Overflow-detectable Floating-point Fully Homomorphic Encryption

Seunghwan Lee and Dong-Joon Shin

Department of Electronic Engineering, Hanyang University,
Seoul , South Korea
{kr3951, djshin}@hanyang.ac.kr

Abstract. We propose a floating-point fully homomorphic encryption (FPFHE) based on torus fully homomorphic encryption equipped with programmable bootstrapping [15]. Specifically, FPFHE for 32-bit and 64-bit floating-point messages are implemented, the latter being state-of-the-art precision in FHEs. Also, a ciphertext is constructed to check if an overflow had occurred or not while evaluating arithmetic circuits with FPFHE, which is useful when the message space or arithmetic circuit is too complex to estimate a bound of outputs such as deep learning applications.

Keywords: Fully homomorphic encryption, homomorphic floating-point arithmetic, overflow detection, subgaussian error analysis

Table of Contents

1 Introduction

Since Gentry's seminal work on fully homomorphic encryption (FHE) [17], various FHEs such as BGV/FV [7], FHEW/TFHE [24, 13], and CKKS [11] have been proposed and intensively studied. Note that these schemes use integers, bits, or approximated complex numbers as their message sets. FHE is a powerful methodology for evaluating arithmetic functions while keeping the privacy of data. As applications of FHE with boolean circuit, deniable FHE [1], private information retrieval (PIR) [18, 22], private set intersection (PSI) [4], and homomorphic transpiler [20] have been studied. Also, homomorphically evaluating machine and deep learning models such as image classification [6] have been mostly studied by using CKKS. Since CKKS deals with a ciphertext packed with a large number of messages, circuits that can evaluate parallel with extensive data are suitable for CKKS. Moreover, no other FHE can process real messages with high precision except CKKS [1]. Therefore, CKKS is widely used to evaluate deep neural networks (DNNs).

However, in CKKS, errors from encoding floating-point numbers to the message space cannot be avoided. Since many DNNs learn with floating-point data and arithmetic, encoding in CKKS inevitably introduce a loss of accuracy. Without solving such encoding error problems, CKKS may not guarantee satisfactory results for the applications requiring complex and accurate results such as privacy-preserving generative models [5]. Therefore, floating-point FHE (FPFHE) is required for achieving equivalent results on plaintexts.

Another problem is an overflow occurrence such that evaluating a deep arithmetic circuit returns irrelevant results when a circuit output takes a value out of the message space. However, in contrast to evaluation with plaintexts, overflow is hard to detect in ciphertext domain. Such overflow frequently occurs if a value of circuit output is not bounded or an input dimension of a circuit is too large, then every input cannot be ensured whether an overflow occurs or not. Also, when the past data is used to update circuit parameters such as in privacy-preserving federated learning [22], then inaccurate updating using meaningless values due to overflow ruins the circuit performance. However, to the best of our knowledge, an overflow detection for FHE has not been proposed.

Contributions Our main contributions are divided into two parts. First, we propose a FPFHE, 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 much better precision than the state-of-the-art CKKS with 40-bit fixed-point precision [23]. Second, an effective overflow-detection (OD) method for the proposed FPFHE is constructed, which can check whether an overflow occurs or not during homomorphic operations. By combining these two schemes, we construct OD-FPFHE.

[1] [14] implements DNNs with 5-bit precision by using TFHE.

Also, we propose homomorphic algorithms to handle the following technical issues, which are important components of OD-FPFHE.

- **Sequential bootstrapping** FHE requires a bootstrapping algorithm for reducing the error after homomorphic operations. We extend the method in [15] to bootstrap a large number of messages bits by using integers modulo Q which is a number theoretic transform(NTT)-friendly. This algorithm enables the proposed FPFHE to bootstrap more message bits and to use NTT algorithm, which is one of the solutions for removing errors generated from fast Fourier transform (FFT), listed as an open problem in [15].
- **Accelerating deterministic gadget decomposition using modulus Q** This algorithm enables the proposed FPFHE to perform 64-bit integer operations when an integer modulus $Q = Q_0Q_1 > 2^{64}$ is chosen such that Q_0 and Q_1 are NTT-friendly primes.
- **Modified blind rotation for GINX-bootstrapping** This algorithm keeps the number of NTT operations constant for any secret key having finite support and hence improves running time compared to the state-of-the-art GINX-bootstrapping [24].
- **Error analysis without independent heuristic** This analysis is applicable even when deterministic gadget decomposition is used and enables choosing small lattice parameters for enhancing running time.
- **Various homomorphic algorithms** We propose homomorphic algorithms such as evaluating min and max, lifting a constant message to a monomial exponent, counting consecutive zeros from the most significant in the fraction of floating-point message until non-zero value occurs, and performing carry over after homomorphic operations. Note that these algorithms are run by using sequential bootstrapping.

Related works Several methods to implement FPFHE have been suggested [21, 25]. However, since these methods do not normalize results after homomorphic operations (See Section 2.6), it is possible that the operation error rapidly grows when consecutive homomorphic operations are performed. Moreover they suffer from slow operation time because they only add floating-point operations over the FHE schemes using gate operations such as TFHE and BGV/FV.

For error analysis, [10] proves that average-case error analysis in [13] does not require independent heuristic when randomized gadget decomposition is used and mean of output is zero. Since randomized gadget decomposition requires sampling from the kernel lattice, it is slow compared to deterministic gadget decomposition.

2 Preliminaries

This section introduces mathematical backgrounds and several fully homomorphic encryption schemes. The main reference and notation of algebraic and statistical background are followed [9], [28], and [31].

2.1 Notation

Let $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$, and $\mathbb{C}$ be a set of natural numbers, integer numbers, rational numbers, 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 will use $[n]$ to denote an index set $0, 1, ..., n-1$ for $n \in \mathbb{N}$. More generally, $[n_1, n_2, ..., n_m] = \prod_{i=1}^{m}[n_i]$ is used as a product index set for $n_1, ..., n_m \in \mathbb{N}$.

We will use notation $\boldsymbol{a} = (a_i)_{i\in[n]}$ as a vector notation and a_i as a i-th element of $\boldsymbol{a}$. Analogously for any polynomial $a(X) \in \mathbb{Z}[X]$, we will use a_i to denote the coefficient of X^i in $a(X)$. When a vector $\boldsymbol{a}(X) \in \prod_{i\in[n]} \mathbb{Z}[X]$ are given, $a_i(X)$ denots the i-th element of $\boldsymbol{a}(X)$ and $a_{i,j}$ is the j-th coefficient of the $a_i(X)$.

To measure the magnitude of element, we always use l_1 metric $|\cdot|$ for $x \in \mathbb{R}$. For any vector $\mathbf{x} \in \mathbb{R}^n$, $|\mathbf{x}|$ be a maximum value of $|x_i|$ and for all $i \in [n]$. If $x \in \mathbb{Z}_q$ is given, the $|x|$ is defined by choosing the representation $\bar{x}$ over $-q/2 \leq \bar{x} < q/2$ and evaluating $|\bar{x}|$. Analogously, if $x \in \mathbb{Z}_q[X]$ than $|x|$ is defined as maximum among $|x_i|$.

We represent a natural number $x \in \mathbb{N}$ by $x = (x_n x_{n-1} ... x_1 x_0)_{(\beta)}$ for a given $\beta \geq 2$ if

$$x = x_0 + x_1\beta + ... + x_n\beta^n,$$

where $0 \leq x_0, ..., x_n < \beta$ and β is called a radix.

2.2 Algebraic background

Let $\Phi_{2N}(X) \in \mathbb{Q}[X]$ be the cyclotomic polynomial with order $2N \in \mathbb{N}$. If N is a power of two, then $\Phi_{2N}(X)$ is known to be $X^N + 1 \in \mathbb{Q}[X]$. 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 product of $a(X), b(X) \in R_{N,q}$ is

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

having the property that a_i and b_i for all $i \in [N]$ participate only once for all in each coefficient of $a(X)b(X)$. This property is called *negacyclic property.*

If Q is a prime number satisfying $2N|(Q-1)$, then a field $\mathbb{Z}_Q$ splits $\Phi_{2N}(X)$ and has a primitive $2N$-th root of unity $\zeta \in \mathbb{Z}_Q$ such that by the Chinese remainder theorem(CRT), there is an isomorphism as follows:

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

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

More generally, if $Q = Q_0Q_1...Q_{n-1} \in \mathbb{N}$ with NTT-friendly primes Q_i are given, there exist isomorphisms being the inverse functions of each other as follows:

$$\begin{aligned}
\phi : \mathbb{Z}_Q \to \prod_{i\in[n]} \mathbb{Z}_{Q_i}, \qquad & a \mapsto \big(a \bmod Q_0, ..., a \bmod Q_{n-1}\big) \\
\phi^{-1} : \prod_{i\in[n]} \mathbb{Z}_{Q_i} \to \mathbb{Z}_Q, \qquad & \big(a_0, ..., a_{n-1}\big) \mapsto \Big(\sum_{i\in[n]} a_i Q_i^* \hat{Q}_i\Big) \bmod Q
\end{aligned} \tag{1}$$

where $Q_i^* = Q/Q_i$, and $\hat{Q}_i = Q_i^{*-1} \mod Q_i$. Then, we can find a primitive $2N$-th root of unity $\zeta_i \in R_{N,Q_i}$ for all $i \in [n]$ and the $\zeta = \phi^{-1}(\zeta_0, ..., \zeta_{n-1}) \in R_{N,Q}$ such that the following isomorphic structures are given as follows:

$$R_{N,Q} = \mathbb{Z}_Q[X]/(X^N+1) \cong \Big(\prod_{i\in[n]} \mathbb{Z}_{Q_i}\Big)[X]/(X^N+1) \cong \prod_{j\in[N]} \Big(\prod_{i\in[n]} \mathbb{Z}_{Q_i}[\zeta_i^{2j-1}]\Big). \tag{2}$$

The isomorphic structures (2) play an essential role in CKKS and BGV/FV, and a methodology to use (2) in TFHE have been an open problem [13]. Let NTT-friendly primes $(Q_0 = \nu 2^{\eta_0}+1, Q_1 = \nu 2^{\eta_1}+1)$ be denoted as *shared primes* when these share the same scaling factor $\nu \in \mathbb{N}$. In this paper, the product of *shared primes* is used as an integer modulus after Section 3.1.

2.3 Statistical background

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be an ambient probability space and $X : \Omega \to \mathbb{R}$ be an one-dimensional random variable. When the co-domain of a random variable is defined on a finite ring $\mathbb{Z}_Q$, we always define X as a function using $\Omega \to \{$-$\lfloor Q/2 \rfloor, ..., \lfloor Q/2 \rfloor$-1$\}$. We call random variable X is B-bounded if $|X| \le B$ almost surely. We explain subgaussian random variable and its properties.

Definition 1 ([9]). *A random variable X is 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 ([9]). *For any random variable $X \sim subG(\sigma)$, 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 from following 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 random variables in terms of minimizing variance, we will focus on conditions when (iii) holds without independence. Random variables $X_i \sim subG(\sigma_i)$ for $i \in [n]$ are called to have *Pythagorean additivity* if $\sum_{i\in[n]} X_i \sim subG((\sum_{i\in[n]} \sigma_i^2)^{1/2})$. Following conditions have been investigated.

Lemma 1 (Sum of dependent subgaussians [10]). *If* $(Z_i|Z_0, ..., Z_{i-1}) \sim subG(\sigma_i)$*,* $\mathbb{E}[Z_i] = 0$ *for all* $i \in [n]$*, and* σ_i *is free of* $Z_0, ..., Z_{i-1}$*, then* $Z_0, ..., Z_{n-1}$ *have Pythagorean additivity.*

Corollary 1 (From Lemma 1 [10]). *Let* $Y_i \sim subG(B_Y)$ *are mutually independent* B_Y*-bounded random variables for all* $i \in [n]$*, and let* $X_i \sim subG(B_X)$ *are* 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_0Y_0*,...,*$X_{n-1}Y_{n-1}$ *have Pythagorean additivity.*

Corollary 1 ensures that although all X_iY_i are dependent on each other, the analytical result of variance is the same as when X_iY_i are assumed to be mutually independent. This property is useful for analyzing a sum of large numbers of dependent random variables, e.g., analyzing an error bound after performing the bootstrapping in FHE. However, Corollary 1 requires both zero mean bounded random variables X_i, and Y_i since X_i and Y_i are subgaussian random variables. The same analytic result under more relaxed conditions is derived in Section 4.1.

2.4 LWE/MLWE symmetric encryption and gadget decomposition

In this section, widely used lattice-based encryption schemes are introduced. First, we recall the LWE symmetric encryption [30]. For any given natural number q, n, $t \in \mathbb{N}$, and t-bit message space $\mathbb{Z}_{2^t}$, a ciphertext with message $m \in \mathbb{Z}_{2^t}$ and symmetric key $\boldsymbol{s} = (-s_1, ..., -s_n, 1) \in \mathbb{Z}_q^{n+1}$ is obtained as follows:

$$\mathbf{ct}[m] \triangleq \big(\boldsymbol{a}, b = \sum_{i=1}^{n} a_i s_i + m\lfloor q/2^t \rceil + e\big)^T \in \mathbb{Z}_q^{(n+1)\times 1},$$

where T refers to the transposition of a vector or matrix, $\boldsymbol{a} \in \mathbb{Z}_q^n$ is chosen from a uniform distribution, e is sampled from a centered discrete Gaussian distribution χ_e on $\mathbb{Z}$ with standard deviation σ, and a secret key $\mathbf{s}$ sampled from a distribution χ_s. We adopt a ternary secret key $\boldsymbol{s}$, which is widely used for FHE [2]. In addition, secret key $\boldsymbol{s}$ for LWE encryption is called h-sparse if the number of non-zero element $\boldsymbol{s}$ is h.

The decryption function $\varphi_{\mathbf{s}}$ of $\mathbf{ct}$ is defined as a $\mathbb{Z}_q$-linear functional as follows:

$$\varphi_{\mathbf{s}}\Big(\mathbf{ct}\big[m\lfloor q/2^t \rceil\big]\Big) \triangleq b - \sum_{i=1}^{n} a_i s_i = m\lfloor q/2^t \rceil + e \in \mathbb{Z}_q.$$

Therefore, if $\lceil \log e \rceil \leq \lfloor \log q \rfloor - t$ is satisfied, then the message m is correctly extracted from $\mathrm{ct}[m]$ and in that case, we call $\mathbf{ct}$ as a *valid* ciphertext.

