs ($\times\Delta$)	0	0	1	1	0	0	1	1
g_2 outputs ($\times\Delta$)	0	0	0	0	1	1	1	1
g_3 outputs ($\times\Delta$)	1	1	1	1	1	1	1	1

output \ location	0	1	2	3	4	5	6	7
g_0 outputs ($\times\Delta$)	1	0	0	0	-	-	-	-
g_1 outputs ($\times\Delta$)	0	1	0	0	-	-	-	-
g_2 outputs ($\times\Delta$)	0	0	1	0	-	-	-	-
g_3 outputs ($\times\Delta$)	0	0	0	1	-	-	-	-

C. CarryAdd

To implement and accelerate **CarryAdd** in the (4,27,-511,511) floating-point FHE, **ACCPoly 7** is used for bootstrapping in Line 4 of Algorithm 8. Note that the i-th coefficient message m_i corresponding to the sum of two fraction polynomials takes a value between −6 and 6. If a carry message takes a value from −2 to 1, then $\mathbf{ct}'_i$ and $\mathbf{ct}^c$ are obtained by adding $8\Delta^2$ and bootstrapping with **ACCPoly 7**. When bootstrapping is performed at $i = p + 1$ in Line 3, **ACCPoly 8** is used to obtain the sign of fraction and message at once. Note that the i-th coefficient message m_i after packing and tensor product in Line 8 takes a value between −3 and 3. Therefore, we obtain $\mathbf{ct}''_i$, $\mathbf{ct}^c$, and IsZero_i by adding $4\Delta^2$ and then bootstrapping using **ACCPoly 9**.

output \ location	0	1	2	3	4	5	6	7
g_0 outputs ($\times\Delta$)	0	1	2	3	0	1	2	3
g_1 outputs ($\times\Delta^2$)	-2	-2	-2	-2	-1	-1	-1	-1
output \ location	8	9	10	11	12	13	14	15
g_0 outputs ($\times\Delta$)	0	1	2	3	0	1	2	3
g_1 outputs ($\times\Delta^2$)	0	0	0	0	1	1	1	1

output \ location	0	1	2	3	4	5	6	7
g_0 outputs ($\times\Delta$)	0	1	2	3	0	1	2	3
g_1 outputs ($\times\Delta$)	-1	-1	-1	-1	-1	-1	-1	-1
output \ location	8	9	10	11	12	13	14	15
g_0 outputs ($\times\Delta$)	0	1	2	3	0	1	2	3
g_1 outputs ($\times\Delta$)	1	1	1	1	1	1	1	1

output \ location	0	1	2	3	4	5	6	7
g_0 outputs ($\times\Delta$)	0	1	2	3	0	1	2	3
g_1 outputs ($\times\Delta^2$)	-1	-1	-1	-1	0	0	0	0
g_2 outputs ($\times\Delta$)	1	0	0	0	1	0	0	0

D. CarryMul

To implement and accelerate **CarryMul** in the (4,27,-511,511) floating-point FHE, carry functions are designed as follows: for the i-th coefficient of l-bit message $m = (m_{l-1} \dots m_0)_{(2)}$, set $\mathfrak{c}_{i\to i+1}(m\Delta^2) = (m_3m_2)_{(2)}$ and $\mathfrak{c}_{i\to i+2j}(m) = (m_{4j+3}m_{4j+2}m_{4j+1}m_{4j})_{(2)}$ for all $j \ge 1$. The remaining carry functions $\mathfrak{c}_{i\to i+2j+3}$ are set to zero function for all $j \in \mathbb{N}$. Then carry functions can be efficiently constructed by using **ACCPoly 10** as follows: If $l \le 4$, ciphertexts of message part and carry are obtained by using only one common bootstrapping. Otherwise if $l > 4$, **ACCPoly 10** is used to obtain four ciphertexts corresponding to four messages $(-1)^{m_3}(m_1m_0)_{(2)}\Delta$, $(-1)^{m_3}(m_2)_{(2)}\Delta$, $(-1)^{m_3}\Delta$, and $(-1)^{m_3}\Delta^2$, respectively. Note that if m_4 is not zero, all calculated ciphertexts have sign-reversed messages. However, the sign of the first and second ciphertexts is removed by doing tensor product with the third ciphertext, and those ciphertexts contain messages $(m_1m_0)_{(2)}$ and $(m_2)_{(2)}$, respectively. Finally, a ciphertext having the message $(m_3m_2)_{(2)}$ is calculated by adding Δ to the output g_2 and the ciphertext having the message $(m_2)_{(2)}$. Therefore, we can always decompose a message having a large bit into ciphertexts having a maximum two bit message.

output \ location	0	1	2	3	4	5	6	7
g_0 outputs ($\times\Delta$)	0	1	2	3	0	1	2	3
g_1 outputs ($\times\Delta$)	0	0	0	0	1	1	1	1
g_2 outputs ($\times\Delta$)	1	1	1	1	1	1	1	1
g_3 outputs ($\times\Delta^2$)	1	1	1	1	1	1	1	1

REFERENCES

[1] I. Chillotti, D. Ligier, J.-B. Orfila, and S. Tap, "Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2021, pp. 670–699.

[2] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Theory Comput.*, May 2009, pp. 169–178.

[3] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 1–36, Jul. 2014.

