Optimization of Homomorphic Comparison Algorithm on RNS-CKKS Scheme

EUNSANG LEE¹, JOON-WOO LEE¹, (Graduate Student Member, IEEE), YOUNG-SIK KIM², (Member, IEEE), JONG-SEON NO¹, (Fellow, IEEE).

¹Department of Electrical and Computer Engineering, INMC, Seoul National University, Seoul 08826, South Korea
²Department of Information and Communication Engineering, Chosun University, Gwangju 61452, South Korea

Corresponding author: Young-Sik Kim

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2021-0-00400, Development of Highly Efficient PQC Security and Performance Verification for Constrained Devices) (contribution: 50%). This work was also supported by the BK21 FOUR program of the Education and Research Program for Future ICT Pioneers, Seoul National University in 2021 (contribution: 50%).

ABSTRACT

Since the sign function can be used to implement the comparison operation, max function, and rectified linear unit (ReLU) function, several studies have been conducted to efficiently evaluate the sign function in the Cheon-Kim-Kim-Song (CKKS) scheme, one of the most promising fully homomorphic encryption schemes. Recently, Lee et al. (IEEE Trans. Depend. Sec. Comp.) proposed a practically optimal approximation method of sign function on the CKKS scheme using a composition of minimax approximate polynomials. In addition, Lee et al. proposed a polynomial-time algorithm that finds degrees of component polynomials minimizing the number of non-scalar multiplications for homomorphic comparison/max/ReLU functions. However, homomorphic comparison/max/ReLU functions using Lee et al.’s approximation method have not been successfully implemented on the residue number system variant CKKS (RNS-CKKS) scheme. In addition, the degrees of component polynomials found by Lee et al.’s algorithm are not optimized for the RNS-CKKS scheme because the algorithm does not consider that the running time of non-scalar multiplication depends much on the ciphertext level in the RNS-CKKS scheme. In this paper, we propose a fast algorithm for inverse minimax approximation error, a subroutine required to find the optimal set of degrees of component polynomials. This proposed algorithm makes it possible to find the optimal set of degrees of component polynomials with higher degrees than the previous study. In addition, we propose a method to find the degrees of component polynomials optimized for the RNS-CKKS scheme using the proposed algorithm for inverse minimax approximation error. We successfully implement the homomorphic comparison, max function, and ReLU function algorithms on the RNS-CKKS scheme with a low comparison failure rate (< $2^{-15}$) and provide the various parameter sets according to the precision parameter $\alpha$. We reduce the depth consumption of the homomorphic comparison, max function, and ReLU function algorithms by one depth for several $\alpha$. In addition, the numerical analysis demonstrates that the homomorphic comparison, max function, and ReLU function algorithms using the degrees of component polynomials found by the proposed algorithm reduce running time by 6%, 7%, and 6% on average compared with those using the degrees of component polynomials found by Lee et al.’s algorithm, respectively.

INDEX TERMS
Cheon–Kim–Kim–Song (CKKS) scheme, fully homomorphic encryption (FHE), homomorphic comparison operation, minimax approximate polynomial, Remez algorithm, residue number system variant CKKS (RNS-CKKS) scheme.

I. INTRODUCTION

HOMOMORPHIC encryption (HE) is a cryptosystem that allows some algebraic operations on encrypted data. Fully homomorphic encryption (FHE) is the HE that allows all algebraic operations on encrypted data. Fully homomorphic encryption (FHE) is the HE that allows all algebraic operations on encrypted...
Then, FHE has attracted significant attention in various applications, and its standardization process is in progress.

The Cheon–Kim–Kim–Song (CKKS) \[2\] scheme, one of the representative FHE schemes, allows the addition and multiplication of real and complex numbers. Since data is usually represented by real numbers, the CKKS scheme that can deal with real numbers has attracted much attention in many applications such as machine learning \[3\]–\[6\]. Thus, lots of research has widely been done to optimize the CKKS scheme \[7\]–\[11\]. In particular, Cheon et al. \[7\] proposed the residue number system variant CKKS scheme (RNS-CKKS). The running time of the RNS-CKKS scheme is ten times faster than that of the original CKKS scheme with one thread. In addition, the running time performance can be more improved in the multi-core environment because the RNS-CKKS scheme enables parallel computation. Thus, many HE libraries such as SEAL \[12\], PALISADE \[13\], and Lattigo \[14\] are implemented using the RNS-CKKS scheme.

Although the CKKS scheme can support virtually all arithmetic operations on encrypted data, several applications require non-arithmetic operations. One of the core non-arithmetic operations is the comparison operation, denoted as \(\text{comp}(a, b)\), and this outputs 1 if \(a > b\), 1/2 if \(a = b\), and 0 if \(a < b\). This comparison operation is widely used in various real-world applications, including machine learning algorithms such as support-vector machines, cluster analysis, and gradient boosting \[15\], \[16\]. The max function and the rectified linear unit (ReLU) function are other essential non-arithmetic operations that are widely used in deep learning applications \[17\], \[18\]. These three non-arithmetic operations can all be implemented using the sign function \(\text{sgn}(x)\), that is,

\[
\text{comp}(a, b) = \frac{1}{2}(\text{sgn}(a - b) + 1),
\]
\[
\text{max}(a, b) = \frac{1}{2}(a + b + (a - b)\text{sgn}(a - b)),
\]
\[
\text{ReLU}(x) = \frac{1}{2}(x + x\text{sgn}(x)),
\]

where \(\text{sgn}(x) = x/|x|\) for \(x \neq 0\), and 0 otherwise. Thus, several studies have been conducted to efficiently implement the sign function on the CKKS scheme \[9\], \[19\]. A method to approximate \(\text{sgn}(x)\) using the composition of component polynomials was proposed in \[19\], and it was proved that this method achieves the optimal asymptotic complexity. In addition, authors in \[9\] proposed a practically optimal method that approximates \(\text{sgn}(x)\) with the minimum number of non-scalar multiplications using a composition of minimax approximate polynomials.

Although authors in \[9\] proposed a comparison operation algorithm with practically optimal performance on the CKKS scheme, there are other research topics on comparison operation on the RNS-CKKS scheme to study further. First, since the rescaling error is somewhat large in the RNS-CKKS scheme, unlike in the CKKS scheme, it is required to deal with this somewhat large rescaling error well to achieve low approximation failure rates. Another research topic is to find a set of degrees of component polynomials that provides better comparison operation performance. Although authors in \[9\] also proposed a polynomial-time algorithm that determines the set of degrees minimizing the number of non-scalar multiplications, this set of degrees is not optimized for the RNS-CKKS scheme, unlike the CKKS scheme. This is because the running time of a non-scalar multiplication changes much with the current ciphertext level on the RNS-CKKS scheme. Thus, if we optimize the degrees of component polynomials considering the running time of non-scalar multiplication according to ciphertext level, the performance will be improved further.

### A. OUR CONTRIBUTIONS

There are three contributions in this paper as follows.

1. For the first time, we successfully implement the homomorphic comparison, max function, and ReLU function algorithms using a composition of minimax approximate polynomials on the RNS-CKKS scheme with low failure rate (< \(2^{-15}\)), and we provide proper parameter sets.

2. We improve the performance of an algorithm to find the inverse minimax approximation error, which is a subroutine to find the optimal set of degrees of component polynomials. While the optimal set of degrees of component polynomials that minimizes the number of non-scalar multiplications was found among degrees only up to 31 in the previous study \[9\], we find the optimal set of degrees of component polynomials among degrees up to 63 using the improved algorithm for inverse minimax approximation error (see Algorithm 7). As a result, the depth consumption of homomorphic comparison operation (resp. max/ReLU functions) is reduced by one depth when \(\alpha\) is 9 or 14 (resp. when \(\alpha\) is 16, 17, or 18), enabling one more multiplication operation. In addition, this improved algorithm for inverse minimax approximation error enables finding a set of degrees of component polynomials optimized for homomorphic comparison operation, max function, or ReLU function on the RNS-CKKS scheme (see Section IV). Our source code of finding optimized degrees is available at https://github.com/eslee3209/MinimaxComp_degrees.

3. We propose a method to find the set of degrees of component polynomials optimized for the homomorphic comparison, max function, and ReLU function on the RNS-CKKS scheme using the proposed fast algorithm for inverse minimax approximation error. Using the optimized set of degrees for the RNS-CKKS scheme obtained from the proposed algorithm, we reduce the running time of the homomorphic comparison, max function, and ReLU function algorithms by 6%, 7%, and 6%, respectively, compared to the previous work in \[9\] on the RNS-CKKS scheme library SEAL \[12\].

### B. RELATED WORKS
While it is not difficult to perform comparison operation (or max/ReLU function) in bit-wise FHE, such as the fastest homomorphic encryption in the West (FHEW) [20] or the fast fully homomorphic encryption over the torus (TFHE) [21], the comparison operation is very challenging in word-wise FHE such as the CKKS scheme. Thus, several studies on comparison operation in the CKKS scheme that uses the evaluation of approximate polynomials have been conducted [9], [19], [22]. Among them, the comparison operation proposed in [9] has the best performance, and we improve the performance of [9].

Another comparison operation method on the CKKS scheme that uses FHEW/TFHE bootstrapping was recently studied [23]–[26]. Although this approach uses FHEW/TFHE bootstrapping, users can still use efficient word-wise operations in the CKKS scheme with this approach, and when a comparison operation is required, users switch the ciphertexts to FHEW/TFHE ciphertexts and perform the comparison operation using FHEW/TFHE bootstrapping. This comparison method that uses FHEW/TFHE bootstrapping can be less efficient than the proposed homomorphic comparison that fully uses the CKKS packing in terms of amortized running time (running time per one comparison operation). However, this comparison method is still interesting research topic because this can have advantages in the case of large-precision comparison.