Based on $R_{N,Q}$, we can define module-LWE (MLWE) ciphertext as follows:

$$\mathrm{CT}\Big[m(X)\lfloor Q/2^t \rceil\Big] \triangleq \big(\boldsymbol{a}(X), b(X) = \sum_{i\in[K]} a_i(X)s_i(X) + m(X)\lfloor Q/2^t \rceil + e(X)\big)^T,$$

where $\mathrm{CT} \in R_{N,Q}^{(K+1)\times 1}$ and every coefficient of $s_i(X)$ and $e(X)$ is sampled from χ_e and χ_s, respectively. Security reduction of LWE and MLWE from shortest independent vector problem (SIVP) on a general lattice or structured Minkowski space (number field) [28], and general reductions between MLWEs have been researched [26].

Especially, following the tight reduction from MLWE to LWE are widely used in FHEs [13, 24]:

$$\begin{aligned}\mathbf{ct}[m_\alpha] \leftarrow & \mathbf{SampleExtract}(\mathrm{CT}[m(X)] \triangleq (a_1(X), ..., a_K(X), b(X)), \alpha) \\ \triangleq & (a'_{(0,0)}, ..., a'_{(1,N-1)}, a'_{(2,1)}, ..., a'_{(K-1,N-1)}, b_\alpha) \in \mathbb{Z}_Q^{KN+1}, \end{aligned} \quad (3)$$

where $a'_{(i,n)}{=}a_{i,(\alpha-n)}$ for $0 \le n \le \alpha$, $a'_{(i,n)}{=}{-}a_{(i,N-n)}$ for $\alpha < n \le N$, for all $i \in [K]$. Then, $\mathbf{ct}[m_\alpha]$ is a *valid* LWE ciphertext for the secret key $\boldsymbol{s} = (s_{(0,0)},...,s_{(0,N-1)},\ s_{(1,0)},...,s_{(K-1,N-1)})$ from the secret key $\boldsymbol{s}(X)$ of $\mathrm{CT}[m(X)]$.

In this paper, LWE and MLWE ciphertexts are denoted as **ct** and CT when the message is clear in context. In addition, for a given N, a ciphertext **ct** is called a *squashed* if the integer modulus of **ct** is $2N$. Note that a *squashed* **ct** is used to introduce FHEW / TFHE schemes in Section 2.5.

Next, we will review an approximated gadget for decomposing on MLWE ciphertext.

Definition 2 ([16]). *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 $\sum_i g_i \cdot x_i$ which satisfies $u - \sum_i g_i \cdot x_i = A$ for some gadget error $A \in \mathfrak{R}$ with $|A| \le \epsilon$, and $\max|x_i| \le \rho$.*

Proposition 2 (Deterministic (signed) gadget decomposition [13, 24]). *Assume that two finite additive group $R_{N,Q}$ and $\mathbb{Z}_Q$, $B \in \mathbb{N}$, and $\bar{l} \in \mathbb{N}$ are given such that $B^{\bar{l}-1} \le Q < B^{\bar{l}}$. Then, there exists a gadget $\boldsymbol{g} = (B^{\bar{l}-l}, ..., B^{\bar{l}-1})$ and deterministic gadget decompositions $\mathrm{G}_1^{-1} : R_{N,Q} \to R_{N,Q}^{1\times l}$ and $\mathrm{G}_2^{-1} : \mathbb{Z}_Q \to \mathbb{Z}_Q^{1\times l}$ with a quality $\rho = \lceil B/2 \rceil$ and a precision $\epsilon = \lceil B^{\bar{l}-l}/2 \rceil$.*

On the other hand, randomized gadget decomposition algorithms have been studied and used in the lattice-based signature as a trapdoor information [16]. Let a (complete) lattice $\mathcal{L} \subset \mathbb{R}^n$ be a finitely generated free $\mathbb{Z}$-module with rank n. For any gadget $\boldsymbol{g}$ from Proposition 2, the kernel lattice $\mathcal{L}^\perp$ is defined as follows:

$$\mathcal{L}^\perp \triangleq \left\{ v \in \mathbb{Z}^l : \sum_{i\in[l]} v_i g_i = 0 \right\}.$$

Since concrete basis of $\mathcal{L}^\perp$ having small norm values are well-known, algorithms of sampling $v \in \mathcal{L}^\perp$ having $|v| = O(B)$ and randomized gadget decomposition are well-studied. By using gadget decomposition, Gentry, Sahai, and Waters (GSW) [19] ciphertext and operation between GSW and MLWE ciphertexts are introduced in the next section.

2.5 Some fully homomorphic encryption for constructing OD-FPFHE: GSW, FHEW and TFHE

This section reviews the GSW cryptosystem, which is necessary for constructing bootstrapping algorithms. Let $I_n \in R_{N,Q}^{n\times n}$ be the identity matrix and $\otimes$ be the kronecker product. Then for given message $m \in \{0,1\}$, MLWE secret key $\boldsymbol{s}(X)$ and $R_{N,Q}$-gadget $\boldsymbol{g}$ from Proposition 2, GSW ciphertext $\mathrm{GCT} \in R_{N,Q}^{l(K+1)\times(K+1)}$ is defined as a matrix as follows:

$$\mathrm{GCT}[m] \triangleq \left(\mathrm{CT}\Big[mS_0(X)\Big]\Big|\mathrm{CT}\Big[mS_1(X)\Big]\Big|...\Big|\mathrm{CT}\Big[mS_{l(K+1)-1}(X)\Big]\right)^T,$$

where $\boldsymbol{S}(X) = (I_{K+1} \otimes \boldsymbol{g})\boldsymbol{s}(X)$.

For any two ciphertexts $\mathrm{CT}[m_1(X)]$ and $\mathrm{GCT}[m_2]$, external product $\boxtimes$ is defined as follows:

$$\boxtimes : \mathrm{CT} \times \mathrm{GCT} \to \mathrm{CT}, \qquad (\mathrm{CT}, \mathrm{GCT}) \mapsto \mathrm{G}^{-1}(\mathrm{CT})\mathrm{GCT}.$$

Decryption result of $\mathrm{GCT} \boxtimes \mathrm{CT}$ is already known as follows [13]:

$$\begin{aligned}&\varphi_{\boldsymbol{s}(X)}(\mathrm{CT} \boxtimes \mathrm{GCT})\\ &= m_2 m_1(X) + m_2 e'(X) + \sum_{i\in[l(K+1)]} e_i(X)\mathrm{G}^{-1}(\mathrm{CT})_i + \sum_{i\in[K+1]} s_i(X)A_i(X)\end{aligned} \tag{4}$$

where $\boldsymbol{e}(X) \in R_{N,Q}^{l(K+1)}$ is the noise contained in GCT, $e'(X)$ is the noise contained in CT, and $\boldsymbol{A}(X)$ is the gadget error. Since every norm value of $e_i(X)$ and $s_i(X)$ are small, validity of $\mathrm{GCT} \boxtimes \mathrm{CT}$ depends on ρ and ϵ of gadget decomposition.

Given LWE ciphertext $\mathbf{ct}$, bootstrapping algorithm of FHEW and TFHE executes the following contents. (i) the modulus of $\mathbf{ct}$ is reduced to a *valid squashed* $\mathbf{ct}'[m] = (a_1, ..., a_n, b)$ with $m = (m_{t-1}, , , m_0)_{(2)}2^v$ for $v = \log 2N - t$ and add a bias to b of $\mathbf{ct}'$ [24]; (ii) **BlindRotate** (See Algorithm 2 in Section 4.2) with public polynomial $\mathrm{ACCPoly}(X) \in R_{N,Q}$ is run and following MLWE ciphertext [2]

$$\mathrm{CT}\left[(-1)^{m_{t-1}}\mathrm{ACCPoly}(X)X^{-\varphi_{\mathbf{s}}(\mathbf{ct})}\right] \tag{5}$$

is obtained; (iii) *valid*, LWE ciphertext $\mathbf{ct}''$ with constant message of (5) is extracted by using **SampleExtract** defined in (3); (iv) secret-key of $\mathbf{ct}''$ is switched to secret-key of $\mathbf{ct}$;

The correctness of bootstrapping algorithm depends of validity of *squashed* $\mathbf{ct}'$ and having zero of most significant bit (MSB), i.e. $m_{t-1} = 0$. Recently, programmable bootstrapping (PBS) is proposed, which enables that server can evaluate a look-up table on ciphertext by programming coefficients of ACCPoly

[2] In this paper $-\varphi_{\mathbf{s}}(\mathbf{ct})$ is located on monomial degree in (5), instead of $\varphi_{\mathbf{s}}(\mathbf{ct})$ [13].

(X) [15]. In addition, the without padding PBS (WoP-PBS) algorithm is proposed that the above bootstrapping process is still correct when $m_{t-1} = 1$. In this paper, WoP-PBS is applied to a $R_{N,Q}$, where Q consisted of *shared primes* in Section 4.3.

2.6 Introduction to floating-point number systems

A floating-point number system can be defined by using four parameters as follows:

Definition 3 ([27]). *A (β, p, $e_{\min}$, $e_{\max}$) 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 $e_{\min} < e_{\max}$, such that every floating-point number $x \in \mathbb{R}$ has at least one representation (s, m, e) satisfying*

$$x = s \cdot m \cdot \beta^e, \tag{6}$$

where $s \in \{1, -1\}$ is the sign bit of x, m is an integer satisfying $0 \leq m < \beta$, and e is an integer satisfying $e_{min} \leq e \leq e_{max}$.

We call m and s as fraction and exponent in (6), respectively. Since Definition 3 does not guarantee uniqueness of floating-point representation (s, m, e), a unique representation, called *normal form*[3] as follows:

Definition 4 ([27]). *For the (β, p, $e_{\min}$, $e_{\max}$)-floating-point number system, (s, m, e) of $x \in \mathbb{R}$ is a normal form if $1 \leq m < \beta$ or if $e = e_{\min}$ with $0 \leq m < 1$.*

Intuitively, a fraction m can be expressed as $(m_{p-1}.m_{p-2}...m_1 m_0)_{(\beta)}$. If $x > \beta^{e_{\min}}$, then we can choose the unique fraction with $m_{p-1} > 0$. Otherwise, we uniquely choose the m with $e = e_{\min}$.

Let $\mathrm{RZ} : \mathbb{R} \to \mathbb{R}$ be a rounding function to a floating-point number such that $\mathrm{RZ}(x)$ rounds down if $x \geq 0$, rounds up otherwise [27]. Then following proposition are known.

Proposition 3 (Chapter 5 of [27]). *Let $\top \in \{+, -, \cdot, \backslash\}$ be the arithmetic operation. If $x, y \in \mathbb{R}$ with $\beta^{e_{\min}} \leq |x \top y| \leq (\beta - \beta^{1-p})\beta^{e_{max}}$ are given, following inequality holds*

$$|x \top y - \mathrm{RZ}(x \top y)| \leq \beta^{1-p} \min\big(x \top y, \mathrm{RZ}(x \top y)\big). \tag{7}$$

Proposition 3 guarantees an error bound after operation and rounding depending on two numbers when overflow and underflow do not occurs. However if *normal form* x and y are not chosen, rounding after p-digit loss a lot of precision and Proposition 3 is useless. For instance, take $x = 0.1 \cdot 10^1$ in $(10, 2, 0, 2)$ floating-point system. When $x^2 = 0.01 \cdot 10^2$ is rounded down on second digit, the result is not equal $\mathrm{RZ}(x^2) = x$ and its error is 1 larger than 2^{-1} calculated from (7).

[3] We do not distinguish the definition of *normal form* and *subnormal form* in [27].

The IEEE Standard [29] introduces $(2, 24, -2^7+2, 2^7-1)$ and $(2, 53, -2^{10}+2, 2^{10}-1)$ floating-point number systems, which are also called single and double precision, respectively. In this paper, analog of those number systems will be homomorphically implemented in Section 5 and simulated in Section 7.

3 Floating-point encryption and decryption

In this section, floating-point encryption and decryption are constructed, which will be used for constructing FPFHE.

3.1 Floating-point encoding and decoding

We propose encoding and decoding algorithm 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 define scaling factor $\Delta = Q_1$ and $\Delta' = \Delta 2^q \nu$ and β-evaluation map $\Psi_\beta : \mathbb{N}[X] \to \mathbb{Q}$, $a(X) \mapsto a(\beta)\beta^{1-p}$. Then for a given *normal form* (s, m, e) of floating point number x, **Encode** and **Decode** are defined as follows:

- **Encode(**s**,**$m = (m_{p-1}.m_{p-2}....m_0)_\beta$**,**$e$**)**
  - For the sign s, set a the polynomial $M^{\mathrm{s}}(X) \triangleq \Delta s$.
  - For the fraction m, set the fraction polynomial $M^{\mathrm{f}}(X) \triangleq \Delta \sum_{i=0}^{p-1} m_i X^i$.
  - For the exponent e, set the exponent polynomial $M^{\mathrm{e}}(X) \triangleq \Delta' e$.
  - Return $(M^{\mathrm{s}}(X), M^{\mathrm{f}}(X), M^{\mathrm{e}}(X))$.

- **Decode(**$M^{\mathrm{s}}(X)$, $M^{\mathrm{f}}(X)$, $M^{\mathrm{e}}(X)$**)**
  - Set $s = 1$ if $M^{\mathrm{s}}_0 > 0$, $s = -1$ otherwise.
  - Calculate $a(X) = \left\lfloor \left(M^{\mathrm{f}}(X) + \lfloor \Delta/2 \rceil\right)/\Delta \right\rceil$ and set $m = \Psi_\beta(a(X))$.
  - Set $e = \left\lfloor \left(M^{\mathrm{e}}_0 + \lfloor \Delta'/2 \rceil\right)/\Delta' \right\rceil$.
  - Return $s \cdot m \cdot \beta^e$.