[4] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption over the torus," *J. Cryptol.*, vol. 33, no. 1, pp. 34–91, Jan. 2020.

[5] D. Micciancio and Y. Polyakov, "Bootstrapping in FHEW-like cryptosystems," in *Proc. 9th Workshop Encrypted Comput. Appl. Homomorphic Cryptogr.*, Nov. 2021, pp. 17–28.

[6] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for approximate homomorphic encryption," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2018, pp. 360–384.

[7] C. Gentry and S. Halevi, "Compressible FHE with applications to PIR," in *Proc. Theory Cryptogr. Conf.*, 2019, pp. 438–464.

[8] P. Kairouz et al., "Advances and open problems in federated learning," 2019, *arXiv:1912.04977*.

[9] S. Badrinarayanan, P. Miao, S. Raghuraman, and P. Rindal, "Multi-party threshold private set intersection with sublinear communication," in *Proc. IACR Int. Conf. Public-Key Cryptogr.*, 2021, pp. 349–379.

[10] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," in *Proc. Annu. Int. Cryptol. Conf.*, 2018, pp. 483–512.

[11] J. Neukirch, *Algebraic Number Theory*, vol. 322. Berlin, Germany: Springer, 2013.

[12] B. Hemenway, S. Lu, R. Ostrovsky, and W. Welser IV, "High-precision secure computation of satellite collision probabilities," in *Proc. Int. Conf. Secur. Cryptogr. Netw.*, 2016, pp. 169–187.

[13] A. Kumari, S. Tanwar, S. Tyagi, and N. Kumar, "Verification and validation techniques for streaming big data analytics in Internet of Things environment," *IET Netw.*, vol. 8, no. 3, pp. 155–163, May 2019.

[14] A. Jäschke and F. Armknecht, "Accelerating homomorphic computations on rational numbers," in *Proc. Int. Conf. Appl. Cryptography Netw. Secur.*, 2016, pp. 405–423.

[15] S. Moon and Y. Lee, "An efficient encrypted floating-point representation using HEAAN and TFHE," *Secur. Commun. Netw.*, vol. 2020, pp. 1–18, Mar. 2020.

[16] M. Capiński and P. E. Kopp, *Measure, Integral and Probability*. London, U.K.: Springer, 2004, vol. 14.

[17] P. Rigollet and J.-C. Hütter, "High dimensional statistics," *Lect. Notes Course 18S997*, vol. 813, no. 814, p. 46, 2015.

[18] B. M. Case, S. Gao, G. Hu, and Q. Xu, "Fully homomorphic encryption with k-bit arithmetic operations," *Cryptol. ePrint Arch.*, 2019. [Online]. Available: https://eprint.iacr.org/2019/521

[19] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, no. 6, pp. 1–40, Sep. 2009.

[20] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Homomorphic encryption standard," in *Protecting Privacy through Homomorphic Encryption*. Cham, Switzerland: Springer, 2021, pp. 31–62.

[21] N. Genise, D. Micciancio, and Y. Polyakov, "Building an efficient lattice gadget toolkit: Subgaussian sampling and more," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2019, pp. 655–684.

[22] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Proc. Annu. Cryptol. Conf.*, 2013, pp. 75–92.

[23] L. Ducas and D. Micciancio, "FHEW: Bootstrapping homomorphic encryption in less than a second," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2015, pp. 617–640.

[24] J.-M. Müller, N. Brisebarre, F. De Dinechin, C.-P. Jeannerod, V. Lefevre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of Floating-Point Arithmetic*, vol. 1. Boston, MA, USA: Springer, 2018.

[25] V. Rajaraman, "IEEE standard for floating point numbers," *Resonance*, vol. 21, no. 1, pp. 11–30, Jan. 2016.

[26] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-LWE and security for key dependent messages," in *Proc. Annu. Cryptol. Conf.*, 2011, pp. 505–524.

[27] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *J. Math. Cryptol.*, vol. 9, no. 3, pp. 169–203, Oct. 2015.

[28] J. H. Cheon, M. Hhan, S. Hong, and Y. Son, "A hybrid of dual and meet-in-the-middle attack on sparse and ternary secret LWE," *IEEE Access*, vol. 7, pp. 89497–89506, 2019.

[29] G. Fan, F. Zheng, L. Wan, L. Gao, Y. Zhao, J. Dong, Y. Song, Y. Wang, and J. Lin, "Towards faster fully homomorphic encryption implementation with integer and floating-point computing power of GPUs," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2023, pp. 798–808.

SEUNGHWAN LEE received the B.S. degree in electronic engineering from Hanyang University, Seoul, South Korea, in 2019, where he is currently pursuing the Ph.D. degree in electronic and computer engineering. His research interests include cryptography, signal processing, machine learning, and error-correcting code.

DONG-JOON SHIN (Senior Member, IEEE) received the B.S. degree in electronics engineering from Seoul National University, Seoul, South Korea, the M.S. degree in electrical engineering from Northwestern University, Evanston, IL, USA, and the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA. From 1999 to 2000, he was a member of the Technical Staff with the Wireless Network Division and the Satellite Network Division, Hughes Network Systems, Germantown, MD, USA. Since September 2000, he has been with the Department of Electronic Engineering, Hanyang University, Seoul. His current research interests include signal processing, error-correcting codes, sequences, discrete mathematics, machine learning, and cryptography.