C. OUTLINE

The remainder of this paper is organized as follows. Section II describes preliminaries regarding the notation, the RNS-CKKS scheme, scaling factor management technique, and homomorphic comparison operation using minimax composite polynomial. In Section III, a fast algorithm to find the inverse minimax approximation error is proposed. A new algorithm that finds the set of degrees of component polynomials optimized for the homomorphic comparison on the RNS-CKKS scheme is proposed in Section IV. In Section V, the application to the min/max and ReLU functions is presented. In Section VI, numerical results for the homomorphic comparison, max function, and ReLU function algorithms that use the proposed set of degrees for the component polynomials are provided on the RNS-CKKS scheme library SEAL. Finally, concluding remarks are given in Section VII.

II. PRELIMINARIES

A. NOTATION

Let \( R = \mathbb{Z}[X]/(X^N + 1) \) and \( R_q = R/qR \) be the polynomial rings, where \( N \) is a power-of-two integer. Let \( C = \{q_0, q_1, \ldots, q_{N-1}\} \) be the set of positive integers that are coprime each other. Then, for \( a \in Z_q \), where \( Z_q \) is the set of integers modulo \( Q \) and \( Q = \prod_{i=0}^{N-1} q_i \), we denote the RNS representation of \( a \) with regard to \( C \) by \([a]_C = ([a]_0, \ldots, [a]_{q_{N-1}}) \in Z_{q_0} \times \cdots \times Z_{q_{N-1}}\). For the set of real numbers \( \mathbb{R} \) and the set of complex numbers \( \mathbb{C} \), a field isomorphism \( \tau: \mathbb{R}[X]/(X^N + 1) \to \mathbb{C}^{N/2} \) is defined by \( \tau: r(X) \mapsto (r(e^{\pi \over N}))_{0 \leq j < N/2} \), where \( \zeta = \exp(-\pi i / N) \) is a \((2N)\)-th root of unity in \( \mathbb{C} \). HWT_\( T_N(h) \) is the set of signed binary vectors in \( \{0, \pm 1\}_N \) with Hamming weight \( h \). For \( 0 < a, b \in \mathbb{R} \), we denote \([-b, -a] \cup [a, b] \) by \( R_{a,b} \).

In particular, if \( a = 1 - \tau \) and \( b = 1 + \tau \) for some \( \tau \in (0, 1) \), then \( R_{a,b} = R_{-1,1} \) is denoted by \( R_\tau \).

\( \{(n_1, n_2, \ldots, n_i): S(n_1, \ldots, n_i)\} \) denotes the number of tuples \((n_1, \ldots, n_i)\) such that the statement \( S(n_1, \ldots, n_i) \) is true. \( \alpha_{\text{max}}, \ell_{\text{max}}, m_{\text{max}}, n_{\text{max}}, \) and \( \ell_{\text{max}} \) denote the upper-bound of precision \( \alpha \), ciphertext level, the number of non-scalar multiplications, depth consumption, and running time, respectively. These values should be set large enough, and thus we set \( \alpha_{\text{max}} = 20, \ell_{\text{max}} = 30, m_{\text{max}} = 70, n_{\text{max}} = 40, \) and \( \ell_{\text{max}} = 240 \) in this paper. \( \ell_{\text{max}} \) denotes the upper-bound of degrees of component polynomials, and \( \ell_{\text{max}} \) of 31 or 63 is used in this paper.

B. RNS-CKKS SCHEME

Before describing the RNS-CKKS scheme, some basic operations for the RNS are presented. Let \( B = \{p_0, \ldots, p_{k-1}\}, C = \{q_0, \ldots, q_{\ell-1}\}, \) and \( D = \{p_0, \ldots, p_{k-1}, q_0, \ldots, q_{\ell-1}\}, \) where \( p_i \) and \( q_j \) are all distinct primes.

- \( \text{Conv} \_\text{C} \to \text{B} \) : For \([a]_C = (a_0, a_1, \ldots, a_{\ell-1}) \in Z_{q_0} \times \cdots \times Z_{q_{\ell-1}}, \) output \( \text{Conv} \_\text{C} \to \text{B} ([a]_C) = \sum_{j=0}^{\ell-1} [a(j) \cdot \tilde{q}_{i,j} \mod p_i]_0 \leq i < k \), where \( \tilde{q}_{i,j} = \prod_{j' \neq j} q_{j'} \in Z_i. \) This algorithm over integers \( \text{Conv} \_\text{C} \to \text{B} (\cdot) : \prod_{j=0}^{\ell-1} Z_{q_j} \to \prod_{j=0}^{k-1} Z_{p_j} \) can be extended to an algorithm over the polynomial rings as \( \text{Conv} \_\text{C} \to \text{B} (\cdot) : \prod_{j=0}^{k-1} R_{q_j} \to \prod_{j=0}^{k-1} R_{p_j}, \) by applying coefficient-wise.

- \( \text{ModUp} \_\text{C} \to \text{D} \) : For \([a]_C \in \prod_{j=0}^{k-1} R_{q_j}, \) output \( \text{ModUp} \_\text{C} \to \text{D} ([a]_C) = \prod_{i=0}^{k-1} R_{p_i} \times \prod_{j=0}^{\ell-1} R_{q_j} \).

- \( \text{ModDown} \_\text{D} \to \text{C} \) : For \(([a]_B, [b]_C) \in \prod_{i=0}^{k-1} R_{p_i} \times \prod_{j=0}^{\ell-1} R_{q_j}, \) output \( ([b]_C - \text{Conv} \_\text{C} \to \text{B} ([a]_B)) \cdot [P^{-1}]_C, \) where \( P = \prod_{i=0}^{k-1} p_i. \)

Then, the basic algorithms in the RNS-CKKS scheme are described as follows:

- \( \text{Setup} (\lambda; \Delta, L) \) : For a security parameter \( \lambda \), a scaling factor \( \Delta \), and the number of levels \( L \) (also called the maximum level), we set some parameters. The polynomial degree \( N \) of \( R \) is chosen so that the number of levels \( L \) can be supported with the security \( \lambda \). A secret key distribution \( \chi_{\text{key}} \), an error distribution \( \chi_{\text{err}} \) over \( R \), and an encryption key distribution \( \chi_{\text{enc}} \) are chosen according to the security parameter \( \lambda \). Bases with prime numbers \( B = \{p_0, p_1, \ldots, p_{k-1}\} \) and \( C = \{q_0, q_1, \ldots, q_{\ell-1}\} \) are selected so that \( p_i \equiv 1 \mod 2N \) for \( 0 \leq i < k - 1 \).
and \( q_j \equiv 1 \mod 2N \) for \( 0 \leq j \leq L \). \( q_0 \) is usually set close to \( 2^{500} \), and \( q_j - \Delta \) are as small as possible for \( 1 \leq j \leq L \). All prime numbers are distinct. Let \( \mathcal{D} = \mathcal{B} \cup \mathcal{C} \). Let \( \mathcal{C}_i = \{q_0, q_1, \ldots, q_i\} \) and \( \mathcal{D}_i = \mathcal{B} \cup \mathcal{C}_i \) for \( 0 \leq \ell \leq L \). Let \( P = \prod_{j=0}^{i-1} p_j \) and \( Q = \prod_{j=0}^{L} q_j \). Let \( \tilde{p}_i = \prod_{0 \leq \ell < i} p_{\ell+1} \) for \( 0 \leq i \leq k - 1 \), and \( \tilde{q}_{i,j} = \prod_{0 \leq \ell < j} q_{\ell+1} \) for \( 0 \leq j \leq \ell \). Then, the following numbers are computed as:

- \( [P^{-1}]_q \), for \( 0 \leq j \leq L \)
- \( [\tilde{p}_i]_p \) and \( [\tilde{p}_i^{-1}]_p \), for \( 0 \leq i \leq k - 1, 0 \leq j \leq L \)
- \( [\tilde{q}_{i,j}]_q \), for \( 0 \leq i \leq k - 1, 0 \leq j \leq \ell \leq L \)
- \( \text{Ecd}(z; \Delta) \): For \( z \in \mathbb{C}^{N/2} \), output \( \lfloor \pi - (\Delta z) \rfloor \) in \( \mathbb{R} \).
- \( \text{Dec}(m; \Delta) \): For \( m \in \mathbb{R} \), output \( \Delta^{-1} \cdot \pi(m) \in \mathbb{C}^{N/2} \).
- \( \text{KSGen}(s_1, s_2) \): This algorithm generates the switching key for switching the secret key \( s_1 \) to \( s_2 \). First, sample \( (a(0), \ldots, a(k+\ell)) \leftarrow U(\prod_{j=0}^{i-1} R_{p_j} \times \prod_{j=i}^{L} R_{q_j}) \) and \( e^r \leftarrow \chi_{\text{err}} \). Then, for given \( s_1, s_2 \in \mathbb{R} \), output the switching key \( \mathbf{swk} = (\mathbf{swk}(0), \ldots, \mathbf{swk}(k+\ell)) = (b(0)^{(0)}, a(0)^{(0)}, \ldots, b(k+\ell)^{(0)}, a(k+\ell)^{(0)})) \leftarrow U(\prod_{j=0}^{i-1} R_{p_j} \times \prod_{j=i}^{L} R_{q_j}) \), where \( b(\ell)^{(i)} \leftarrow -a(\ell)^{(i)} \cdot s_2 + e^r \) for \( 0 \leq \ell \leq k - 1 \) and \( b(k+\ell)^{(i)} \leftarrow -a(k+\ell)^{(i)} + s_2 + [P]_{q_j} \cdot s_1 + e^r \) mod \( q_j \) for \( 0 \leq j \leq L \).
- \( \text{KeyGen}(\lambda) \): This algorithm generates the secret key, public key, and the evaluation key. First, sample \( s \leftarrow \chi_{\text{key}} \) and set the secret key \( \mathbf{s} \leftarrow (1, s) \). Sample \( (a(0), \ldots, a(k+\ell)) \leftarrow U(\prod_{j=0}^{i-1} R_{p_j} \times \prod_{j=i}^{L} R_{q_j}) \) and \( e^r \leftarrow \chi_{\text{err}} \). Then, the public key is \( \mathbf{pk} = (\mathbf{pk}(0), \ldots, \mathbf{pk}(k+\ell)) \in \mathbb{R}_{p_j}^2 \cdot [(0, 0)] \), where \( b(\ell)^{(i)} \leftarrow -a(\ell)^{(i)} \cdot s + e^r \) for \( 0 \leq \ell \leq k - 1 \) and \( b(k+\ell)^{(i)} \leftarrow -a(k+\ell)^{(i)} + s_2 \cdot [P]_{q_j} + e^r \) mod \( q_j \) for \( 0 \leq j \leq L \).