Note that after analyzing bootstrapping error in Section 4.2, following facts are obtain: (i) The size of Q_1 controls bootstrapping error. (ii) Rounding error after doing tensor product of two MLWE ciphertexts are relatively small when Q_0 and Q_1 are *shared primes*. Moreover, the size of $Q_0 > Q_1$ is determined depending on β, p, and carry system, analyzed in Section 5.2. The q is also used to integer modulus of LWE ciphertext and *shared primes* are found by exhaustive search.

Since evaluation map Ψ_β is a homomorphism-like map as

$$\Psi_\beta\big(M^{\mathrm{f},1}(X) + M^{\mathrm{f},2}(X)\big) = \Psi_\beta\big(M^{\mathrm{f},1}(X)\big) + \Psi_\beta\big(M^{\mathrm{f},2}(X)\big),$$
$$\Psi_\beta\big(M^{\mathrm{f},1}(X) \cdot M^{\mathrm{f},2}(X)\big) = \Psi_\beta\big(M^{\mathrm{f},1}(X)\big) \cdot \Psi_\beta\big(M^{\mathrm{f},2}(X)\big)\beta^{1-p},$$

for given fraction polynomials $M^{\mathrm{f},1}(X)$ and $M^{\mathrm{f},2}(X)$, MLWE ciphertext having fraction message polynomial can perform leveled homomorphic operations. However, these relation is not hold on the quotient polynomial ring $R_{N,Q}$ when an one of coefficient or degree of result exceeds its integer or polynomial modulus. This problem will be resolved and fully homomorphic operations will be obtain in Section 5.

3.2 Floating-point encryption and decryption schemes

In this section, a floating-point encryption scheme is proposed. We adopt the state-of-the-art TFHE cryptosystem having tensor product [15] and change its torus modulus to *shared primes*. To product two MLWE ciphertexts, $K(K+1)/2$ evaluation keys having message $sk_i(X)sk_j(X)$ for some $0 \leq j \leq i \leq K-1$ and MLWE secret key $\boldsymbol{sk}$ are required.

For clear expression of those indexes, following index function is used. Let $\theta : \{(i,j) \in \mathbb{N}^2 | x \geq y\} \to \mathbb{N}$, $(i,j) \mapsto i(i+1)/2 + j$ be a bijection function which is counting indexes of lower triangular of a matrix from left-top first, and θ^{-1} be its inverse function and $\theta_1^{-1}, \theta_2^{-1}$ be coordinate functions of θ^{-1} such that $\theta^{-1}(x) = (\theta_1^{-1}(x), \theta_2^{-1}(x))$.

Given Q_0, Q_1, and q, proposed floating-point encryption scheme is given as follows:

- **Setup(1^λ)** Given security parameter λ, generate follow items:
  - Choose N_{gct} and K_{gct} for GSW, n and K_{ct} for MLWE, n for LWE ciphertext, and t for message space $\mathbb{Z}_{2^t}$ of LWE ciphertext. Choose h for sparsity of secret key.
  - Choose gadget parameters B_{bl}, B_{pack}, B_{ten}, and B_{ks} with $B_*/2$-quality and $B_*/2$-precision, $l_* = \lceil \log Q / \log B_* \rceil - 1$ for all $* \in \{\text{bl}, \text{pack}, \text{ten}, \text{ks}\}$.
- **KeyGen(1^λ)** Given security parameter λ, generate following keys:
  - Two ternary MLWE keys ***sk-bl***, ***sk*** and h-sparse LWE keys ***sk-ks***.
  - $\text{KS}_{i,j,k} = \mathbf{ct}[sk_{i,j} B_{\text{ks}}^{k+1}]$ by using ***sk-ks*** for all $(i,j,k) \in [K_{\text{ct}}, n, l_{\text{ks}}]$.
  - $\text{BL}_i^{\mathbf{1}}$=$\text{GCT}[m_i]$ and $\text{BL}_i^{\mathbf{-1}}$=$\text{GCT}[m_i']$ by using ***sk-bl*** for $i \in [n]$ where $(m_i, m_i') = (1,0)$ if $sk\text{-}ks_i = 1$, $(m_i, m_i') = (0,1)$ if $sk\text{-}ks_i = -1$, $(m_i, m_i') = (0,0)$ otherwise.
  - $\text{P}_{i,j,k} = \text{CT}[sk\text{-}bl_{i,j} B_{\text{pack}}^{k+1}]$ by using ***sk*** for all $(i,j,k) \in [K_{\text{gct}}, N_{\text{gct}}, l_{\text{pack}}]$.
  - $\text{Ten}_{i,j} = \text{CT}[sk_{i_1}(X) sk_{i_2}(X) B_{\text{ten}}^{k+1}]$ by using ***sk*** for all $(i,j) \in [(K_{\text{ct}} + 1)K_{\text{ct}}/2, l_{\text{ten}}]$, where $i_1 = \theta_1^{-1}(i)$ and $i_2 = \theta_2^{-1}(i)$.

  Set a public key as $ev = (\mathbf{P}, \mathbf{BL}^1, \mathbf{BL}^{-1}, \mathbf{KS}, \mathbf{Ten})$ and a secret key as ***sk***.
- **Enc$_{\boldsymbol{sk}}$**(x) :
  - Choose *normal form* (s, m, e) of x and run Encode(e, m, s).
  - Return ($\text{CT}_{\text{sign}}[M^{\text{s}}(X)]$, $\text{CT}_{\text{frac}}[M^{\text{f}}(X)]$, $\text{CT}_{\text{exp}}[M^{\text{e}}(X)]$) by using ***sk***.
- **Dec$_{\boldsymbol{sk}}$**$(\text{CT}_{\text{sign}}, \text{CT}_{\text{frac}}, \text{CT}_{\text{exp}})$:
  - Run the $\varphi_{\boldsymbol{sk}}$ for all inputs and get $M^{\text{s}}(X)$, $M^{\text{f}}(X)$, and $M^{\text{e}}(X)$.
  - Return Decode$(M^{\text{s}}(X), M^{\text{f}}(X), M^{\text{e}}(X))$.

The proposed encryption scheme is analogous to [15] however, we adopt h-sparse secret key for encryption key-switching key, which will be used to control errors after bootstrapping. Moreover when Q is larger than 2^{64}, G^{-1} suffers a slowdown due to using high precision integer arithmetic. Next section introduces accelerating gadget decomposition on such modulus Q.

Algorithm 1 $c \leftarrow \mathrm{G}_{\mathrm{crt}}^{-1}(\boldsymbol{d})$, accelerating gadget decomposition on CRT

Input: $Q_0 = \nu 2^{\eta_0} + 1$, $Q_1 = \nu 2^{\eta_1} + 1$, $\boldsymbol{d} = (a_1, b_1) \times ... \times (a_n, b_n) \in (\mathbb{Z}_{Q_0} \times \mathbb{Z}_{Q_1})^n$
Output: $c = (c_{0,0}, c_{0,1}, ..., c_{0,l-1}, c_{1,0}, ..., c_{1,l-1}, ..., c_{n-1,w}) \in \mathbb{Z}_Q^{1 \times nl}$

```
calculate Q̂_1 = Q_0^{-1} mod Q_1            ▷ Without loss of generality, Q_1 ≤ Q_0
for i ∈ [n] do
    x = (a_i − b_i)Q̂_1 mod Q_1
    y = x + b_i mod 2^{η_0}
    z = xν + ⌊(x + b_i)/2^{η_0}⌋
    c_i = (c_{i,0}, ..., c_{i,l−1}) := G^{−1}(y + 2^{η_0} z)    ▷ Guarantee that 0 ≤ y < 2^{η_0}
end for
return c = (c_1, ..., c_n)
```

3.3 Accelerating gadget decomposition on shared primes

Let gadget decomposition G^{-1} from Proposition 2 is given. To avoid using arithmetic operation larger than 64-bit, we save every element x as a CRT form as (1) and Algorithm 1, denoted as $\mathrm{G}_{\mathrm{crt}}^{-1}$, is used.

The correctness of $\mathrm{G}_{\mathrm{crt}}^{-1}$ is 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 $c = \phi^{-1}(a, b)$, there exists $x < Q_0$ such that $c = b + xQ_1 = (b + x) + (x\nu 2^{\eta_0})$. Since $Q_1 \le Q_0$, $\phi^{-1}(b, b) = b$, hence we obtain

$$xQ_1 = \phi^{-1}(a, b) - \phi^{-1}(b, b) = \phi^{-1}(a - b, 0) = (a - b)Q_1\hat{Q}_1 \mod Q,$$

which means $x = (a - b)\hat{Q}_1 \mod Q_0$. Then we can split $c \in \mathbb{Z}_Q$ into lower significant η_0-bit and more significant η_1-bit without calculating exact value of c, and run gadget decomposition twice, by using 64-bit integer operations.

4 Error analysis with deterministic gadget decomposition and sequential bootstrapping for FHE

In this section, we revisit error analysis of bootstrapping [15] and propose sequential bootstrapping on *shared primes*. Section 4.1 proves more generalized result of previous work [10], which is used to analyze error amplification without independent heuristic even when deterministic gadget decomposition is used.

Section 4.2 performs an error analysis for the product of two fraction ciphertexts following case, which is the worst case of constructing fully homomorphic operations in Section 5. Assume that *valid squashed* two ciphertexts $\mathbf{ct}_i^1$ and $\mathbf{ct}_i^2$ having i-digit fraction messages are given for all $i \in [p]$. Then 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 13 runs to pack outputs of **BlindRotate** into two MLWE ciphertexts CT_1 and CT_2; (iii) A **TensorProd** in Algorithm 14 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 3 runs to generate *squashed* LWE ciphertext from $(\mathbf{ct}'_i)_{i \in [p]}$.

If the results of above process are *valid* ciphertexts, a server can rerun this process a polynomial number of times. Note that this process is analogous to **PackedSumProducts** in [15], however we use **KeySwitch** to reduce ring and integer modulus of ciphertext after doing tensor product.

However after multiplying two fraction message polynomials having degree $p-1$, the $p-1$ coefficient can have a large message bit than 2^t. To solve this problem, Section 4.3 introduces a sequential bootstrapping.

4.1 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 a subgaussian. However, we will show that it is enough to require boundedness of one of X and Y and subgaussian others as follows:

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

Proof. By using uncorrelated property, we obtain $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y] = 0$. Since following inequality holds,

$$\mathbb{P}(|XY| > t) \leq \mathbb{P}(|Y| > t/B) \leq 2\exp\left(-\frac{t^2}{\sigma^2 B^2}\right), \tag{8}$$

it is known fact that (i) every k-th momentum is bounded as $\mathbb{E}[|XY|^k] \leq (2B^2\sigma^2)^{k/2}\ k\Gamma(k/2)$ where $\Gamma(\cdot)$ is a gamma function and (ii) a moment function is bounded by $M_{XY}(s) \leq \exp(4B^2\sigma^2 s^2)$ [31]. □

However, the factor $\sqrt{8}$ in Lemma 2 is undesirable and if we can use somewhat more information of Y is given, $\sqrt{8}$ is removed as follows:

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

Proof. By using independent property, we obtain $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y] = 0$. By Lemma 2, there exists a measurable function pointwisely larger than the moment function $M_{XY}(s)$ for all $s \in \mathbb{R}$. Then for any $s \in \mathbb{R}$,

$$\begin{aligned} M_{XY}(s) = \int_\Omega \mathrm{e}^{sXY} d\mathbb{P} &= \int_\Omega \lim_{m\to\infty} \sum_{n=1}^{m} \frac{(sXY)^n}{n!} d\mathbb{P} \overset{(a)}{=} \lim_{m\to\infty} \sum_{n=1}^{m} \int_\Omega \frac{(sXY)^n}{n!} d\mathbb{P} \\ &\overset{(b)}{\leq} \lim_{m\to\infty} \sum_{n=1}^{m} \int_\Omega \frac{(sB)^{2n} Y^{2n}}{2n!} d\mathbb{P} \overset{(c)}{=} M_{BY}(s) \overset{(d)}{\leq} \exp\left(\frac{(\sigma B)^2 s^2}{2}\right), \end{aligned} \tag{9}$$

where (a) holds from monotone convergence of measurable functions with boundedness of $M_{XY}(s)$ [9], (c) holds from the property that Y has a symmetric distribution, and (d) holds from the property of B-scaled subgaussian. In addition,

(b) holds since odd momentum of $\mathbb{E}[(XY)^{2n+1}] = \mathbb{E}[X^{2n+1}] \cdot \mathbb{E}[Y^{2n+1}]$ is zero for all $n \in \mathbb{N}$ by using independent property, and even momentum is bounded by Lebesgue integral property with $Y^{2n} \leq B^{2n}$ [9] for all $n \in \mathbb{N}$. □

Then, we can derive the following corollary based on Lemma 1 and 3.

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

Proof. By using Lemma 3, $Z_i = X_iY_i$ are $subG(\sigma B)$ random variables for all $i \in [n]$. Since Y_i is independent to all $Z_1, ..., Z_{i-1}$ and X_i is still B-bounded even if $Z_1, ..., Z_{i-1}$ are given, therefore X_0Y_0,...,$X_{n-1}Y_{n-1}$ have *Pythagorean additivity* by using Lemma 2. □

Note that Corollary 2 does not require X to have subgaussian property, meaning that $\mathbb{E}[X]$ can not be equal to zero compared to previous Corollary 1. Most of additional errors after running bootstrapping are formed of product of output of gadget decomposition and error in ciphertext or secret key. Since statistics of deterministic gadget decomposition relies on its input, many case of expectation of output is not zero, which is suitable for applying Corollary 2.

4.2 Error analysis: after running BlindRotate, Pack, TensorProd, and KeySwitch algorithms

In this section, four algorithms **BlindRotate**, **Packing**, **TensorProd**, and **KeySwitching** are run sequentially and error analysis for the output of each algorithms is performed. Since these algorithms have been widely studied in FHE with GINX-bootstrapping [13, 15], these algorithms are provided in Supplementary material except **BlindRotate** and **KeySwitch** which are modified to properly operate the proposed OD-FPFHE. Moreover, result of error analysis is given in this section and detailed proofs is provided in Supplementary material.

Analyzing BlindRotate First, we analyze **BlindRotate**. Assume that a *squashed* **ct** and ACCPoly(X) are given which determine the output message as in (5). First, the blind rotation in [24] is modified to Algorithm 2 and error analysis is performed as in Lemma 4. Note that **BlindRotate** returns 2^c LWE ciphertexts by using **SampleExtract** in (3) where c is pre-determined when **ct** is generated by running **KeySwitch** which is defined below.

The difference between Algorithm 2 and blind rotation in [24] is Line 3, where [24] runs with following equation ACC+ $= [(X^{a_i} - 1)\mathrm{ACC} \boxtimes \mathrm{BL}_i^1] + [(X^{-a_i} - 1)\mathrm{ACC} \boxtimes \mathrm{BL}_i^{-1}]$. To calculate external product $\boxtimes$, gadget decomposition should be run as $\mathrm{G}_{\mathrm{crt}}^{-1}((X^{a_i} - 1)\mathrm{ACC})$ and $\mathrm{G}_{\mathrm{crt}}^{-1}((X^{a_i} + 1)\mathrm{ACC})$. However, in Algorithm 2, only $\mathrm{G}_{\mathrm{crt}}^{-1}(\mathrm{ACC})$ calculation is required.

Algorithm 2 **out**←**BlindRotate**(**ct**, ACCPoly(X), ev)

Input: $\mathbf{ct} = (a_1, ..., a_n, b) \in \mathbb{Z}_{2N_{\text{gct}}}^{n+1}$, 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 := $(0(X), ..., 0(X), X^{-b}\text{ACCPoly}(X)) \in R_Q^{k+1}$
2: **for** $i \in [n]$ **do**
3: $\quad$ ACC+ $= (X^{a_i} - 1)[\text{ACC} \boxtimes \text{BL}_i^1] + (X^{-a_i} - 1)[\text{ACC} \boxtimes \text{BL}_i^{-1}]$
4: **end for**
5: **return** **out** $= (\mathbf{SampleExtract}(\text{ACC}, \alpha))_{\alpha\in[2^c]}$

Such difference gives more impact in the complexity when **ct** is encrypted by using secret key having a support larger than ternary case. Suppose that secret key having the support $\{d_0, ..., d_{m-1}\}$ and BL_i^j is encrypted with message 1 if $s_i = d_j$, and 0 otherwise for $(i, j) \in [n, m]$. Then Line 3 of **BlindRotate** is replaced by following operation

$$\text{ACC} \mathrel{+}= \sum_{j\in[m]} (X^{a_i d_j} - 1)[\text{ACC} \boxtimes \text{BL}_i^j]. \tag{10}$$

Note that $(X^{a_i d_j} - 1)$ and BL_i^j can be saved as NTT-transformed elements and assume that previous ACC is saved as the NTT-transformed element. Since only $\text{G}_{\text{crt}}^{-1}(\text{ACC})$ is used to perform $\boxtimes$ for all index j in (10), only a number of one inverse NTT and $(K_{\text{gct}} + 1)l_{bl}$ NTT operations are required which is free of support size, and up to double errors are added compared to previous result [24].

Next, an error amplification of **BlindRotate** is analyzed as follows:

Lemma 4. *Assume that* Algorithm 2 *runs with valid squashed* **ct***, and returns* $\text{out}_\alpha \in \mathbb{Z}_Q^{N_{gct}K_{gct}+1}$ *for the message* $(\text{ACCPoly}(X)\ X^{-\varphi(\mathbf{ct})})_\alpha$*. Then for any* $\alpha \in [N_{gct}]$*, the error* α*-coefficient error* $\mathcal{E}_{\text{bl}}^{(\alpha)}$ *of* out_α *is bounded except with probability* $2^{-\Omega(v)}$ *as follows:*

$$|\mathcal{E}_{\text{bl}}^{(\alpha)}| = O\left(B_{\text{bl}}\sqrt{vN_{\text{gct}}K_{\text{gct}}}\left(n + \sigma\sqrt{nl_{\text{bl}}}\right)\right). \tag{11}$$

Proof is listed in Supplementary material.

Analyzing Packing Second, we analyze **Packing** p ciphertexts obtained by **BlindRotate** are packed into one ciphertext by **Packing** which generates CT_{frac}. Since **Packing** has been widely used and studied in FHEs [11, 24, 13], it is provided as Algorithm 13 in Supplementary material. An error amplification of **Packing** after running **BlindRotate** is analyzed in Lemma 5.

Lemma 5. *Assume that* Algorithm 13 *runs with* p *ciphertexts* $(\mathbf{ct}_i[\Delta m_i])_{i\in[p]}$ *where* $\mathbf{ct}_i \in \mathbb{Z}_Q^{N_{\text{gct}}K_{\text{gct}}+1}$ *are generated by running* Algorithm 2 *with valid squashed*

$(\mathbf{ct}'_j)_{j\in[p]}$, *and returns a ciphertext* $\mathrm{OUT}[\Delta\sum_{i\in[p]} m_i X^i] \in R^{K_{\mathrm{ct}}+1}_{N_{\mathrm{ct}},Q}$. *Then for any coefficient* $\alpha \in [2p]$, *the* α*-coefficient error* $\mathcal{E}^{(\alpha)}_{Pack}$ *of* OUT *is bounded except with probability* $2^{-\Omega(v)}$ *as follows:*

$$|\mathcal{E}^{(\alpha)}_{\mathrm{pack}}| = |\mathcal{E}^{(\alpha)}_{\mathrm{bl}}| + O\Big(B_{\mathrm{pack}}\sqrt{vN_{\mathrm{gct}}K_{\mathrm{gct}}}(p+\sigma\sqrt{l_{\mathrm{pack}}p})\Big). \tag{12}$$

Proof is listed in Supplementary material.

Analyzing TensorProd Third, we analyze **TensorProd** Two ciphertexts packed from **Packing** are multiplied by using **TensorProd**. Since **TensorProd** have been widely used and studied in FHEs [11, 15], it is listed as Algorithm 14 in Supplementary material.

An error amplification of **TensorProd** after running **BlindRotate** and **Packing** is analyzed in Lemma 6.

Lemma 6. *Assume that* Algorithm 14 *runs with two ciphertexts* $\mathrm{CT}_1[m_1(X)]$ *and* $\mathrm{CT}_2[m_2(X)] \in R^{K_{\mathrm{ct}}+1}_{N_{\mathrm{ct}},Q}$ *which are generated by running* Algorithm 2 *and* 13 *with valid squashed* $(\mathbf{ct}_j)_{j\in[p]}$ *and* $(\mathbf{ct}'_j)_{j\in[p]}$, *and returns* $\mathrm{OUT}[\Delta^2 m_1(X)m_2(X)] \in R^{K_{\mathrm{ct}}+1}_{N_{\mathrm{ct}},Q}$. *If* $\Delta = \Omega(N_{\mathrm{ct}}|\mathcal{E}_{\mathrm{pack}}|)$ *is chosen and both coefficient of* $m_1(X)$ *and* $m_2(X)$ *are bounded by* $\Delta(\beta-1)$, *respectively, then for any* $\alpha \in [2p]$, *the* α*-coefficient error* $\mathcal{E}^{(\alpha)}_{Ten}$ *of* OUT *is bounded except with probability* $2^{-\Omega(v)}$ *as follows:*

$$|\mathcal{E}^{(\alpha)}_{\mathrm{ten}}| = O\Big(\Delta p\beta\Big|\mathcal{E}^{(p-1)}_{\mathrm{pack}}\Big| + K^2_{\mathrm{ct}}N^2_{\mathrm{ct}}l_{\mathrm{ten}}B_{\mathrm{ten}} + \sigma B_{\mathrm{ten}}K_{\mathrm{gct}}\sqrt{l_{\mathrm{ten}}N_{\mathrm{gct}}v}\Big) \tag{13}$$

Proof is listed in Supplementary material.

Analyzing KeySwitch Finally, we analyze **KeySwitch** After two ciphertexts are multiplied by **TensorProd** and **SampleExtract** for some coefficient $\alpha \in [2p]$, the result is *squashed* by **KeySwitch**. For a while, the message of *squashed* **ct** has the formed of $m\Delta^2$, where $m = (m_{t'-1}...m_1m_0)_{(2)}$ and $m_{f-1} = ... = m_1 = m_0 = 0$ for given $f \in [t']$. The goal of **KeySwitch** is to bootstrap the following $\boldsymbol{s}$ bits $(m_{f+s-1}...m_{f+1}...\ m_f)_{(2)}$ after $0's$ and to return 2^c outputs[15] after running **BlindRotate**. The above assumption will be resolved in next section.

Lemma 7. *Assume that* Algorithm 3 *runs with a ciphertext* $\mathbf{ct}[m\Delta^2]$ *generated by running* Algorithm 2, 13, 14, *and* **SampleExtract** *with valid squashed* $(\mathbf{ct}_j)_{j\in[p]}$ *and* $(\mathbf{ct}'_j)_{j\in[p]}$. *If message* $m = (m_{t'-1}...\ m_0)_{(2)}$ *with* $m_{f-1} = ... = m_0 = 0$ *is given and* $\Delta = \Omega(N_{\mathrm{ct}}|\mathcal{E}_{\mathrm{pack}}|)$, *then the error of* **out** *with message* $(m_{f+s}...m_s)_{(2)}2^{\log N_{\mathrm{gct}}-s}$ *is bounded except with probability* $2^{-\Omega(v)}$ *as*

$$O\left(\frac{|\mathcal{E}^{(p-1)}_{\mathrm{ten}}| + \Delta\nu p\beta^2}{\Delta\nu 2^{f+\eta_1+s-\log N_{\mathrm{gct}}}} + \frac{\sigma B_{\mathrm{ks}}\sqrt{l_{\mathrm{ks}}N_{\mathrm{ct}}K_{\mathrm{ct}}v}}{2^{q-1-\log N_{\mathrm{gct}}}} + \sqrt{hv}\right) \tag{14}$$

Algorithm 3 $\mathbf{out} \leftarrow$ **KeySwitch**$(\mathrm{CT}, f, s, c, ev)$

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

Output *squashed* $\mathbf{out}\left[(m_{f+s}...m_f)_{(2)} 2^{\log N_{\mathrm{gct}}-s}\right] \in \mathbb{Z}_{2N_{\mathrm{gct}}}^{n+1}$

1: $\mathbf{ct} \leftarrow \lfloor \mathbf{ct} \rceil_{Q \to Q_0 \approx \nu 2^{\eta_0}}$
2: Calculate the bias $= 2^{f+\eta_0-1}\nu$ and add it to b of $\mathbf{ct}$.
3: $\mathbf{ct} \leftarrow \lfloor \mathbf{ct}/\nu 2^{\eta_1+s+c+1+f-q} \rceil \mod 2^q$
4: Set $\mathbf{out} = (0, 0, ..., 0, b_{\mathrm{ct}}) \in \mathbb{Z}_{2^q}^{K_{\mathrm{ct}} N_{\mathrm{ct}}+1}$ ▷ where b_{ct} is b of $\mathbf{ct}$
5: **for** $(j, x) \in [K_{\mathrm{ct}}, N_{\mathrm{ct}}]$ **do**
6: $\quad v = \mathrm{G}^{-1}(a_{\mathrm{ct},j})$ ▷ where $a_{\mathrm{ct},j}$ is the j-coefficient of a of $\mathbf{ct}$
7: $\quad \mathbf{out} \mathrel{+}= \sum_k v_k \mathrm{KS}_{j,x,k}$
8: **end for**
9: **return** $\mathbf{out} \leftarrow \lfloor \mathbf{out}/2^{q-1-\log N_{\mathrm{gct}}} \rceil 2^c \mod 2N_{\mathrm{gct}}$

Proof is listed in Supplementary material.

If (14) is less than or equal $2^{\log N_{\mathrm{gct}}+1-t}$, output of **KeySwitch** is *valid* ciphertext and hence **BlindRotate** can be applied again. From now on, We will call that a proposed floating-point encryption is *valid* if parameters satisfy those inequality with $f = 0$, $c = 0$ and $s = t - 1$. Intuitively, *valid* floating-point encryption enables to bootstrap ciphertext with message $(m_{t-1}m_{t-2}...m_0)_{(2)}\varDelta^2$.

Moreover if *shared primes* are used, messages and approximated modulus $\nu 2^{\eta_0}$ after running Line 1 in **KeySwitch** share same scaling factor ν with negligible error amplification. This property enables to change from calculating denominator in Line 3 to shifting ν without extra message deformation.

In addition, if a non-sparse ternary secret is used key for encrypting $\mathrm{KS}_{j,x,k}$, the third error term of (14) becomes $O(\sqrt{nv})$. Although the first and second error terms of (14) can be reduced by increasing Q_1 and q, $O(\sqrt{nv})$ cannot be controlled hence makes an error floor. However if a sparse ternary secret key is used for $\mathrm{KS}_{j,x,k}$, this error term is controlled by a sparsity h.

4.3 Sequential bootstrapping for accommodating large numbers

In this section, we discuss a methodology to bootstrap a ciphertext for a large message bit based on the error analysis Lemma 4-7. We recall that WoP-PBS algorithm in [15] enables bootstrapping and moreover returns correct ciphertext even when MSB of message in a *squashed* ciphertext is one. Analogously, we will explain how to construct WoP-PBS on integer modulo product Q which is product of *shared primes* by using the following **ACCPoly 1**.

To instantiate WoP-PBS on *shared primes*, assume that a ciphertext $\mathbf{ct}[m\varDelta^2]$ with $m = (m_{t'}...m_1m_0)_{(2)}$ is given which is generated by running Algorithms 2 and 13, then extracted by using **SampleExtract**, and assume that the message space of LWE ciphertext is $\mathbb{Z}_{2^t} = \mathbb{Z}_{2^6}$. Although a *valid* floating-point encryption is given and Algorithm 3 runs with $f = 0$ and $\boldsymbol{s} = 4$, output of **BlindRotate** is sign-reversed by when $m_4 = 1$ in (5).