D. HOMOMORPHIC COMPARISON OPERATING USING MINIMAX COMPOSITE POLYNOMIAL

In this paper, we assume the required depth consumption and the number of non-scalar multiplications for evaluating a polynomial of degree \( d \) with odd-degrees terms using the odd baby-step giant-step algorithm and the optimal level consumption are denoted by \( \text{deg}(d) \) and \( \text{mult}(d) \), respectively. The values of \( \text{deg}(d) \) and \( \text{mult}(d) \) for odd degrees \( d \) up to 63 are presented in Table IV.

The minimax approximate polynomial of degree at most \( D \) on \( \text{sgn}(x) \) is denoted by \( \text{MP}(D; d) \). In addition, for the minimax approximate polynomial \( p(x) = \text{MP}(D; d) \), the minimax approximation error \( \max_{x \in \mathbb{D}} ||p(x) - \text{sgn}(x)||_{\infty} \) is denoted by \( \text{ME}(D; d) \). It is known that for any continuous function \( f \) on \( D \), the minimax approximate polynomial of degree at most \( D \) on \( D \) is unique [29]. Moreover, the minimax approximate polynomial can be obtained using the improved multi-interval Remez algorithm [30].

For a domain \( D = R_{a,b} \) and a set of odd integers \( \{d_i\}_{1 \leq i \leq k} \), a composite polynomial \( p_k \circ \cdots \circ p_1 \) is called a minimax composite polynomial on \( D \) for \( \{d_i\}_{1 \leq i \leq k} \), denoted by \( \text{MCP}(D; \{d_i\}_{1 \leq i \leq k}) \), if the followings are satisfied:

- \( p_1 = \text{MP}(D; d_1) \)
- \( p_i = \text{MP}(p_{i-1} \circ \cdots \circ p_1(D); d_i) \) for \( i, 2 \leq i \leq k \).
TABLE 1. Required depth consumption and the number of non-scalar multiplications for evaluating polynomials of degree $d$ with odd-degree terms using the optimal level consumption technique \cite{10} and the odd baby-step giant-step algorithm \cite{28}.

| Polynomial degree $d$ | Depth consumption $\text{dep}(d)$ | Multiplications $\text{mult}(d)$ |
|-----------------------|-----------------------------------|-------------------------------|
| 3                     | 2                                 | 2                             |
| 5                     | 3                                 | 3                             |
| 7                     | 4                                 | 4                             |
| 9                     | 5                                 | 5                             |
| 11                    | 6                                 | 6                             |
| 13                    | 7                                 | 7                             |
| 15                    | 8                                 | 8                             |
| 17                    | 9                                 | 9                             |
| 19                    | 10                                | 10                            |
| 21                    | 11                                | 11                            |
| 23                    | 12                                | 12                            |
| 25                    | 13                                | 13                            |
| 27                    | 14                                | 14                            |
| 29                    | 15                                | 15                            |
| 31                    | 16                                | 16                            |
| 33                    | 17                                | 17                            |

Since $\text{ME}(R_\tau; d)$ is a strictly increasing function of $\tau$, its inverse function exists, which is called inverse minimax approximation error and denoted by $\text{IME}(\tau'; d)$. That is, for $\tau' \in (0, 1)$ and $d \in \mathbb{N}$, $\text{IME}(\tau'; d)$ is equal to a value $\tau' \in (0, 1)$ that satisfies $\text{ME}(R_{\tau'}; d) = \tau$. An approximate value of $\text{IME}(\tau'; d)$ can be obtained using binary search as in Algorithm \cite{3}. \cite{9}.

Algorithm 1: ApplyIMEbinary($\tau'; d, \tilde{i}$) \cite{9}

Input: Target maximum error $\tau'$, an odd degree $d$, and iteration number $\tilde{i}$
Output: An approximate value of $\text{IME}(\tau'; d)$
1. \text{min} \leftarrow 2^{-2^{-21}} \text{and} \text{max} \leftarrow 1 - 2^{-21}
2. \text{while } \tilde{i} > 0 \text{ do}
3. \text{if } \text{ME}(R_{\min + \max}/2; d) < \tau' \text{ then}
4. \text{min} \leftarrow \frac{\min + \max}{2}
5. \text{else}
6. \text{max} \leftarrow \frac{\min + \max}{2}
7. \tilde{i} \leftarrow \tilde{i} - 1
8. \text{end}
9. \text{return } \frac{\min + \max}{2}

Algorithm 2: ComputehG($\tau$) \cite{9}

Input: An input $\tau$ and an odd maximum degree $d_{\text{max}}$
Output: 2-dimensional tables $\hat{h}$ and $\hat{G}$
1. Generate 2-dimensional tables $\hat{h}$ and $\hat{G}$, both of which have size of $(m_{\text{max}} + 1) \times (n_{\text{max}} + 1)$.
2. \text{for } m \leftarrow 0 \text{ to } m_{\text{max}} \text{ do}
3. \text{for } n \leftarrow 0 \text{ to } n_{\text{max}} \text{ do}
4. \text{if } m \leq 1 \text{ or } n \leq 1 \text{ then}
5. $\hat{h}(m, n) \leftarrow \tau$
6. $\hat{G}(m, n) \leftarrow \phi$
7. \text{else}
8. $j \leftarrow \arg\max_{1 \leq i \leq \min(m\cdot\text{mult}(2i + 1) / \text{mult}(2i + 1) \leq m \text{ mult}(2i + 1) \leq n \text{ dep}(2i + 1) \leq n)} \{1, n - \text{dep}(2i + 1); 2i + 1\}$
9. $\hat{h}(m, n) \leftarrow \text{IME}(\hat{h}(m - \text{mult}(2j + 1); 2j + 1))$
10. $\hat{G}(m, n) \leftarrow \{2j + 1\} \cup \hat{G}(m - \text{mult}(2j + 1), n - \text{dep}(2j + 1))$
11. \text{end}
12. \text{end}
13. \text{end}

Algorithm 3: ComputeMinDep($\alpha, \epsilon$) \cite{9}

Input: Precision parameters $\alpha$ and $\epsilon$
Output: Minimum depth consumption $M_{\text{dep}}$
1. $\hat{h}, \hat{G} \leftarrow \text{ComputehG}(2^{1 - \alpha})$
2. \text{for } i \leftarrow 0 \text{ to } n_{\text{max}} \text{ do}
3. \text{if } \hat{h}(m_{\text{max}}, i) \geq \delta = \frac{1 - \epsilon}{1 + \epsilon} \text{ then}
4. $M_{\text{dep}} \leftarrow i$
5. \text{return } M_{\text{dep}}$
6. \text{end}
7. \text{if } i = n_{\text{max}} \text{ then}
8. \text{return } \perp$
9. \text{end}
10. \text{end}

The comparison operation is denoted as
\[
\text{comp}(a, b) = \begin{cases} 
1 & \text{if } a > b \\
1/2 & \text{if } a = b \\
0 & \text{if } a < b.
\end{cases}
\]

The procedure of obtaining an approximate value of $\text{comp}(a, b)$ for given precision parameters $\alpha, \epsilon$ and inputs $a, b \in [0, 1]$ is summarized as follows:
1. Obtain the minimum depth consumption $M_{\text{dep}}$ from ComputeMinDep algorithm in Algorithm\cite{3}
2. Choose depth consumption $D(\geq M_{\text{dep}})$ and obtain the optimal set of degrees $M_{\text{degs}}$ from ComputeMinMultDegs algorithm in Algorithm\cite{3}
3. For some appropriate margin $\eta$, perform the homomorphic comparison algorithm MinimaxComp in Algo-
Algorithm 4: ComputeMinMultDegs($\alpha, \epsilon, D$) [9]

Input: Precision parameters $\alpha$ and $\epsilon$, and depth consumption $D$

Output: Minimum number of multiplications $M_{\text{mult}}$ and the optimal set of degrees $M_{\text{degs}}$