ACCPoly 1: Used to WoP-PBS, with $c = 1$, $s = 4$, and $f = 0$

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

Then the ACCPoly(X) constructed from **ACCPoly 1** is used to fix its sign. Let ot_1 and ot_2 be output functions of **ACCPoly 1** with the inputs $0, ..., 15$. Then, the the ACC initial polynomial $\mathrm{ACCPoly}(X) = \sum_{i \in [N_{\mathrm{gct}}]} (a_i + b_i) X^i$ is defined as follows: If i is even, then set $a_i = \mathrm{ot}_1\big(\lfloor i * 2^s / N_{\mathrm{gct}} \rfloor\big)\Delta$ and $b_i = 0$, otherwise, set $a_i = 0$ and $b_i = \mathrm{ot}_2\big(\lfloor i * 2^s / N_{\mathrm{gct}} \rfloor\big)\Delta = \Delta$.

After running **BlindRotate** with **ACCPoly 1**, we obtain two ciphertexts $\mathrm{CT}_1[(-1)^{m_4}\Delta(m_3 m_2 m_1 m_0)_{(2)}]$ and $\mathrm{CT}_2[(-1)^{m_4}\Delta]$. Finally, if four main algorithms run again with two ciphertexts and **ACCPoly 1**, *valid* ciphertext $\mathbf{CT}'[(m_3 m_2 m_1 m_0)_{(2)}\Delta]$. This is the application of WoP-PBS in our *shared primes*.

ACCPoly 2: Used to sequential bootstrapping, with $c = 1$, $\boldsymbol{s} = 4$, and $f = 0$

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

However to bootstrap next 4-bits $(m_7 m_6 m_5 m_4)_{(2)}$ message, lower significant 4-bits should be removed from $\mathbf{ct}[m\Delta^2]$. This can be done by replacing the second used **ACCPoly 1** above process to **ACCPoly 2**. Since error $|\mathcal{E}_{\mathrm{ten}}(X)|$ is Q_1 times larger than $|\mathcal{E}_{\mathrm{pack}}(X)|$ in Lemma 5 and 6, subtracting or adding ciphertexts generated from second output of **ACCPoly 2** from a **ct** still *valid* as many $\mathrm{poly}(\eta_1)$-times. Therefore after subtraction, we obtain $\mathbf{ct}'[\Delta(m_{t'} ... m_6 m_5)_{(2)}]$ hence **KeySwitch** can run with $\mathbf{ct}'$ correctly. Finally we proposes a sequential bootstrapping from above observation.

Theorem 1 (sequential bootstrapping, analogous of Lemma 5 of [15]). *Let valid floating-point encryption scheme with* LWE *message space* $\mathbb{Z}_{2^t}$ *for* $\boldsymbol{t} \geq 2$ *is given. Then for every message bits* m_i *of ciphertext* $\mathbf{ct}[(m_{t'-1} ... m_0)_{(2)}\Delta^2]$ *returned from running* Algorithm 2*,* 13*,* 14*, and* **SampleExtract** *and for any scaling* $\Delta'' \geq \Delta$*, a valid ciphertext* $\mathrm{CT}[m_i \Delta'']$ *can be generated.*

Although Theorem 1 is similar to Lemma 5 in [15], NTT algorithm can be used in the proposed floating-point encryption, and hence having a large messages are possible without generating extra noise contrast to using FFT.

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

5 Overflow-detectable floating-point FHE

In this section, we propose an overflow-detectable floating-point FHE. Section 5.1 proposes two homomorphic arithmetic operations **ADD** and **MULT**. Section 5.2 constructs various homomorphic algorithms which are important subroutines for **ADD** and **MULT**. Section 5.3 proposes a homomorphic normalization method for the floating-point outputs. Finally, Section 5.4 introduces a homomorphic algorithms for generating ciphertext with the message indicating overflow occurrence.

Due to Theorem 1, various floating-point homomorphic operations can be constructed and we implement (4,27,-511,511) and (4,12,-127,127) floating-point FHE as examples, which achieve double and single precision, respectively. Also, we choose LWE message space parameter $t = 6$ for bootstrapping each 5-bit of message by using WoP-PBS. After explaining each pseudo-code of homomorphic algorithms in Section 5.2, various ACC initial polynomials used to implement those algorithms are introduced with (4,27,-511,511) floating-point FHE and $t = 6$.

5.1 Overview of homomorphic operations for OD-FPFHE: addition, multiplication and overflow-detection

A homomorphic addition of two floating-point ciphertexts is proposed in Algorithm 4, denoted as **ADD**. Note that before adding two fraction ciphertexts, both exponents should be equal. Lines 1-5 show the process of equalizing both exponents. Also, subtraction can be easily constructed by replacing + to − in Line 6 of Algorithm 4.

Algorithm 4 : $(\mathrm{Out}_{\mathrm{frac}}, \mathrm{Out}_{\mathrm{sign}}, \mathrm{Out}_{\mathrm{exp}}, \mathrm{proof'}) \leftarrow$ **ADD**$(\mathrm{FCT}^1, \mathrm{FCT}^2, \mathrm{proof})$

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

The following operations are homomorphically performed. (*i*) **ADD** takes two floating-point ciphertexts, and MLWE ciphertext proof with the message indicating the overflow occurrence in the previous operations; (*ii*) The maximum of two exponents is calculated in Line 1; (*iii*) The differences between each exponent and max values are calculated and cut less than p in Line 3; (*iv*) The difference values are sign-reversed and lifted to the monomial exponent multiplied with its sign message by **ConstToExp**, and then the outputs are multiplied with each fraction message by **TensorProd** in Line 4; (*v*) **CarryAdd** bootstraps each coefficient to be less than the precision β and moves its carry to higher coefficients.

Algorithm 5 : $(\mathrm{Out}_{\mathrm{frac}}, \mathrm{Out}_{\mathrm{sign}}, \mathrm{Out}_{\mathrm{exp}}$,proof')$\leftarrow$ **MULT**$(\mathrm{FCT}^1, \mathrm{FCT}^2$,proof)

Input $\mathrm{FCT}^1 = (\mathrm{CT}^1_{\mathrm{frac}}, \mathrm{CT}^1_{\mathrm{sign}}, \mathrm{CT}^1_{\mathrm{exp}})$, $\mathrm{FCT}^2 = (\mathrm{CT}^2_{\mathrm{frac}}, \mathrm{CT}^2_{\mathrm{sign}}, \mathrm{CT}^2_{\mathrm{exp}})$, proof
1: $(\mathrm{Tmp}_{\mathrm{frac}}, (\mathrm{IsZero}_i)_{i\in[27]})$ = **CarryMul**(**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: $\mathrm{Out}_{\mathrm{sign}}$ = Bootstraps and **Packing** with **TensorProd**$(\mathrm{CT}^1_{\mathrm{sign}}, \mathrm{CT}^2_{\mathrm{sign}})$
4: proof' := **GenProof**$(\mathrm{Out}_{\mathrm{exp}}$, proof)
5: $(\mathrm{Out}_{\mathrm{frac}}, \mathrm{Out}_{\mathrm{exp}})$ = **Normal**$((\mathrm{IsZero}_i)_{i\in[p]}, \mathrm{FCT}^{\max}_{\mathrm{exp}}, \mathrm{Out}_{\mathrm{frac}})$
6: **return** $(\mathrm{Out}_{\mathrm{frac}}, \mathrm{Out}_{\mathrm{sign}}, \mathrm{Out}_{\mathrm{exp}}$, proof')

A homomorphic multiplication is proposed in Algorithm 5, denoted as **MULT**, which takes the homomorphic calculations. (*i*) **MULT** takes two floating-point ciphertexts, and proof; (*ii*) Fractions of two floating point numbers are multiplied by **TensorProd** in Line 1; (*iii*) **CarryMul** bootstraps each coefficient to be less than the precision β and moves its carry to higher coefficients in Line 1; (*iv*) Exponents of two floating point numbers are added in Line 2; (*v*) Signs of two floating point numbers are multiplied by **TensorProd** and bootstrapped in Line 3;

At the last part of both **ADD** and **MULT**, exponent is examined to check whether an overflow is occurs or not by **GenProof** which is explained in Section 5.4. In addition, outputs is changed into a *normal form* by **Normal** which is explained in Section 5.3. Next section will introduce homomorphic sub-algorithms for **ADD** and **MULT**.

5.2 Various homomorphic algorithms for ADD and MULT

We introduce sub-algorithms **Max**, **Min**, **ConstToExp**, **CarryAdd**, and **CarryMul**.

First, we propose **Max** as in Algorithm 6. The correctness is followed from the equation $\max(x, y) = \mathrm{ReLU}(x - y) + y$ where $\mathrm{ReLU}(x)$ return 0 if $x < 0$ and x otherwise. To examine the sign of message in the ciphertext $\mathrm{CT}^{(1)}_{\mathrm{exp}} - \mathrm{CT}^{(2)}_{\mathrm{exp}}$, we assume that both messages take values between $e_{\min}$ and $e_{\max}$ (Otherwise, overflow occurrence is already contained in proof ciphertext. See Section 5.4). Since the magnitude of message in $\mathrm{CT}^{(1)}_{\mathrm{exp}} - \mathrm{CT}^{(2)}_{\mathrm{exp}}$ is less than 2^e, we add a $2^e\Delta'$ and check whether m_e is still one or not by processing Line 6. Then, a ciphertext $\mathrm{CTtmp}_e[m_e\Delta]$ can mask other ciphertexts after running Line 7, which is same as of ReLU.

Algorithm 6 : Out$\leftarrow$ **Max**($\mathrm{CT}^{(1)}_{\mathrm{exp}}, \mathrm{CT}^{(2)}_{\mathrm{exp}}$)

1: $\mathrm{CT}_{\mathrm{exp}}[m_e m_{e-1}...m_{0(2)}\Delta'] \leftarrow \mathrm{CT}^{(1)}_{\mathrm{exp}} - \mathrm{CT}^{(2)}_{\mathrm{exp}} + 2^e\Delta'$ ▷ Where e is a smallest natural number satisfying $e_{\max} - e_{\min} < 2^e$.
2: **for** $i \in [e+1]$ bit of message m_i **do**
3: Generate MLWE ciphertext $\mathrm{CTtmp}_i[m_i\Delta]$ by using sequential bootstrapping
4: **end for**
5: **for** $i \in [e]$ **do**
6: Out += $\mathrm{CT}[m_e m_i 2^i \Delta'] \leftarrow$ Bootstrap **TensorProd**($\mathrm{CTtmp}_e, \mathrm{CTtmp}_i$)
7: **end for**
8: **return** Out $+ \mathrm{CT}^{(2)}_{\mathrm{exp}} - 2^e\Delta'$

To implement and accelerate **Max** in our (4,27,-511,511) floating-point FHE and $e = 10$, we apply two sequential bootstrapping to generate ciphertext having 4-bit messages in Line 2. Moreover, following **ACCPoly 3** are used once to bootstrap remaining two message bits in Line 2, and we obtain two ciphertexts having message $m_{10}(m_9 m_8 2^8 - 2^{10})\Delta'$ and $m_{10}\Delta$ at once. Note that **Min** can be implemented by equation $\min(x, y) = -\mathrm{ReLU}(x - y) + x$ with similar way.

ACCPoly 3 Used for checking whether 2^e is zero or not

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

Next, we propose a homomorphic algorithm lifting a constant message $m\Delta'$ to the monomial exponent message ΔX^m as Algorithm 7, denoted as **ConstToExp**, which is used for equalizing exponent values before addition and for normalizing after addition or multiplication. For the first case, **ConstToExp** returns ciphertext corresonding to message $m_s\Delta X^{-m}$ for given its sign message m_s, and the last case, returns ciphertext with message ΔX^m.

Algorithm 7 : Out$\leftarrow$ **ConstToExp**($\mathrm{CT}_{\mathrm{exp}}, \mathrm{CT}_{\mathrm{sign}}$)

Input $\mathrm{CT}_{\mathrm{exp}}[m\Delta']$ where $m = (m_e...m_0)_{(2)}$, (*optional*)$\mathrm{CT}_{\mathrm{sign}}[m_s\Delta]$
Output $\mathrm{Out}[\Delta X^m]$ where $\mathrm{Out}[m_s\Delta X^{-m}]$ can also be returned by packing with reversed index and multiply -1 when $i = e$ in Line 4.
1: Calculate ciphertexts $\mathrm{CT}_i[m_i\Delta]$ by using sequential bootstrapping and **Packing**
2: Out $\leftarrow \mathrm{CT}_{\mathrm{sign}}$ if $\mathrm{CT}_{\mathrm{sign}}$ is given, Out $= \Delta$ otherwise.
3: **for** $i \in [e+1]$ **do**
4: Out $\leftarrow$ **TensorProd**($(X^{2^i} - 1)\mathrm{Out}, \mathrm{CT}_i$) $+ \Delta\mathrm{Out}$
5: Out $\leftarrow$ Bootstraps Out for every index $j \in [2^i]$ and **Packing**
6: **end for**
7: **return** Out

The correctness of Algorithm 7 is analogous to the **BlindRotate**. Let message $m = (m_e m_{e-1} ... m_0)_{(2)}$ and $\mathrm{CT}_{\mathrm{sign}}[m_s \Delta]$ be given. When $i = 0$ in Line 4, Out is assigned with the message $\Delta^2 m_s((X-1)m_0+1) = \Delta^2 m_s X^{m_0}$. By inducting on i, Out is assigned with message $\Delta^2 m_s X^{m_i \cdots m_0 (2)}$ if the previous message of Out is $\Delta^2 m_s X^{m_{i-1} \cdots m_0 (2)}$. In addition, we already know that Out in Line 4 can be bootstrapped with sufficiently large Q_1 since addied error of Out is relatively small by Lemma 5 and 6.

To implement and accelerate **ConstToExp** in our (4,27,-511,511) floating-point FHE, **ACCPoly 4** and **5** are used for bootstrapping in Line 2. Since the message is cut and $m \leq p = 27 < 2^5$ is less than 5 bits, less significant 3 bits are sequentially bootstrapped by using **ACCPoly 4** and more significant 2 bits are bootstrapped by using **ACCPoly 5**. Note that a ciphertext $\mathrm{CT}[\Delta X^{32m_5+16m_4}]$ can be constructed by using **Packing** with **ACCPoly 5**. Then **ACCPoly 4** is used to process Line 3 when $i = 0, 1$, and 2 at once, and $\mathrm{CT}[\Delta X^{32m_5+16m_4}]$ is multiplied, which is desired result.