1. $\hat{h}, \hat{G} \leftarrow \text{ComputeG}(2^{1-\alpha})$
2. for $j \leftarrow 0$ to $m_{\text{max}}$
3. if $h(j, D) \geq \delta = \frac{1-\epsilon}{1+\epsilon}$ then
4. $M_{\text{mult}} \leftarrow j$
5. Go to line 11
6. end
7. if $j = m_{\text{max}}$ then
8. return $\perp$
9. end
10. end
11. $M_{\text{degs}} \leftarrow \hat{G}(M_{\text{mult}}, D)$; // $M_{\text{degs}}$: ordered set
12. return $M_{\text{mult}}$ and $M_{\text{degs}}$

Algorithm 5: MinimaxComp($a, b; \alpha, \epsilon, D, \eta$) [9]

Input: Inputs $a, b \in (0, 1)$, precision parameters $\alpha$ and $\epsilon$, depth consumption $D$, and margin $\eta$

Output: Approximate value of $\text{comp}(a, b)$

1. $M_{\text{degs}} = \{d_1, d_2, \cdots, d_k\} \leftarrow \text{ComputeMinMultDegs}(\alpha, \epsilon, D)$
2. $p_1 \leftarrow \text{MP}(R_{1-\epsilon}; d_1)$
3. $\tau_1 \leftarrow \text{ME}(R_{1-\epsilon}; d_1) + \eta$
4. for $i \leftarrow 2$ to $k$
5. $p_i \leftarrow \text{MP}(R_{\tau_{i-1}}; d_i)$
6. $\tau_i \leftarrow \text{ME}(R_{\tau_{i-1}}; d_i) + \eta$
7. end
8. return $p_0 \circ p_{a-1} \circ \cdots \circ p_1(a-b)+1$

Computing $\hat{G}$ using $M_{\text{degs}}$.

The degrees $M_{\text{degs}} = \{d_1, \cdots, d_k\}$ obtained from the ComputeMinMultDegs algorithm satisfies $\text{deg}(p_i) = d_i$, $1 \leq i \leq k$. $M_{\text{degs}}$ is the optimal set of degrees such that the homomorphic comparison operation minimizes the number of non-scalar multiplications and satisfies the comparison operation error condition in [1] for the given depth consumption $D$.

III. FAST ALGORITHM FOR INVERSE MINIMAX APPROXIMATION ERROR

The ComputeG algorithm in Algorithm [2] [9] should be performed to obtain the optimal set of degrees from the ComputeMinMultDegs algorithm. For performing the ComputeG algorithm, the inverse minimax approximation error $\text{IME}(\tau'; d)$ should be computed many times. That is, the AppIMEbinary algorithm [9] determining an approximate value of $\text{IME}(\tau'; d)$ should be called many times. However, a single call of AppIMEbinary algorithm also requires multiple computations of $\text{ME}(R_{\tau'; d})$, that is, several calls for the improved multi-interval Remez algorithm [30]. As a result, the ComputeG algorithm requires significant running time. Specifically, the number of computations of $\text{IME}(\tau'; d)$ in ComputeG for each precision parameter $\alpha$ is given as follows:

$$
\sum_{m=0}^{m_{\text{max}}} \sum_{n=0}^{n_{\text{max}}} \left\{ i; \text{mult}(2i+1) \leq m, \text{dep}(2i+1) \leq n, \right. \\
\left. 1 \leq i \leq \frac{d_{\text{max}}-1}{2} \right\}
$$

$$
= \sum_{i=1}^{d_{\text{max}}-1} \left\{ (m, n); \text{mult}(2i+1) \leq m \leq m_{\text{max}}, \right. \\
\left. \text{dep}(2i+1) \leq n \leq n_{\text{max}} \right\}
$$

$$
= \sum_{i=1}^{d_{\text{max}}-1} (m_{\text{max}} - \text{mult}(2i+1) + 1)(n_{\text{max}} - \text{dep}(2i+1) + 1)
$$

$$
= \frac{d_{\text{max}}-1}{2}(m_{\text{max}}+1)(n_{\text{max}}+1)
$$

If we set $d_{\text{max}} = 31$, $m_{\text{max}} = 70$, and $n_{\text{max}} = 40$ as in [9], the number of computations of $\text{IME}(\tau'; d)$ is $117,675$.

To obtain a precise approximate value of $\text{IME}(\tau'; d)$ using the AppIMEbinary algorithm, it is required to iterate at
least ten times. Then, the expected number of computations of \( \text{ME}(R_{t}; d) \) in \text{ComputeG} is at least \( 117,675 \times 10 = 1,176,750 \). It should be noted that this is the case for only one value of a precision parameter \( \alpha \), where the input \( \tau \) of \text{ComputeG} algorithm is \( 2^{1-\alpha} \). To perform the \text{ComputeG} algorithm for \( \alpha \) from 4 to 20, around 1,176,750 \times 17 = 20,004,750 calls for \( \text{ME}(R_{t}; d) \) are required. Because of the large number of calls for \( \text{ME}(R_{t}; d) \), the value of \( d_{\text{max}} \) larger than 31 could not be used in \cite{9}, failing to improve the performance of homomorphic comparison operation using higher degrees. Thus, it is desirable to study how to efficiently find approximate value of \( \text{IME}(\tau'; d) \).

**A. PROPOSED ALGORITHM FOR INVERSE MINIMAX APPROXIMATION ERROR**

We propose a fast method to find the approximate value of \( \text{IME}(\tau'; d) \), which enables a value of \( d_{\text{max}} \) larger than 31. Our procedure of the proposed method is given as follows:

1) Sample the values of \( \tau \) at moderate intervals.
2) Compute the values of \( \text{ME}(R_{t}; d) \) for the sampled \( \tau \).
3) For \( \tau' \in (0, 1) \), obtain an approximate value of \( \text{IME}(\tau'; d) \) by interpolation using the computed sample values of \( \text{ME}(R_{t}; d) \).

For \( \alpha_{\text{max}} \), which is the upper-bound of \( \alpha \), we consider sampling \( \tau \) between \( 2^{-\alpha_{\text{max}}-1} \) and \( 1-2^{-\alpha_{\text{max}}-1} \). If \( \tau \) is close to zero or one, sampling should be very dense. However, sampling the whole range densely between \( 2^{-\alpha_{\text{max}}-1} \) and \( 1-2^{-\alpha_{\text{max}}-1} \) requires a large number of samples. Thus, we propose to sample densely when \( \tau \) is close to zero or one, and sparsely, otherwise.

Specifically, we first sample \( t \) uniformly between \( -\alpha_{\text{max}} - 1 \) and \( 1 \). Then, we compute \( \text{ME}(R_{2t}; d) \) for the sampled values of \( t \). In addition, we sample \( t \) uniformly between 1 and \( \alpha_{\text{max}} + 1 \) and compute \( \text{ME}(R_{2t-2-\cdots}; d) \) for the sampled values of \( t \). Using interpolation with these samples, we can achieve a precise approximation of \( \text{IME}(\tau'; d) \) with a smaller number of samples. Precision \( \bar{n} \) determines how frequently the values of \( t \) are sampled, and we set \( \bar{n} = 10 \). For a given maximum degree \( d_{\text{max}} \) and a precision \( \bar{n} \), \text{StoreME} algorithm in Algorithm 6 stores \( 2^t \) (resp. \( 1-2^{-t} \)) in a two-dimensional table \( \hat{X} \) and \( \hat{ME}(R_{t'); d) \) and \( \text{ME}(R_{t-2-\cdots}; d) \) in a two-dimensional table \( \hat{Y} \) for all degrees \( d \), \( 3 \leq d \leq d_{\text{max}} \). For \( \bar{n} = 10 \) and \( \alpha_{\text{max}} = 20 \), the number of calls for \( \text{ME}(R_{t'}; d) \) is 6,030 for \( d_{\text{max}} = 31 \) and 12,462 for \( d_{\text{max}} = 63 \).

\text{AppIME} algorithm in Algorithm 7 outputs an approximate value of \( \text{IME}(\tau'; d) \) using the tables \( \hat{X} \) and \( \hat{Y} \) obtained from \text{StoreME} algorithm. Here, many calls for \text{AppIME} algorithm require only one computation of tables \( \hat{X} \) and \( \hat{Y} \), that is, one execution of \text{StoreME} algorithm. That is, \text{StoreME} is performed only once for various precision parameters \( \alpha \).

**B. RUNNING TIME OF THE PROPOSED ALGORITHM**

We compare the running time of \text{ComputeG} algorithm using the previous algorithm for inverse minimax approximation error with that using the proposed algorithm. The numerical analysis is conducted on a Linux PC with Intel Core i7-10700 CPU at 2.90GHz with one thread. One call for the improved multi-interval Remez algorithm takes about 1.4 s on average when \( d_{\text{max}} = 31 \) and about 4.9 s on average when \( d_{\text{max}} = 63 \). Then, the expected running time of \text{ComputeG} can be obtained. Table 2 shows the expected running time of \text{ComputeG} algorithm for \( \alpha \) from 4 to 20 using the previous and proposed algorithms for inverse minimax approximation error. It can be seen from Table 2 that using the proposed \text{AppIME} algorithm requires much less running time than using the previous \text{AppIME} algorithm, enabling the execution of \text{ComputeG} algorithm for \( d_{\text{max}} = 63 \). While the running time of 17 hours might still seem to be large, it should be noted that this process only needs to be done once because the goal of this process is just to find the optimal set of degrees.

While \text{ComputeG} algorithm in Algorithm 2 could only be performed for \( d_{\text{max}} \leq 31 \) in \cite{9}, we perform \text{ComputeG} algorithm for \( d_{\text{max}} \leq 63 \) using the proposed fast \text{AppIME} algorithm in Algorithm 7. Table 3 lists the optimal sets of degrees \( M_{\text{degs}} \) and the corresponding minimum depth consumption \( D_{\text{min}} \) for \( d_{\text{max}} = 31 \) and \( d_{\text{max}} = 63 \). From Table 3, it can be seen that the depth consumption is reduced by one when \( \alpha = 9 \) or 14. That is, for \( \alpha = 9 \) or \( \alpha = 14 \), high \( d_{\text{max}} \) enables one more non-scalar multiplication per homomorphic comparison operation in the FHE setting, where the available number of operations is very limited per bootstrapping. Furthermore, the proposed \text{AppIME} algorithm enables finding a set of degrees optimized for the RNS-CKKS scheme, described in Section IV.

**IV. FINDING DEGREES OF COMPONENT POLYNOMIALS OPTIMIZED FOR THE RNS-CKKS SCHEME**

Unlike the previous study on homomorphic comparison operation in the CKKS scheme \cite{9}, we study homomorphic-computation...
Algorithm 7: AppIME(τ, d; d_{max}, n)

Input: Target maximum error τ, degree d, the odd maximum degree d_{max}, and precision n
Output: An approximate value of IME(τ; d)
1 \( \tilde{X}, \tilde{Y} \leftarrow \text{StoreME}(d_{\text{max}}, n) \)
2 for \( j = 0 \) to \( 2n_\alpha_{\text{max}} + 1 \) do
3 \( x_j \leftarrow \tilde{X}(\frac{d_{\text{max}} - 3}{2}, j) \)
4 \( y_j \leftarrow \tilde{Y}(\frac{d_{\text{max}} - 3}{2}, j) \)
5 end
6 for \( j = 1 \) to \( 2n_\alpha_{\text{max}} + 1 \) do
7 if \( y_j \geq \tau \) then
8 \( x_j < 0.5 \) and \( x_j \geq 0.5 \) then
9 \( \text{return } x_j \)
10 else if \( y_j < 0.5 \) and \( y_j \geq 0.5 \) then
11 \( \text{return } x_j \)
12 else if \( x_j < 0.5 \) and \( y_j < 0.5 \) then
13 \( p \leftarrow \log x_j + \left( \tau - \log y_j \right) \frac{\log x_j - \log x_j - 1}{\log y_j - \log y_j - 1} \)
14 \( \text{return } 2^p \)
15 else if \( x_j < 0.5 \) and \( y_j < 0.5 \) then
16 \( p \leftarrow -\log \left( 1 - x_j \right) \)
17 \( -\left( \tau - y_j \right) \frac{\log \left( 1 - x_j \right) - \log \left( 1 - x_j \right) - 1}{\log y_j - \log y_j - 1} \)
18 \( \text{return } 1 - 2^{-p} \)
19 else if \( y_j < 0.5 \) then
20 \( p \leftarrow -\log \left( 1 - x_j \right) \)
21 \( +\left( \tau + \log \left( 1 - y_j \right) \right) \frac{\log \left( 1 - x_j \right) - \log \left( 1 - x_j \right) - 1}{\log y_j - \log y_j - 1} \)
22 \( \text{return } 1 - 2^{-p} \)
23 else
24 \( \text{return } \bot \)
25 end
26 end
27 \text{return } \bot

TABLE 2. Expected running time of ComputeG algorithm for α from 4 to 20 using the previous and proposed algorithms for inverse minimax approximation error.

| \( d_{\text{max}} \) | \( d_{\text{max}} \) using AppIMEBinary | \( d_{\text{max}} \) using AppIME |
|-------------------|-------------------------------|-----------------|
| 31                | 7,092,400                     | 6,030           |
| 63                | 20,004,750                    | 12,462          |

expected number of calls for ME(\( R_\alpha \); d)

| expected running time of ComputeG for α from 4 to 20 |
|-------------------|------------------|
| 31                | 2,758 hours      |
| 63                | 27,228 hours     |

There is another difference between the homomorphic comparison operation in the CKKS scheme and that in the RNS-CKKS scheme. For a given depth consumption \( D \), the set of degrees \( M_{\text{degs}} \) that minimizes the number of non-scalar multiplications can be obtained using the ComputeMinMultDegs algorithm. Because the computation time of a non-scalar multiplication does not depend much on the current ciphertext modulus in the CKKS scheme, minimizing the number of non-scalar multiplications corresponds to minimizing running time. However, since the computation time of a non-scalar multiplication depends much on the current level in the RNS-CKKS scheme, minimizing the number of non-scalar multiplications does not always correspond to minimizing running time.

Fig. 1 shows the computation time of an example polynomial of degree seven according to the current level on the RNS-CKKS scheme library SEAL [12]. From Fig. 1 it can be seen that the computation time of a polynomial tends to increase quadratically according to the maximum level. For example, we consider two ordered sets \( M_{\text{degs}} = \)
{7, 31, 31} and \( M_{\text{degs}} = \{31, 31, 7, 7\} \). In the CKKS scheme, the computation time of the homomorphic comparison operation using two sets of degrees will be almost the same. However, the homomorphic comparison operation using the former is faster in the RNS-CKKS scheme because a high degree polynomial is computed in a lower level in the former case. Our core idea is to determine the set of degrees \( M_{\text{degs}} \) that minimizes the running time itself rather than the number of non-scale multiplications. Specifically, we modify the previous ComputeHGG and ComputeMinMultDega planner algorithms so that the modified algorithms can find the set of degrees \( M_{\text{degs}} \) that minimizes the running time.

Now, we define \( C_t(i, \ell') \) for \( 0 \leq i \leq \frac{d_{\text{max}} - 3}{2}, 0 \leq \ell \leq \ell_{\text{max}} \). First, we set up for the maximum level \( \ell \). Then, starting from level \( \ell'(\ell \leq \ell) \), any polynomial of degree \( 2i + 3 \) is evaluated using the optimal level consumption technique [10] and odd baby-step giant-step algorithm [28]. If \( t \) is the running time of the evaluation polynomial in milliseconds, then we define \( C_t(i, \ell') \) as \( C_t(i, \ell') = \frac{t}{100} \). Here, if \( \ell < \ell' \) or \( \ell' \leq \left\lfloor \log_2(2i + 3) \right\rfloor \), the polynomial evaluation is infeasible, and thus, \( C_t(i, \ell') \) is set to a large enough number 100,000 in this case. We obtain the values of \( C_t(i, \ell') \) by performing polynomial evaluation on encrypted data, and this computation is done on Intel Core i7-10700 CPU at 2.90GHz in single thread with an Ubuntu 20.04 LTS distribution. Then, \( u_{\tau, L}(m, n) \) and \( V_{\tau, L}(m, n) \) are defined recursively using the values of \( C_t(i, \ell') \) as follows:

\[
\begin{align*}
  u_{\tau, L}(m, n) &= \begin{cases} 
  \tau, & \text{if } m \leq 1 \text{ or } n \leq 1 \\
  \text{IME}(u_{\tau, L}(m - CL(j_{m, n} - 1, n), n - \text{dep}(2j_{m, n} + 1)); 2j_{m, n} + 1), & \text{otherwise,}
  \end{cases} \\
  V_{\tau, L}(m, n) &= \begin{cases} 
  \phi, & \text{if } m \leq 1 \text{ or } n \leq 1 \\
  \{2j_{m, n} + 1\} \cup V_{\tau, L}(m - CL(j_{m, n} - 1, n), n - \text{dep}(2j_{m, n} + 1)), & \text{otherwise,}
  \end{cases}
\end{align*}
\]

where \( j_{m, n} = \arg\max_{1 \leq i \leq n} \text{IME}(u_{\tau, L}(m - CL(i - 1, n), n - \text{dep}(2i + 1)); 2i + 1). \)

\( u_{\tau, L}(m, n) \) implies the maximum value of \( \tau' \in (0, 1) \) such that there exists a set of degrees \( \{d_i\}_{1 \leq i \leq k} \) that satisfies the followings:

\[
\begin{align*}
  \text{MCE}(R_{\tau, L}(m, n); \{d_i\}_{1 \leq i \leq k}) &\leq \tau, \\
  \sum_{i=1}^{k} \text{dep}(d_i) &\leq n, \\
  \sum_{i=1}^{k} \frac{d_i - 3}{2} \times L - \sum_{j=1}^{i-1} \text{dep}(d_j) &\leq m,
\end{align*}
\]

where \( \sum_{i=1}^{k} \frac{d_i - 3}{2} \times L - \sum_{j=1}^{i-1} \text{dep}(d_j) \) is the running time of the homomorphic comparison operation using set of degrees \( M_{\text{degs}} = \{d_i\}_{1 \leq i \leq k}. \). In addition, for \( V_{\tau, L}(m, n) = \{d_i\}_{1 \leq i \leq k} \), we have

The ComputeuV algorithm in Algorithm 8 outputs two-dimensional tables \( \hat{u} \) and \( \hat{V} \) that store the values of \( u_{\tau, L} \) and \( V_{\tau, L} \), respectively. This ComputeuV algorithm requires many computations of \( \text{IME}(\tau'; d) \) for the modified algorithms can find the set of degrees \( M_{\text{degs}} \) that minimizes the running time.