ACCPoly 4(Left) and **5**(Right). Used for splitting constant messages

output \ location	0	1	2	3	4	5	6	7
1st outputs ($\times\Delta$)	0	1	0	1	0	1	0	1
2st outputs ($\times\Delta$)	0	0	1	1	0	0	1	1
3st outputs ($\times\Delta$)	0	0	0	0	1	1	1	1
4st outputs ($\times\Delta$)	1	1	1	1	1	1	1	1

output \ location	0	1	2	3	4	5	6	7
1st outputs ($\times\Delta$)	1	0	0	0	-	-	-	-
2st outputs ($\times\Delta$)	0	1	0	0	-	-	-	-
3st outputs ($\times\Delta$)	0	0	1	0	-	-	-	-
4st outputs ($\times\Delta$)	0	0	0	1	-	-	-	-

Next, a homomorphic carry over algorithm for addition is proposed in Algorithm 8, denoted as **CarryAdd**, which is a core part to deal with the carries occurred in addition of two fractions. Let $\pi : \mathbb{Z} \to [\beta]$, $\pi(x) = x \mod \beta$ be a message-extraction function. After two ciphertext added, each message in coefficient should be adjusted by using π and remaining message, denoted as carry, should be added to high order coefficient.

Since defining carry function is not unique in general, we propose definition of abstract 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 $i < j$. Then we define carry collection $\mathfrak{C}_j$ from $j = 0$ recursively, and carry system $\mathfrak{C}$ of polynomial ring $\mathbb{Z}[X]$ as follows:

$$\mathfrak{C}_j : \mathbb{Z}[X] \to \mathbb{Z}, \quad \alpha(X) = \sum_{i=0}^{n} \alpha_i X^i \mapsto \left[\alpha_j + \sum_{i=0}^{j-1} (\mathfrak{c}_{i \to j} \circ \mathfrak{C}_i)(\alpha(X))\right], \quad \forall j = 1, 2, ...$$

$$\mathfrak{C} : \mathbb{Z}[X] \to \mathbb{Z}[X], \quad \alpha(X) = \sum_{j=0}^{n} \alpha_j X^j \mapsto \sum_{j=0}^{n} (\pi \circ \mathfrak{C}_j)(\alpha(X)) X^j, \tag{15}$$

where $\mathfrak{C}_0\left(\sum_j \alpha_j X^j\right) = \alpha_0$. Intuitively, the carry collection $\mathfrak{C}_j$ adds every carry $\mathfrak{c}_{i \to j}$ from the coefficient $i < j$ to the j-coefficient α_j. In addition, we call $\mathfrak{C}$ is a *valid* carry system if $\varphi_\beta(\alpha(X)) = (\varphi_\beta \circ \mathfrak{C})(\alpha(X)) \in \mathbb{Q}$ for all $\alpha(X) \in$

$\mathbb{Z}[X]$, i.e., sharing the same values when evaluating β. For **ADD**, carry functions $\mathfrak{c}_{i\to i+1}(x) = (x - \pi(x))/\beta$ are used for all $i \in \mathbb{N}$.

Algorithm 8 $(\mathrm{CT}'_{\mathrm{frac}}, \mathrm{CT}'_{\mathrm{sign}}, (\mathrm{IsZero}_i)_{i\in[p]}) \leftarrow$ **CarryAdd**$(\mathrm{CT}_{\mathrm{frac}}[m(X)\Delta^2])$

1: Set $\mathbf{ct}^c = 0$ for $\mathbf{ct}^c \in \mathbb{Z}_Q^{K_{\mathrm{ct}}N_{\mathrm{ct}}+1}$ and $\mathrm{CT}_{\mathrm{frac}} \leftarrow \mathrm{CT}_{\mathrm{frac}}X^1$
2: **for** $i \in [p+2]$ **do**
3: $\quad \mathrm{Tmp}[\Delta^2\mathfrak{C}_i(m(X))] \leftarrow \mathbf{SampleExtract}(\mathrm{CT}_{\mathrm{frac}}, i) + \mathbf{ct}^c$
4: $\quad \mathbf{ct}'_i\big[\Delta(\pi\circ\mathfrak{C}_i)(m(X))\big], \mathbf{ct}^c\big[\Delta^2(\mathfrak{c}_{i\to i+1}\circ\mathfrak{C}_i)(m(X))\big] \leftarrow$ Bootstraps Tmp
5: **end for**
6: $\mathrm{CT}'_{\mathrm{sign}} \leftarrow$ sequential bootstrap with CT^c and extract its sign.
7: $\mathrm{CT}' \leftarrow$ **Packing** with $\mathbf{ct}'_0, ..., \mathbf{ct}'_{p+1}$ and **TensorProd** with $\mathrm{CT}'_{\mathrm{sign}}$
8: **for** $i \in [p+2]$ **do**
9: $\quad (\mathbf{ct}''_i, \mathbf{ct}^c, \mathrm{IsZero}_i) \leftarrow$ Bootstrap $\mathbf{SampleExtract}(\mathrm{CT}', i) + \mathbf{ct}^c$ ▷ where IsZero_i has a message $m\Delta$ with $m = 1$ if message of $\mathbf{ct}''_i$ is zero, and $m = 0$ otherwise.
10: **end for**
11: **return** $\Big(\mathrm{CT}'_{\mathrm{frac}} \leftarrow \mathbf{Packing}\big((\mathbf{ct}''_i)_{i\in[p+2]}\big), \mathrm{CT}'_{\mathrm{sign}}, (\mathrm{IsZero}_i)_{i\in[p+1]}\Big)$

The Correctness of **CarryAdd** is as follows: After Algorithm 8 runs every iteration on Line 2, the sign of carry message $\mathbf{ct}^c$ is the sign of addition of two ciphertexts. However if the sign is negative, packed messages from $\mathbf{ct}'_0, ..., \mathbf{ct}'_{p+1}$ becomes sign-reversed. To fix its sign, calculated sign in Line 6 is multiplied to CT' in Line 7 and bootstrap again.

In addition, **ADD** checks whether each coefficient of $\mathfrak{C}(m(X))$ is zero or not and generates a ciphertext IsZero_i containing boolean message above information. This ciphertext is used to calculate *normal form*.

ACCPoly 6-8. Used for generating carry and IsZero ciphertexts in Algorithm 8

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

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

output \ location (adding $4\Delta^2$)	0	1	2	3	4	5	6	7
1st outputs ($\times\Delta$)	0	1	2	3	0	1	2	3
2nd outputs ($\times\Delta^2$)	-1	-1	-1	-1	0	0	0	0
3st outputs ($\times\Delta$)	1	0	0	0	1	0	0	0

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

Similar to **CarryAdd**, homomorphic carry over for multiplication is proposed Algorithm 9, denoted as **CarryMul** with *valid* $\mathfrak{C}$ with carry functions $\mathfrak{c}_{i\to j}$.

Algorithm 9 $(\mathrm{CT}'_{\mathrm{frac}}, (\mathrm{IsZero}_i)_{i\in[p]}) \leftarrow$ **CarryMul**$(\mathrm{CT}_{\mathrm{frac}}[m(X)\Delta^2])$

1: $\mathbf{ct}_i^c = 0$, where $\mathbf{ct}_i^c \in \mathbb{Z}_Q^{K_{\mathrm{ct}}N_{\mathrm{ct}}+1}$, for all $i \in [2p]$
2: **for** $i \in [2p]$ **do**
3: $\quad \left(\mathbf{ct}_i'\left[\Delta(\pi \circ \mathfrak{C}_i)(m(X))\right], \left(\mathbf{ct}_j^{cc}\left[\Delta^2(\mathfrak{c}_{i\to j} \circ \mathfrak{C}_i)(m(X))\right]\right)_j, \mathrm{IsZero}_i\right) \leftarrow$ Sequential bootstrap with **SampleExtract**$(\mathrm{CT}_{\mathrm{frac}}, i) + \mathbf{ct}_i^c$
4: $\quad$ Update carry $\mathbf{ct}_j^c \mathrel{+}= \mathbf{ct}_j^{cc}$ for all $j > i$ of generated carry ciphertexts
5: **end for**
6: **return** (**Packing**$(\mathbf{ct}_i')_{i\in[2p]}, (\mathrm{IsZero}_i)_{i\in[2p]}$)

To implement and accelerate **CarryMul** in our (4,27,-511,511) floating-point FHE, carry functions are designed as follows: for any given i-th coefficient of l-bit message $m = (m_{l-1}...m_0)_{(2)}$, 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 $j \geq 1$. Then carry functions are constructed efficiently by using **ACCPoly 9** as follows: If $l \leq 4$, then ciphertexts of messages and carry are obtained by using sequential bootstrapping ones. Otherwise, **ACCPoly 9** is used and obtains four ciphertexts having messages $(-1)^{m_3}(m_1m_0)_{(2)}\Delta$, $(-1)^{m_3}(m_2)_{(2)}\Delta$, $(-1)^{m_3}\Delta$, and $(-1)^{m_3}\Delta^2$. Then sign of two first and second ciphertexts can be removed by doing tensor product with third ciphertext. Finally, we obtain a ciphertext with message $(m_3m_2)_{(2)}\Delta^2$ by adding the sign-fixed second ciphertext multiplied with Δ, forth ciphertext, and Δ^2.

ACCPoly 9. Used for generating carry in Algorithm 9

output \ location	0	1	2	3	4	5	6	7
1st outputs ($\times\Delta$)	0	1	2	3	0	1	2	3
2nd outputs ($\times\Delta$)	0	0	0	0	1	1	1	1
3rd outputs ($\times\Delta$)	1	1	1	1	1	1	1	1
4th outputs ($\times\Delta^2$)	1	1	1	1	1	1	1	1

For accelerating **CarryMul**, upper-bound of $\mathfrak{C}_i(m_1(X)m_2(X))$ can be analyzed for any *valid* fraction message polynomials $m_1(X)$ and $m_2(X)$ by following Proposition 4.

Proposition 4. *Suppose that two polynomials $a(X), b(X) \in \mathbb{N}[X]$ are given with each coefficient satisfy $a_i \le b_i$ and carry functions $\mathfrak{c}_{i\to j} : \mathbb{N} \to \mathbb{N}$ are given for all $i, j \in \mathbb{N}$ satisfying $\mathfrak{c}_{i\to j}(x) \le \mathfrak{c}_{i\to j}(y)$ if $x \le y$ for all $x, y \in \mathbb{N}$. Then following inequality $\mathfrak{C}_j(a(X)\alpha(X)) \le \mathfrak{C}_j(b(X)\alpha(X))$ holds for all j-coefficient and $\alpha(X) \in \mathbb{N}[X]$.*

Proof. Since every coefficient of $b(X)$ is greater than or equal $a(X)$, following equation $\mathfrak{C}_0(a(X)\alpha(X)) = a_0\alpha_0 \le b_0\alpha_0 \le \mathfrak{C}_0(b(X)\alpha(X))$ holds. To induct on j, let assume $\mathfrak{C}_0, ..., \mathfrak{C}_{j-1}$ satisfies Proposition 4. Then for every index $j \in \mathbb{N}$,

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

□

Therefore, $\mathfrak{C}_i(m_1(X)m_2(X)) \le \mathfrak{C}_i(m_1(X)m_{\max}(X)) \le \mathfrak{C}_i(m_{\max}^2(X))$ is upper bound for any product of two message polynomial by using Proposition 4 with polynomial $m_{\max}(X) = (\beta - 1)\Delta \sum_{j=0}^{p-1} X^j$.

From Proposition 4, we obtain condition for Q_0 such that $2^{\eta_0-\eta_1} > \log \max_i \mathfrak{C}_i(m_{\max}^2(X))$. Otherwise, messages are deformed due to small Q_0. Moreover, $\max_i \mathfrak{C}_i(m_{\max}^2(X))$ can be pre-calculated and it is upper-bounded by 2^{10} for our carry system and selected parameters. Therefore, index i is enough to run from $p-4$ to $2p-1$ in Line 2 since every index $i \le p-5$ cannot influence index $p-1$.

5.3 Algorithm for normalizing after homomorphic floating-point operations

When **CarryAdd** or **CarryMul** are ended, fraction and exponent should be adjusted to a *normal form*. The first step is counting the number of zeros from most significant in fraction and stopping when nonzero values are occurred. We propose **HomeCount** as Algorithm 10 as follows:

When **HomCount** runs on the Line 4, product of CT_1 and IsZero_i acts like AND gate. Therefore, every returned ciphertexts CT_1 in Line 4 encrypt Δ until IsZero_i encrypts 0 for the first index i, and encrypt 0 after i-iteration. Then, **HomCount** adds all returned ciphertext CT_2 to Out in Line 5, which has message scaled by Δ^2. Finally, we bootstrap Out and obtains ciphertext with message having a number of zeros until nonzero significant occurs.

By using Algorithm **HomCount**, we proposes Algorithm 11, denoted as **Normalize** to normalize fraction and exponent on the ciphertext. Since the number of nonzero significant fraction are calculated by using **HomCount**,

Algorithm 10 : Out← **HomCount**($(\text{IsZero})_{i\in[p']}$)

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** $i \in [p'-1]$ **do**
4: $\quad (\text{CT}_1[\Delta \prod_{j=i}^{p'-1} m_j], \text{CT}_2[\Delta^2 \prod_{j=i}^{p'-1} m_j]) \leftarrow$ Sequential bootstrap and **Packing** with **TensorProd**$(\text{CT}_1, \text{IsZero}_i)$
5: $\quad$ Out += CT_2
6: **end for**
7: **return** $\text{Out}[m\Delta'] \leftarrow$ Bootstrap Out, where m is a number of zeros until nonzero significant occurs

Normalize can subtracts it from exponent ciphertext. However subtracted messages can be less than $e_{\min}$, therefore **Normalize** evaluates **Min** as in Line 1 and subtracts its output from exponent ciphertext. In addition, the min ciphertext having constant message converts to the CTtmp having message in monomial exponent by using **ConstToExp** in Line 2. Then fraction can be adjusted to *normal form* by doing tensor product with CTtmp and CT_{frac}.