\[
\begin{algorithm}
\textbf{Input:} \tau, maximum level \( \ell \) \\
\textbf{Output:} 2-dimensional tables \( \hat{u} \) and \( \hat{V} \)
\begin{enumerate}
  \item Generate 2-dimensional tables \( \hat{u} \) and \( \hat{V} \), both of which have size of \( (\ell_{\text{max}} + 1) \times (n_{\text{max}} + 1) \)
  \item for \( m \leftarrow 0 \) to \( \ell_{\text{max}} \) do
    \item for \( n \leftarrow 0 \) to \( n_{\text{max}} \) do
      \item if \( m \leq 1 \) or \( n \leq 1 \) then
        \item \( \hat{u}(m, n) \leftarrow \tau \)
      \item \( \hat{V}(m, n) \leftarrow \phi \)
      \item else
        \item \( j \leftarrow \arg\max_{1 \leq i \leq n} \text{IME}(\hat{u}(m - CL(i - 1, n), \text{dep}(2i + 1)); 2i + 1) \)
        \item \( \hat{u}(m, n) \leftarrow \text{IME}(\hat{u}(m - CL(j - 1, n), n - \text{dep}(2j + 1)); 2j + 1) \)
        \item \( \hat{V}(m, n) \leftarrow \{2j + 1\} \cup \hat{V}(m - CL(j - 1, n), n - \text{dep}(2j + 1)) \)
    \item end
  \item end
\end{enumerate}
\end{algorithm}

Then, the ComputeMinTimeDega algorithm in Algorithm 9 outputs the minimum running time \( M_{\text{time}} \) (in 100 ms) and the optimal set of degree \( M_{\text{degs}} \) using two tables \( \hat{u} \) and \( \hat{V} \) obtained from the ComputeuV algorithm.

\[
\begin{algorithm}
\textbf{Input:} Precision parameters \( \alpha \) and \( \epsilon \), and depth consumption \( D \) \\
\textbf{Output:} Minimum running time \( M_{\text{time}} \) and the optimal set of degrees \( M_{\text{degs}} \)
\begin{enumerate}
  \item \( \hat{u}, \hat{V} \leftarrow \text{ComputeuV}(2^{1 - \alpha}; D) \)
  \item for \( j \leftarrow 0 \) to \( \ell_{\text{max}} \) do
    \item if \( \hat{u}(j, D) \geq \frac{1 - \epsilon}{1 + \epsilon} \) then
      \item \( M_{\text{time}} \leftarrow j \)
    \item Go to line 11
  \item end
  \item if \( j = \ell_{\text{max}} \) then
    \item return \( \perp \)
  \item end
\end{enumerate}
\end{algorithm}

\[
M_{\text{degs}} \leftarrow \hat{V}(M_{\text{time}}, D); \quad // M_{\text{degs}}: \text{ordered set}
\]
Now, we propose the homomorphic comparison algorithm OptMinMaxComp in Algorithm 10 that uses ComputeMinTimeDegs algorithm. This is the modified algorithm of the previous MinimaxComp algorithm in Algorithm 5, minimizing the running time on the RNS-CKKS scheme for a given depth consumption $D$.

**Algorithm 10: OptMinMaxComp**

**Input:** Inputs $a, b \in (0, 1)$, precision parameters $\alpha$ and $\epsilon$, depth consumption $D$, and margin $\eta$.

**Output:** Approximate value of $\text{comp}(a, b)$.

1. $M_{\text{degs}} = \{d_1, d_2, \cdots, d_k\} \leftarrow \text{ComputeMinTimeDegs}(\alpha, \epsilon, D)$.
2. $p_1 \leftarrow \text{MP}(\tilde{R}_{1-\epsilon,1}; d_1)$.
3. $\tau_1 \leftarrow \text{ME}(\tilde{R}_{1-\epsilon,1}; d_1) + \eta$.
4. for $i \leftarrow 2$ to $k$ do
   5. $p_i \leftarrow \text{MP}(R_{\tau_{i-1}}; d_i)$.
   6. $\tau_i \leftarrow \text{ME}(R_{\tau_{i-1}}; d_i) + \eta$.
5. end
6. return $\frac{(a-b)p_k \circ \cdots \circ p_1 (a-b) + (a+b)}{2}$.

**Algorithm 11:** OptMinMaxMax($a, b, \alpha, \zeta, D, \eta$)

**Input:** Inputs $a, b \in [0, 1]$, precision parameter $\alpha$, max function factor $\zeta$, depth consumption $D$, and margin $\eta$.

**Output:** Approximate value of $\max(a, b)$.

1. $M_{\text{degs}} = \{d_1, d_2, \cdots, d_k\} \leftarrow \text{ComputeMinTimeDegs}(\alpha, \zeta \cdot 2^{-\alpha}, D - 1)$.
2. $p_1 \leftarrow \text{MP}(R_{1-\epsilon,1}; d_1)$.
3. $\tau_1 \leftarrow \text{ME}(\tilde{R}_{1-\epsilon,1}; d_1) + \eta$.
4. for $i \leftarrow 2$ to $k$ do
   5. $p_i \leftarrow \text{MP}(R_{\tau_{i-1}}; d_i)$.
   6. $\tau_i \leftarrow \text{ME}(R_{\tau_{i-1}}; d_i) + \eta$.
5. end
6. return $\frac{(a-b)p_k \circ \cdots \circ p_1 (a-b) + (a+b)}{2}$.

V. APPLICATION TO MIN/MAX AND RELU FUNCTION

In this section, we apply the methods of improving homomorphic comparison operation proposed in Sections III and IV to max and ReLU functions. First, the max function is an important operation that is used in many applications including deep learning. The max function is easily implemented using the sign function, that is,

$$
\max(a, b) = \frac{(a + b) + (a - b) \text{sgn}(a - b)}{2}.
$$

Thus, the approximate polynomial for the max function, $\tilde{p}(a, b)$, can be obtained from the approximate polynomial for the sign function $p(x)$ as:

$$
\tilde{p}(a, b) = \frac{(a + b) + (a - b)p(a - b)}{2}.
$$

Then, $\tilde{p}(a, b)$ should satisfy the following max function error condition for the precision parameter $\alpha$:

$$
|\tilde{p}(a, b) - \max(a, b)| \leq 2^{-\alpha} \text{ for any } a, b \in [0, 1]. \quad (2)
$$

Since we have $\min(a, b) = a + b - \max(a, b)$, the approximate polynomial for the min function, $\tilde{p}(a, b)$, can be also easily obtained, that is, $\tilde{p}(a, b) = a + b - \tilde{p}(a, b)$.

The previous homomorphic max function MinimaxMax in (9) uses the set of degrees of component polynomials obtained by executing ComputeMinMultDegs algorithm in Algorithm 4 for inputs $\alpha, \zeta \cdot 2^{-\alpha}$, and $D - 1$, where $\zeta$ is a max function factor that can be determined experimentally. The proposed algorithm in Algorithm 11 improves the previous MinimaxMax algorithm, and we obtain the set of degrees using the ComputeMinTimeDegs algorithm instead of ComputeMinMultDegs algorithm.

In addition, authors in (1) proposed a method to approximate the ReLU function precisely using the approximation of sign function. This precise approximation of the ReLU function is necessary to evaluate pre-trained convolutional neural networks on FHE. The ReLU and sign function have the following relationship:

$$
\text{ReLU}(x) = \frac{x + x \text{sgn}(x)}{2}.
$$

Thus, approximate polynomial $r(x)$ for the ReLU function can be implemented using the approximate polynomial $p(x)$ for the sign function as follows:

$$
r(x) = \frac{x + xp(x)}{2}. \quad (3)
$$

Then, $r(x)$ should satisfy the following ReLU function error condition for the precision parameter $\alpha$:

$$
|r(x) - \text{ReLU}(x)| \leq 2^{-\alpha} \text{ for any } x \in [-1, 1]. \quad (4)
$$

The previous ReLU function algorithm that uses the equation in (3) can be described as Algorithm 12, which we call MinimaxReLU. The proposed ReLU function algorithm in Algorithm 13 improves the previous ReLU function algorithm MinimaxReLU, and we obtain the set of degrees using the ComputeMinTimeDegs algorithm instead of ComputeMinMultDegs algorithm. It should be noted that the ReLU function algorithm uses the same value of max function factor $\zeta$ as the max function algorithm for a given precision parameter $\alpha$.

As explained in Section III, the proposed AppIME algorithm makes it possible to perform ComputeIM algorithm for $d_{\text{max}} = 63$, which enables obtaining a better set of degrees of component polynomials using ComputeIMMultDegs. Table 4 presents the optimal set of degrees $M_{\text{degs}}$ for max/ReLU functions and the corresponding minimum depth consumption $D_{\text{min}}$ for $d_{\text{max}} = 31$ and $d_{\text{min}} = 63$. From Table 4, it can be seen that the depth consumption is reduced by one when $\alpha$ is 16, 17, or 18, enabling one more non-scalar multiplication per homomorphic max or ReLU function.
Algorithm 12: MinimaxReLU($x; \alpha, \zeta, D, \eta$) [17]

Input: Inputs $x \in [-1, 1]$, precision parameter $\alpha$, max function factor $\zeta$, depth consumption $D$, and margin $\eta$

Output: Approximate value of ReLU($x$)

1. $M_{\text{degs}} = \{d_1, d_2, \ldots, d_k\} \leftarrow \text{ComputeMinMultDegs}(\alpha, \zeta \cdot 2^{-\alpha}, D - 1)$
2. $p_1 \leftarrow \text{MP}(R_{1-\epsilon, 1}; d_1)$
3. $\tau_1 \leftarrow \text{ME}(\tilde{R}_{1-\epsilon, 1}; d_1) + \eta$
4. for $i \leftarrow 2$ to $k$ do
   5. $p_i \leftarrow \text{MP}(R_{\tau_{i-1}, 1}; d_i)$
   6. $\tau_i \leftarrow \text{ME}(R_{\tau_{i-1}, 1}; d_i) + \eta$
7. end
8. return $\frac{x_p \cdot p_{k-1} \cdot \ldots \cdot p_1(x) + x}{2}$

Algorithm 13: OptMinimaxReLU($x; \alpha, \zeta, D, \eta$)

Input: Inputs $x \in [-1, 1]$, precision parameter $\alpha$, max function factor $\zeta$, depth consumption $D$, and margin $\eta$

Output: Approximate value of ReLU($x$)

1. $M_{\text{degs}} = \{d_1, d_2, \ldots, d_k\} \leftarrow \text{ComputeMinTimeDegs}(\alpha, \zeta \cdot 2^{-\alpha}, D - 1)$
2. $p_1 \leftarrow \text{MP}(R_{1-\epsilon, 1}; d_1)$
3. $\tau_1 \leftarrow \text{ME}(\tilde{R}_{1-\epsilon, 1}; d_1) + \eta$
4. for $i \leftarrow 2$ to $k$ do
   5. $p_i \leftarrow \text{MP}(R_{\tau_{i-1}, 1}; d_i)$
   6. $\tau_i \leftarrow \text{ME}(R_{\tau_{i-1}, 1}; d_i) + \eta$
7. end
8. return $\frac{x_p \cdot p_{k-1} \cdot \ldots \cdot p_1(x) + x}{2}$

TABLE 4. The optimal set of degrees $M_{\text{degs}}$ and corresponding minimum depth consumption for the homomorphic max/ReLU function for $d_{\text{max}} = 31$ and $d_{\text{max}} = 63$. $D_{\text{min}}$ is the minimum depth consumption for the homomorphic max and ReLU function.

| $\alpha$ | $\zeta$ | the optimal set of degrees from $\text{ComputeMinTimeDegs}(\alpha, \zeta \cdot 2^{-\alpha}, D_{\text{min}} - 1)$ | maximum degree $d_{\text{max}} = 31$ | maximum degree $d_{\text{max}} = 63$ |
|---|---|---|---|---|
| 4 | 5 | [5] | 4 | [5] | 4 |
| 5 | 6 | [13] | 5 | [13] | 5 |
| 6 | 7 | [3,7] | 6 | [3,7] | 6 |
| 7 | 8 | [7,7] | 7 | [7,7] | 7 |
| 8 | 9 | [7,15] | 8 | [7,15] | 8 |
| 9 | 10 | [15,13] | 9 | [15,13] | 9 |
| 10 | 11 | [7,13] | 10 | [7,13] | 10 |
| 11 | 12 | [7,27] | 11 | [7,27] | 11 |
| 12 | 13 | [7,27] | 12 | [7,27] | 12 |
| 13 | 14 | [7,15,27] | 13 | [7,15,27] | 13 |
| 14 | 15 | [15,15,27] | 14 | [15,15,27] | 14 |
| 15 | 16 | [15,27,29] | 15 | [15,27,29] | 15 |
| 16 | 17 | [29,29,29] | 16 | [29,29,29] | 16 |
| 17 | 18 | [7,7,15,15] | 17 | [7,7,15,15] | 17 |
| 18 | 19 | [7,7,15,15] | 18 | [7,7,15,15] | 18 |
| 19 | 20 | [7,7,15,15,15] | 19 | [7,7,15,15,15] | 19 |

Furthermore, the proposed OptMinimaxMax and OptMinimaxReLU algorithms minimize the running time on the RNS-CKKS scheme for a given depth consumption $D$ by using ComputeMinTimeDegs algorithm instead of ComputeMinMultDegs algorithm.

VI. NUMERICAL RESULTS

In this section, numerical results of the proposed OptMinimaxComp, OptMinimaxMax, and OptMinimaxReLU algorithms in Algorithms 10, 11, and 13 respectively, are presented. The performances of OptMinimaxComp, OptMinimaxMax, and OptMinimaxReLU algorithms using the proposed ComputeMinTimeDegs algorithm are evaluated and compared with those of MinimaxComp, MinimaxMax, and MinimaxReLU algorithms using the previous ComputeMinMultDegs algorithm. The numerical analyses are conducted using the representative RNS-CKKS scheme library SEAL [12] on Intel Core i7-10700 CPU at 2.90GHz in single thread with an Ubuntu 20.04 LTS distribution.

A. PARAMETER Setting

The precision parameters $\epsilon$ and $\alpha$ are the input and output precisions of the homomorphic comparison algorithm MinimaxComp or OptMinimaxComp in the RNS-CKKS scheme. We set $\epsilon = 2^{-\alpha}$, which implies that the input and output precisions are the same. On the other hand, the homomorphic max function algorithms MinimaxMax and OptMinimaxMax and the homomorphic ReLU function algorithms MinimaxReLU and OptMinimaxReLU only use input precision parameter $\alpha$. We set $N = 2^{16}$, MinimaxComp, MinimaxMax, MinimaxReLU, OptMinimaxComp, OptMinimaxMax, or OptMinimaxReLU is performed simultaneously for $N/2$ tuples of real numbers. Then, the amortized running time is obtained by dividing the running time by $N/2$.

1) Scaling Values and Margins

We use the scaled Chebyshev polynomials $\tilde{T}_i(t) = T_i(t/w)$ for a scaling value $w > 1$ as basis polynomials. The scaled Chebyshev polynomials can be computed using the following recursion:

$$\tilde{T}_0(x) = 1$$
$$\tilde{T}_1(x) = x/w$$
$$\tilde{T}_{i+j}(x) = 2\tilde{T}_i(x)\tilde{T}_j(x) - \tilde{T}_{i-j}(x) \text{ for } i \geq j \geq 1.$$ 

The scaling values $w$ and margins $\eta$ are obtained experimentally. The obtained scaling values and margins used in our numerical analyses on homomorphic comparison operation and homomorphic max/ReLU functions are shown in Tables 5 and 6 respectively.

2) Scaling Factor

If the output of the homomorphic comparison operation or homomorphic max function for one input tuple of two real numbers $a$ and $b$ does not satisfy the comparison operation error condition in (1) or the max function error condition
in \( \alpha \) respectively, it is said to be failed. In addition, if the output of the homomorphic ReLU function for one input real number \( x \) does not satisfy the ReLU function error condition in (4), it is said to fail. The homomorphic comparison operation, max function, or ReLU function is performed for \( 2^{15} \) inputs for each \( \alpha \), and the number of failures is obtained. Then, the failure rate is the number of failures divided by the total number of inputs, \( 2^{15} \). We set the scaling factor large enough so that the homomorphic comparison operation, max function, or ReLU function does not fail in any slot, and the failure rate is said to be less than \( 2^{-15} \) in this case. We set the scaling factor \( \Delta = 2^{15} \) in all our numerical analyses, and the number of failures is zero in all of the numerical results.

3) Bases with Prime Numbers

Bases with prime numbers \( B = \{p_0, p_1, \ldots, p_{k-1}\} \) and \( C = \{q_0, q_1, \ldots, q_L\} \) should be selected. We set \( k = 1 \) and \( p_0 \approx 2^{60} \). In the numerical analysis for the homomorphic comparison operation that consumes \( D \) depth, we set the maximum level \( L = D \). We set \( q_0 \approx 2^{60} \) and \( q_j \approx \Delta \) for \( 1 \leq j \leq L \).

### B. PERFORMANCE OF THE PROPOSED HOMOMORPHIC COMPARISON ALGORITHM

The previous homomorphic comparison operation uses the set of degrees \( M_{\text{degs}} \) from ComputeMinMultDegs for \( d_{\text{max}} = 31 \). On the other hand, the proposed homomorphic comparison operation obtains \( M_{\text{degs}} \) for \( d_{\text{max}} = 63 \) from ComputeMinTimeDegs. The depth consumption \( D \) should satisfy \( D \geq M_{\text{degs}} \), where \( M_{\text{degs}} \) is the minimum depth consumption obtained from ComputeMinDep algorithm. The used sets of degrees and running times (amortized running times) of the previous homomorphic comparison algorithm MinimaxComp and the proposed algorithm OptMinimaxComp are shown in Table 7. It can be seen that the proposed homomorphic comparison algorithm reduces running time by 6% on average compared with the previous algorithm.

Increasing the depth consumption \( D \) sometimes increases the running time. In that case, the larger depth consumption than \( D \) does not need to be used, and Table 7 does not include this case. Table 7 also does not include cases when the previous and proposed algorithms use the same set of degrees \( M_{\text{degs}} \).