Algorithm 11 $\text{Out}_{\text{frac}}, \text{Out}_{\text{exp}} \leftarrow$ **Normalize**$((\text{IsZero}_i)_{i\in[p]}, \text{CT}_{\text{exp}}, \text{CT}_{\text{frac}})$

1: $\text{CT}_{\text{exp}}^{\min} =$ **Min**(**HomCounter**$((\text{IsZero}_i)_{i\in[p]})$, CT_{exp})
2: CTtmp = **ConstToExp**$(\text{CT}_{\text{exp}}^{\min})$
3: $\text{Out}_{\text{frac}} \leftarrow$ **SampleExtract**$(\cdot, i)$, Bootstrap and **Packing** from **TensorProd**(CTtmp, CT_{frac}) for all $i \in [p]$
4: **return** $(\text{Out}_{\text{frac}}, \text{CT}_{\text{exp}} - \text{CT}_{\text{exp}}^{\min})$

5.4 Generating a proof to detect overflow occurrence

In this section, we propose an algorithm to generate ciphertext having message of overflow occurrence. Since message space $\mathcal{M} \subseteq \mathbb{R}$ for encrypting real numbers is finite in practice, unique maximum and minimum norm values $|x|$ for $x \in \mathcal{M}\backslash\{0\}$ exist. Let $U_{\mathcal{M}}$ and $L_{\mathcal{M}}$ be the maximum and minimum norm values, respectively. Assume that finite n messages $x_i \in \mathcal{M}$ for $i \in [n]$ and bounded depth arithmetic circuit $f : \mathcal{M}^n \to \mathbb{R}$ are given. $\boldsymbol{x}$ are called f-overflow numbers if $|f(x)| > U_{\mathcal{M}}$ and if a norm value of any intermediate result is greater than $U_{\mathcal{M}}$ while evaluating f, it is called that an overflow occurs.

It is clear that CKKS has f-overflow numbers for any circuit f due to the finite message space in $\mathbb{C}$. Note that original BGV/FV do not show f-overflow numbers because is uses a message space with t-characteristic ring for some $t \in \mathbb{N}$. However, if BGV/FV are used to encrypt a subset of $\mathbb{Z}$, then f-overflow numbers exist.

Since the messages of calculated ciphertext cannot be checked during homomorphic operations, overflow occurrence has to be informed to a user by generating extra ciphertext having such information as a message. Generation of those for CKKS and BGV/FV is a complicated problem because fixed-point operations are used and it may require a lot of extra precision to save and check overflowed-results. In FPFHE, however, inspection of the exponent is enough to check overflow and just extra one bit precision in exponent is required. We propose Algorithm 12, denoted as **GenProof** is proposed, which uses $e' = \lfloor \log \max(|e_{\max} - 1|, |e_{\max} - 2e_{\min} + 1|) \rfloor + 1$ to generate MLWE ciphertext for the message indicating whether the message of $\mathrm{CT}_{\exp}$ is larger than $e_{\max}$ or not and this ciphertext is called a proof.

Algorithm 12 $\mathrm{proof}' \leftarrow$ **GenProof**$(\mathrm{CT}_{\exp}, \mathrm{proof})$

Input $\mathrm{CT}_{\exp}[m\Delta']$, $\mathrm{proof}[m_{pf}\Delta']$
Output $\mathrm{proof}[(m_{pf} + \alpha)\Delta']$ with $\alpha = 1$ if $m > e_{\max}$, and $\alpha = 0$ otherwise
1: $\mathrm{CT}[(m_{e'-1}...m_0)_{(2)}\Delta'] = \mathrm{CT}_{\exp} + (2^{e'} - e_{\max} - 1)\Delta'$
2: $\mathrm{proof}'[\alpha\Delta'] \leftarrow$ Use sequential bootstrap to have a message $\alpha = 1 - m_{e'-1}$
3: **return** $\mathrm{proof}' \leftarrow \mathrm{proof} + \mathrm{proof}'$

GenProof is analogous to the **Max** which operates as follows: If the previous proof has a message 0, i.e., an overflow does not occur while performing the previous operations, then the message m is in $2e_{\min} \leq m \leq 2e_{\max}$. Therefore, $m' = e_{\max} - m + 1$ is strict positive if and only if $m \leq e_{\max}$. Moreover, e'-bit from binary representation of $2^{e'} - m'$ is one if and only if m' is strict positive, meaning that proof' in Line 2 has a message of whether $m > e_{\max}$ or not.

Otherwise if the previous proof has a non-zero message, then it already contains the information of overflow occurrence. Then, by returning the proof that is the sum of all previous proofs, a user can check whether an overflow occurs or not by decrypting the proof. Therefore, by using the proposed OD-FPFHE and given bootstrapping failure probability $2^{-\Omega(v)}$, a user can detect f-overflow numbers for any $\mathrm{poly}(v)$ bounded function f.

6 Security analysis

This paper relies on key-dependent message (KDM) and circular security assumption to generate public keys [8, 17, 19] which is used for FHEs. 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 $2^{0.292k+16.4}$ [3]. In addition, we apply hybrid primal and dual attack [12] to LWE key-switching key encrypted by h-sparse ***sk-ks***. Such derived concrete parameters are listed in Table 1.

In Table 1 the number after D and S refers to the security level. For instance, parameters D128 guarantees 128-bit security for Primal, Dual, and hybrid attacks. D and S refer to the double and signal precision of OD-FPFHE, respectively. Note that, OD-FPFHE with D128 can deal with the ciphertexts for double and single precision messages, but however OD-FPFHE with S128 can deal with the ciphertexts for single precision messages only.

Table 1: Concrete parameters of OD-FPFHE for various security levels

\	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	785	$521\cdot 2^{39}$	$521\cdot 2^{29}$	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
S128	2^8	13	2^{12}	1	785	$135\cdot 2^{36}$	$135\cdot 2^{30}$	2^{21}	2^{12}	2^{16}	2^{18}	2	131
S160	2^8	15	2^{12}	1	1089	$135\cdot 2^{36}$	$135\cdot 2^{30}$	2^{21}	2^{12}	2^{16}	2^{18}	2	131
S192	2^8	18	2^{11}	3	1292	$135\cdot 2^{36}$	$135\cdot 2^{30}$	2^{22}	2^{10}	2^{14}	2^{18}	2	160

7 Simulation results and conclusions

We implement (4,27,-511,511) and (4,12,-127,127) floating-point number system by using PALISADE v1.11. 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 were applied by the compiler) and WITH-INTEL-HEXL= ON (AVX-512 acceleration was used), by Clang++10.0.0. Running times for various parameters are listed in Table 2. Since S128, S160, and S190 does not support double precision, the time consumption is not available for these cases.

We simulate Algorithms 4 and 5 by using the number of 1 (single-core), 4, and 10 threads using parameters in Table 1, and list the operation time in Table 2. In addition, we simulate addition and multiplication time per threads, which is listed in Table 2 as Amortized time. Therefore, if many thread are available, run time is expected to approach 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 approach to the Time (10 thread) in Table 2.

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

$$x_d^1 = \text{-}9.1763514236254290 * 10^{-32},\ x_d^2 = 6.2467247246375865 * 10^{-24},$$
$$x_d^3 = 2.4523526872362373 * 10^{22},\ x_d^4 = \text{-}5.4324663335297274 * 10^{17},$$
$$x_f^1 = \text{-}2.7914999921796382 * 10^{-15},\ x_f^2 = 8.3867001884896375 * 10^{-12},$$

Table 2: Time consumption for various parameters (second)

Addition		**D128**	D160	D192	**S128**	S160	S192
Single precision	Time (1 thread)	**530**	823	1516	**525**	700	1495
	Time (4 thread)	**264**	374	657	**239**	321	654
	Time (10 thread)	**181**	269	452	**183**	248	423
	Amortized time	**57.5**	70.2	131	52.1	66.8	130
Double precision	Time (1 thread)	**858**	1303	2439	-	-	-
	Time (4 thread)	**366**	543	950	-	-	-
	Time (10 thread)	**256**	387	630	-	-	-
	Amortized time	**103**	112	253	-	-	-

Multiplication		**D128**	D160	D192	**S128**	S160	S192
Single precision	Time (1 thread)	**443**	674	1257	**426**	580	1236
	Time (4 thread)	**223**	314	551	**203**	282	530
	Time (10 thread)	**169**	249	392	**168**	226	383
	Amortized time	**42.7**	61.1	112	49.7	61.2	112
Double precision	Time (1 thread)	**808**	1230	2303	-	-	-
	Time (4 thread)	**402**	565	946	-	-	-
	Time (10 thread)	**293**	438	704	-	-	-
	Amortized time	**110**	165	190	-	-	-

$$x_f^3 = 1.82634005135360 * 10^{14}, \qquad x_f^4 = \text{-}6.278269952 * 10^{9},$$

where x_d^i and x_f^i denote double and single precision message, respectively. Then we evaluate $z_1 = (x_1 + x_2)$, $z_2 = (x_3 - x_4)$, $z_3 = z_1 \cdot z_2$, and $z_4 = z_3^2$ on the ciphertext domain, and results are listed in Table 3 and 4 and correct calculation values are listed which is round down. It can be directly checked that error between correct values and decryption result where overflow is not occur is bounded as known as Proposition 3.

Table 3: double precision(64-bit) operation results

\	$x_d^1 + x_d^2$	$x_d^3 - x_d^4$
Correct value	**6.24672463287407**32 $\cdot 10^{-24}$	**2.452407011899572**69 $\cdot 10^{22}$
OD-FPFHE	**6.24672463287407**17 $\cdot 10^{-24}$	**2.452407011899572**30 $\cdot 10^{22}$

\	$(x_d^1 + x_d^2)(x_d^3 - x_d^4)$	$[(x_d^1 + x_d^2)(x_d^3 - x_d^4)]^2$
Correct value	**0.1531951129106**61	**0.0234687426197**10
OD-FPFHE	**0.1531951129106**57	**0.0234687426197**09

Also, we perform the previous circuits with single precision floating-point numbers x_f^1, x_f^2, x_f^3, and x_f^4 and results are as given in Table 4.

Since the precision $p = 12$ for single precision is less than $p = 27$ for double precision, the error values between correct and decryption result in Table 4 is

Table 4: single precision(32-bit) operation results

\	$x_f^1 + x_f^2$	$x_f^3 - x_f^4$
Correct value	$\mathbf{8.38390868} \cdot 10^{-12}$	$\mathbf{1.82640283} \cdot 10^{14}$
OD-FPFHE	$\mathbf{8.38390815} \cdot 10^{-12}$	$\mathbf{1.82640279} \cdot 10^{14}$
\	$(x_f^1 + x_f^2)(x_f^3 - x_f^4)$	$[(x_f^1 + x_f^2)(x_f^3 - x_f^4)]^2$
Correct value	**1531.239**45	**2344694**.28
OD-FPFHE	**1531.239**37	**2344694**.00

bigger than Table 3. However, errors are bounded properly bounded as known as 3 when overflow is not occurs.

In addition, we choose following numbers DBL-MAX, DBL-MIN, FLT-MAX and FLT-MIN which are maximum and minimum of double and single precision floating-pint numbers, respectively, where these are provided in standard library in C++ language. We evaluate DBL-MAX·1000, DBL-MIN·0.001, FLT-MAX·1000, and FLT-MIN·0.001 on the ciphertext. All of decrypted results are invalid however, message of proof ciphertext was not a zero, meaning that overflow occurs. These all simulation codes are opened in public [4].

Conclusions In this paper, We proposed a floating-point fully homomorphic encryption. Since floating-point number system is widely used in many areas such as deep learning models, the proposed FPFHE can guarantee both privacy and accuracy for many applications. In addition, we proposed an OD-FPFHE, which has many applications. For instance, it is quite useful for continual learning models while keeping the privacy of training data such as privacy-preserving federated learning because the encrypted training data can be excluded from the training to avoid learning degeneration when it results in overflow.

Future works and open problems Since many applications need accurate floating-point division algorithm, more accurate and efficient division algorithm should be constructed. In addition, efficient floating-point homomorphic elementary functions such as exponential, logarithm, and N-th root function are also desirable in privacy-preserving machine learning.

The critical disadvantage of OD-FPFHE is having slower operation time. However, speed of operation can be improved in further researches as follows: Since large modulus Q affects bootstrapping time slower, a method of reducing a size of Q should be investigated. For instance, randomized gadget decomposition are reported that it reduces error amplification after running GSW-like multiplication[16]. Therefore, effective randomized gadget decomposition for OD-FPFHE and both rigorous and practical error analysis will improve speed of operation time.

[4] Codes are available in URL: github.com/Lee-Seung-Hwan/OD-FPFHE.

Acknowledgements This work was supported by the Samsung Research Funding and Incubation Center of Samsung Electronics under Project SRFC-IT1801-08

References

1. Agrawal, S., Goldwasser, S., Mossel, S.: Deniable fully homomorphic encryption from learning with errors. In: Annual International Cryptology Conference. pp. 641–670. Springer (2021)
2. Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., et al.: Homomorphic encryption standard. In: Protecting Privacy through Homomorphic Encryption, pp. 31–62. Springer (2021)
3. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)
4. Badrinarayanan, S., Miao, P., Raghuraman, S., Rindal, P.: Multi-party threshold private set intersection with sublinear communication. In: IACR International Conference on Public-Key Cryptography. pp. 349–379. Springer (2021)
5. Beaulieu-Jones, B.K., Wu, Z.S., Williams, C., Lee, R., Bhavnani, S.P., Byrd, J.B., Greene, C.S.: Privacy-preserving generative deep neural networks support clinical data sharing. Circulation: Cardiovascular Quality and Outcomes **12**(7), e005122 (2019)
6. Bourse, F., Minelli, M., Minihold, M., Paillier, P.: Fast homomorphic evaluation of deep discretized neural networks. In: Annual International Cryptology Conference. pp. 483–512. Springer (2018)
7. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
8. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: Annual cryptology conference. pp. 505–524. Springer (2011)
9. Capiński, M., Kopp, P.E.: Measure, integral and probability, vol. 14. Springer (2004)
10. Case, B.M., Gao, S., Hu, G., Xu, Q.: Fully homomorphic encryption with k-bit arithmetic operations. Cryptology ePrint Archive (2019)
11. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 360–384. Springer (2018)
12. Cheon, J.H., Hhan, M., Hong, S., Son, Y.: A hybrid of dual and meet-in-the-middle attack on sparse and ternary secret lwe. IEEE Access **7**, 89497–89506 (2019)
13. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)
14. Chillotti, I., Joye, M., Paillier, P.: Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In: International Symposium on Cyber Security Cryptography and Machine Learning. pp. 1–19. Springer (2021)
15. Chillotti, I., Ligier, D., Orfila, J.B., Tap, S.: Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for tfhe. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 670–699. Springer (2021)

16. Genise, N., Micciancio, D., Polyakov, Y.: Building an efficient lattice gadget toolkit: Subgaussian sampling and more. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 655–684. Springer (2019)
17. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the forty-first Annual ACM Symposium on Theory of Computing. pp. 169–178 (2009)
18. Gentry, C., Halevi, S.: Compressible fhe with applications to pir. In: Theory of Cryptography Conference. pp. 438–464. Springer (2019)
19. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Annual Cryptology Conference. pp. 75–92. Springer (2013)
20. Gorantala, S., Springer, R., Purser-Haskell, S., Lam, W., Wilson, R., Ali, A., Astor, E.P., Zukerman, I., Ruth, S., Dibak, C., et al.: A general purpose transpiler for fully homomorphic encryption. arXiv preprint arXiv:2106.07893 (2021)
21. Jäschke, A., Armknecht, F.: Accelerating homomorphic computations on rational numbers. In: International Conference on Applied Cryptography and Network Security. pp. 405–423. Springer (2016)
22. Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A.N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al.: Advances and open problems in federated learning. arXiv preprint arXiv:1912.04977 (2019)
23. Lee, J.W., Lee, E., Lee, Y., Kim, Y.S., No, J.S.: High-precision bootstrapping of rns-ckks homomorphic encryption using optimal minimax polynomial approximation and inverse sine function. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 618–647. Springer (2021)
24. Micciancio, D., Polyakov, Y.: Bootstrapping in fhew-like cryptosystems. In: Proceedings of the 9th Workshop on Encrypted Computing & Applied Homomorphic Cryptography. pp. 17–28 (2021)
25. Moon, S., Lee, Y.: An efficient encrypted floating-point representation using heaan and tfhe. Security and Communication Networks (2020)
26. Mukherjee, T., Stephens-Davidowitz, N.: Lattice reduction for modules, or how to reduce modulesvp to modulesvp. In: Annual International Cryptology Conference. pp. 213–242. Springer (2020)
27. Muller, J.M., Brisebarre, N., De Dinechin, F., Jeannerod, C.P., Lefevre, V., Melquiond, G., Revol, N., Stehlé, D., Torres, S., et al.: Handbook of floating-point arithmetic, vol. 1. Springer (2018)
28. Neukirch, J.: Algebraic number theory, vol. 322. Springer Science & Business Media (2013)
29. Rajaraman, V.: Ieee standard for floating point numbers. Resonance **21**(1), 11–30 (2016)
30. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 1–40 (2009)
31. Rigollet, P., Hütter, J.C.: High dimensional statistics. Lecture notes for course 18S997, **813**(814), 46 (2015)

Supplementary material

Omitted algorithms and proofs in the main paper are listed.

Proof of Lemma 4

Proof. When Algorithm 2 runs with i on Line 2, by using (4) and CMux gate analysis in Section 3.4 of [13], the additive error is derived as follows:

$$\sum_{j\in[l_{\mathrm{bl}}]}(X^{a_i}-1)\mathrm{G}^{-1}_{\mathrm{crt}}(\mathrm{ACC}^{(i)})E^{1}_{j}(X)+\sum_{j\in[l_{\mathrm{bl}}]}(X^{-a_i}-1)G^{-1}_{\mathrm{crt}}(\mathrm{ACC}^{(i)})E^{-1}_{j}(X)$$
$$+\sum_{j\in[\bar{l}_{\mathrm{bl}}]}\left[(X^{a_i}-1)A^{1}_{j}(X)+(X^{-a_i}-1)A^{-1}_{j}(X)\right]\mathit{sk\text{-}bl}(X)_j, \tag{16}$$

where $\mathrm{ACC}^{(i)}$ is the computed value after the $i-1$st iteration on line 3, $A^{1}_{j}(X)$ and $A^{-1}_{j}(X)$ are gadget error polynomials, $E^{1}_{j}(X)$ and $E^{-1}_{j}(X)$ are j-column error polynomials of BL^{1}_{i} and BL^{-1}_{i}, respectively.

Since errors $E^{1}_{j}(X)$, $E^{-1}_{j}(X)$ and secret key $\mathit{sk\text{-}bl}_j(X)$ follow symmetric distribution and each of them is multiplied with independent and bounded random variable, then the summands in each summation have *Pythagorean additivity* by Corollary 2. By induction on i, we obtain (11) by using *negacyclic property*, and Proposition 1. □

Algorithm 13 for Packing and proof of Lemma 5

Algorithm 13 : OUT ← **Packing**$((\mathbf{ct}_i)_{i\in[p]}, ev)$

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: $\mathrm{OUT}\left[\Delta\sum_{i\in[p]} m_i X^i\right] \in R^{K_{\mathrm{ct}}+1}_{N_{\mathrm{ct}},Q}$

```
Set OUT = (0(X), ...., Σ_{i∈[p]} b_i X^i)            ▷ where b_i is the b of ct_i
for (i, j, x) ∈ [p, K_gct, N_gct] do
    v = G_crt^{-1}(CT_{i,j,x})
    OUT += Σ_{y∈[l_pack]} v_y P_{j,x,y} X^i
end for
return OUT
```

Proof. The decryption output of the result OUT Algorithm 13 is as follows:

$$\varphi\left(\left(0(X),...,\sum_{i\in[p]}b_iX^i\right)+\sum_{i,j,x,y\in[p,K_{\mathrm{gct}},N_{\mathrm{gct}},l_{\mathrm{ks}}]}\mathrm{G}^{-1}_{\mathrm{crt}}(\mathrm{CT}_{i,j,x})_y\mathrm{P}_{j,x,y}X^i\right)$$

$$= \sum_{i\in[p]} b_i X^i + \sum_{i,j,x,y} \mathrm{G}_{\mathrm{crt}}^{-1}(\mathrm{CT}_{i,j,x})_y (sk\text{-}bl_{j,x} B_{\mathrm{pack}}^{y+1} + E'_{j,x,y}(X))X^i$$

$$= \sum_i \varphi(\mathrm{CT}_i)X^i + \sum_{j,x}(\sum_i A'_{j,x,y}X^i)sk\text{-}bl_{j,x} + \sum_{i,j,x,y} \mathrm{G}_{\mathrm{crt}}^{-1}(\mathrm{CT}_{i,j,x})_y X^i E'_{j,x,y}(X), \tag{17}$$

where $A'_{j,x,y}$ is a gadget error, and $E'_{j,x,y}(X)$ is the error polynomial of packing key $\mathrm{P}_{j,x,y}$.

Since errors $E'_{j,x,y}(X)$ and secret key $sk\text{-}bl_{j,x}$ follow symmetric distribution and each of them is multiplied with independent and bounded random variable, then the second and third summands in (17) have *Pythagorean additivity* by using Corollary 2. Since the first summation in (17) is $\sum_i(\Delta m_i X^i + \mathcal{E}_{\mathrm{bl}}^{(i)})$, hence (12) holds. □

Algorithm 14 for TensorProd and proof of Lemma 6

Algorithm 14 : OUT ← **TensorProd**$(\mathrm{CT}_1, \mathrm{CT}_2, ev)$

Input $\mathrm{CT}_1[\Delta m_1(X)] = (a_0, ..., a_{K_{\mathrm{ct}}\text{-}1}, b)$, $\mathrm{CT}_2[\Delta m_2(X)] = (\mathfrak{a}_0, ..., \mathfrak{a}_{K_{\mathrm{ct}}\text{-}1}, \mathfrak{b}) \in R_{n,Q}^{K_{\mathrm{ct}}+1}$
Output $\mathrm{OUT}[\Delta^2 m_1(X)m_2(X)] \in R_{n,Q}^{K_{\mathrm{ct}}+1}$
OUT $= \leftarrow \mathfrak{b}\mathrm{CT}_1 + b\mathrm{CT}_2 - (0, ..., 0, b\mathfrak{b})$
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.
  $v(X) = \mathrm{G}_{\mathrm{crt}}^{-1}\Big(\gamma_k\big(a_i\mathfrak{a}_j + a_j\mathfrak{a}_i\big)\Big)$
  OUT $\mathrel{+}= \sum_x v_x(X)\mathrm{Ten}_{k,x}$
end for
return OUT

Proof. We apply decryption function to OUT returned from Algorithm 14 as follows:

$$\varphi\Big(\mathfrak{b}\mathrm{CT}_1 + b\mathrm{CT}_2 - (0, ..., 0, b\mathfrak{b}) + \sum_{k,x} \mathrm{G}_{\mathrm{crt}}^{-1}\Big(\gamma_k\big[a_j\mathfrak{a}_i + a_i\mathfrak{a}_j\big]\Big)_x \mathrm{Ten}_{k,x}\Big)$$

$$= \varphi(\mathrm{CT}_1)\varphi(\mathrm{CT}_2) + \sum_{k,x} A''_{k,x}(X)sk_i(X)sk_j(X)$$

$$+ \sum_{k,x} \mathrm{G}_{\mathrm{crt}}^{-1}\Big(\gamma_k\big[a_j\mathfrak{a}_i + a_i\mathfrak{a}_j\big]\Big)_x E''_{k,x}(X), \tag{18}$$

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

$$m_1(X)\mathcal{E}_{\mathrm{pack},2}(X) + m_2(X)\mathcal{E}_{\mathrm{pack},1}(X) + \mathcal{E}_{\mathrm{pack},1}(X)\mathcal{E}_{\mathrm{pack},2}(X), \tag{19}$$

where $\mathcal{E}_{\text{pack},1}(X)$ and $\mathcal{E}_{\text{pack},2}(X)$ are packing error of CT_1 and CT_2, respectively.

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

$$\sum_{i\in[p]} m_{1,i}\mathcal{E}_{\text{pack},2}^{(p-1-i)} + m_{2,i}\mathcal{E}_{\text{pack},1}^{(p-1-i)} \tag{20}$$

by *negacyclic property*. Since other coefficients consist of p summation of product of message and packing error, without loss of generality, we analyze a worst-case error of $p-1$-th coefficient having messages $m_1(X) = m_2(X) = \sum_{i\in[p]} \Delta(\beta - 1)X^i$. Since $|\mathcal{E}_{\text{pack},1}(X)\mathcal{E}_{\text{pack},2}(X)| = O(N_{\text{ct}}|\mathcal{E}_{\text{pack},1}^{(p-1)}|^2)$, by using the fact $\Delta = \Omega(N_{\text{ct}}\ |\mathcal{E}_{\text{pack}}|)$, (16), and (17), then the (19) is bounded as $O(\Delta p\beta|\mathcal{E}_{\text{pack}}^{(p-1)}|)$. Moreover for (18), the first summation is bounded as $O(K_{\text{ct}}^2 N_{\text{ct}}^2 l_{\text{ten}} B_{ten})$, and the second summation is bounded as $O\big(\sigma B_{\text{ten}}\ K_{\text{gct}}\sqrt{l_{\text{ten}} N_{\text{gct}} v}\big)$, by using Corollary 2. □

Proof of Lemma 7

Proof. After running Algorithm 3 with input **ct**, the **ct** is multiplied by three values Δ^{-1}, $(\nu 2^{\eta_1+s+c+1+f-q})^{-1}$, and $2^{c+1+\log N_{\text{gct}}-q}$, that are listed in Line 1, 3, and 9. Since $2^{c+\log N_{\text{gct}}+1-q}/\Delta\nu 2^{\eta_1+s+c+1+f-q}$ is multiplied to **ct**, error in **ct** becomes the left of first term of (14). When Algorithm 3 runs on Line 1, $O(\sqrt{K_{\text{ct}}N_{\text{ct}}v})$-bounded floor errors are added, which is negligible compared to $|\mathcal{E}_{\text{ten}}^{(p-1)}|/\Delta$.

Next, we consider **ct** in Line 1 as the ciphertext modulo $\nu 2^{\eta_0}$. If the modulus $Q_0 = \nu 2^{\eta_0} + 1$ of **ct** is changed to $\nu 2^{\eta_0}$, then $O(\sqrt{K_{\text{ct}}N_{\text{ct}}})$ errors are added by following decryption equation on $\mathbb{Z}$:

$$b - a_i s_i = m + e + \bar{h}Q_0 = m + e + \bar{h} + \bar{h}\nu 2^{\eta_0} \in \mathbb{Z}$$

, for some $\bar{h} \in \mathbb{Z}$ where $\bar{h} = O(\sqrt{K_{\text{ct}}N_{\text{ct}}})$ for ternary secret key [11], which is negligible. Moreover, we regard the message $mQ_1 = m\nu 2^{\eta_1} + m\nu$ as a message $m\nu 2^{\eta_1}$ with error $m\nu$. Therefore, the message **out** becomes $(m_{f+s}...m_f)_{(2)}2^{\log N_{\text{gct}}-s}$ after rounding all in Algorithm 3 and up to $m\nu \le p(\beta-1)^2\nu$ error is added. Therefore, $\nu p(\beta-1)^2$ becomes the rest of first term of (14).

After rounding on Line 3, $O(\sqrt{N_{\text{ct}}K_{\text{ct}}v})$-bounded rounding error is added. After running on Line 5, errors in $\text{KS}_{j,x,k}$ are added and it is $O(\sigma B_{\text{ks}}\sqrt{l_{\text{ks}}N_{\text{ct}}K_{\text{ct}}v})$-bounded random variable by Corollary 2. Both errors are divided by $2^{q-1-\log N_{\text{gct}}}$, after running Line 9, which is second term in (14).

Finally, rounding errors after running on Line 9 are added. However, we use h-sparse secret key for encrypting KS and only a number of h rounding errors are added. By using subgaussian property with Corollary 2, this error is $O(\sqrt{hv})$-bounded, hence third term of (14) holds. □