### C. PERFORMANCE OF THE PROPOSED HOMOMORPHIC MAX/RELU FUNCTION ALGORITHM

As in the numerical analysis of the homomorphic comparison operation, the proposed homomorphic max and ReLU function algorithms obtain \( M_{\text{degs}} \) from ComputeMinTimeDegs for \( d_{\text{max}} = 63 \). The used sets of degrees and running times (amortized running times) of the

---

**TABLE 5.** Set of scaling values and margins for the previous homomorphic comparison algorithm MinimaxComp and the proposed algorithm OptMinimaxComp.

| \( \alpha \) | depth | previous homomorphic comparison algorithm MinimaxComp | proposed homomorphic comparison algorithm OptMinimaxComp |
|-------------|-------|-----------------------------------------------------|-------------------------------------------------------|
|             |       | scaling value \( w \) | margin \( \eta \) | scaling value \( w \) | margin \( \eta \) |
| 8           | 12    | \{1,2,1,7\}               | \( 2^{-12} \) | \{1,2,1,8\}               | \( 2^{-26} \) |
| 12          | 16    | \{1,2,2,1,6\}            | \( 2^{-16} \) | \{1,2,2,1,7\}            | \( 2^{-16} \) |
| 17          | 18    | \{1,2,2,2,1,6\}         | \( 2^{-16} \) | \{1,2,2,2,1,7\}         | \( 2^{-16} \) |
| 16          | 21    | \{1,2,2,2,1,6\}         | \( 2^{-20} \) | \{1,2,2,2,1,8\}         | \( 2^{-24} \) |
| 22          | 23    | \{1,2,2,2,2,1,6\}      | \( 2^{-20} \) | \{1,2,2,2,2,1,8\}      | \( 2^{-24} \) |
| 20          | 25    | \{1,2,2,2,2,1,6\}      | \( 2^{-22} \) | \{1,2,2,2,2,1,7\}      | \( 2^{-29} \) |
| 26          | 27    | \{1,2,2,2,2,2,1,6\}    | \( 2^{-22} \) | \{1,2,2,2,2,2,1,7\}    | \( 2^{-29} \) |
| 28          |       | \{1,2,2,2,2,2,2,1,6\}  | \( 2^{-22} \) | \{1,2,2,2,2,2,2,1,6\}  | \( 2^{-29} \) |

**TABLE 6.** Set of scaling values and margins for the previous homomorphic max/ReLU function algorithms MinimaxMax/MinimaxReLU and the proposed algorithms OptMinimaxMax/OptMinimaxReLU.

| \( \alpha \) | \( \zeta \) | depth | previous homomorphic max/ReLU function algorithms MinimaxMax/MinimaxReLU | proposed homomorphic max/ReLU function algorithms OptMinimaxMax/OptMinimaxReLU |
|-------------|------------|-------|---------------------------------------------------------------|-----------------------------------------------------------------|
|             |            |       | scaling value \( w \) | margin \( \eta \) | scaling value \( w \) | margin \( \eta \) |
| 8           | 12         | 19    | \{1,2,1,6\}            | \( 2^{-12} \) | \{1,1,7\}            | \( 2^{-17} \) |
| 12          | 15         | 18    | \{1,2,1,6\}            | \( 2^{-16} \) | \{1,2,1,6\}            | \( 2^{-16} \) |
| 16          | 19         | 18    | \{1,2,2,1,6\}         | \( 2^{-20} \) | \{1,2,2,1,6\}         | \( 2^{-20} \) |
| 20          | 21         | 23    | \{1,2,2,1,6\}         | \( 2^{-24} \) | \{1,2,2,1,6\}         | \( 2^{-24} \) |
| 24          |            |       | \{1,2,2,2,2,1,6\}     | \( 2^{-24} \) | \{1,2,2,2,2,1,6\}     | \( 2^{-24} \) |
previous homomorphic max/ReLU function algorithms MinimaxMax/MinimaxReLU and the proposed algorithms OptMinimaxMax/OptMinimaxReLU are shown in Table 8. It can be seen that the proposed homomorphic max and ReLU function algorithms reduce running time by 7% and 6% on average compared with the previous homomorphic max and ReLU function algorithms, respectively. As in the numerical analysis of the homomorphic comparison operation, Table 8 does not include the cases when larger depth increases the running time or when the previous and proposed algorithms use the same set of degrees \( M_{\text{degs}} \).

### VII. CONCLUSION

We implemented the optimized homomorphic comparison, max function, and ReLU function algorithms on the RNS-CKKS scheme using a composition of minimax approximate polynomials for the first time. We successfully implemented the algorithms on the RNS-CKKS scheme with low failure rate (\(< 2^{-15}\)) and provided the parameter sets according to the precision parameter \( \alpha \). In addition, we proposed a fast algorithm for inverse minimax approximation error, which is a subroutine required to find the optimal set of degrees. This algorithm allowed us to find the optimal set of degrees for a higher maximum degree than the previous study. Finally, we proposed a method to find the set of degrees that is optimized for the RNS-CKKS scheme using the proposed fast algorithm for inverse minimax approximation error. We reduced the depth consumption of homomorphic comparison operation (resp. max/ReLU functions) by one depth when \( \alpha = 9 \) or 14 (resp. when \( \alpha \) is 16, 17, or 18). In addition, the numerical analysis demonstrated that the proposed homomorphic comparison, max function, and ReLU function algorithms reduced the running time by 6%, 7%, and 6% on average compared with the previous algorithms respectively.

### REFERENCES

[1] C. Gentry, “Fully homomorphic encryption using ideal lattices,” in *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, 2009, pp. 169–178.

[2] J. H. Cheon, A. Kim, M. Kim, and Y. Song, “Homomorphic encryption for arithmetic of approximate numbers,” in *Proceedings of International Conference on the Theory and Application of Cryptology and Information Security*, 2017, pp. 409–437.

[3] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, “Graph-HE: A graph compiler for deep learning on homomorphically encrypted data,” in *Proceedings of the 16th ACM International Conference on Computing Frontiers*, 2019, pp. 3–13.

[4] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, “ngraph-he2: A high-throughput framework for neural network inference on encrypted data,” in *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2019, pp. 45–56.

[5] X. Jiang, M. Kim, K. Lauter, and Y. Song, “Secure outsourced matrix computation and application to neural networks,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1209–1222.
[6] F. Bourse, M. Minelli, M. Minhöld, and P. Paillier, “Fast homomorphic evaluation of deep discretized neural networks,” in Proceedings of Annual International Cryptology Conference, 2018, pp. 483–512.

[7] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, “A full rns variant of approximate homomorphic encryption,” in Proceedings of International Conference on Selected Areas in Cryptography, 2018, pp. 347–368.

[8] Y. Lee, J.-W. Lee, Y.-S. Kim, and J.-S. No, “Near-optimal polynomial for modulus reduction using l2-norm for approximate homomorphic encryption,” IEEE Access, vol. 8, pp. 144 321–144 330, 2020.

[9] E. Lee, J.-W. Lee, J.-S. No, and Y.-S. Kim, “Minimax approximation of sign function by composite polynomial for homomorphic comparison,” IEEE Transactions on Dependable and Secure Computing, accepted for publication, 2021.

[10] J.-P. Bossuau, C. Mouchet, J. Troncoso-Pastoriza, and J.-P. Hubaux, “Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys,” in International Conference on the Theory and Applications of Cryptographic Techniques, 2021, pp. 587–617.

[11] Y. Lee, J.-W. Lee, Y.-S. Kim, H. Kang, and J.-S. No, “High-precision approximate homomorphic encryption by error variance minimization,” Cryptol. ePrint Arch., Tech. Rep. 2020/834, 2021.

[12] “Microsoft SEAL (release 3.5),” https://github.com/Microsoft/SEAL (Apr 2020), microsoft Research, Redmond, WA.

[13] “Lattice cryptography library (release 1.10.4),” https://palisade-crypto.org/ (Sep 2020).

[14] “Lattigo v2.2.0,” Online: http://github.com/ldsec/lattigo, Jul. 2021, ePFL-LDS.

[15] C. Cortes and V. Vapnik, “Support-vector networks,” Machine learning, vol. 20, no. 3, pp. 273–297, 1995.

[16] J. A. Hartigan and M. A. Wong, “Ak-means clustering algorithm,” Journal of the Royal Statistical Society: Series C (Applied Statistics), vol. 28, no. 1, pp. 100–108, 1979.

[17] J. Lee, E. Lee, J.-W. Lee, Y. Kim, Y.-S. Kim, and J.-S. No, “Precise approximation of convolutional neuralnetworks for homomorphically encrypted data,” arXiv preprint arXiv:2105.10879, 2021.

[18] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim et al., “Privacy-preserving machine learning with fully homomorphic encryption for deep neural network,” arXiv preprint arXiv:2106.07729, 2021.

[19] J. H. Cheon, D. Kim, and D. Kim, “Efficient homomorphic comparison methods with optimal complexity,” in Proceedings of International Conference on the Theory and Application of Cryptology and Information Security, 2020, pp. 221–256.

[20] L. Ducas and D. Micciancio, “FHEW: bootstrapping homomorphic encryption in less than a second,” in Proceedings of International Conference on the Theory and Applications of Cryptographic Techniques, 2015, pp. 617–640.

[21] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, “TFHE: fast fully homomorphic encryption over the torus,” Journal of Cryptology, vol. 33, no. 1, pp. 34–91, 2020.

[22] J. H. Cheon, D. Kim, H. H. Lee, and K. Lee, “Numerical method for comparison on homomorphically encrypted numbers,” in Proceedings of International Conference on the Theory and Application of Cryptology and Information Security, 2019, pp. 415–445.

[23] I. Chillotti, D. Ligier, J.-B. Orfila, and S. Tap, “Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE,” Cryptol. ePrint Arch., Tech. Rep. 2021/729, 2021.

[24] K. Khuziemaki and L. Schild, “FDBF: Full domain functional bootstrapping towards practical fully homomorphic encryption,” arXiv preprint arXiv:2109.02731, 2021.

[25] Z. Liu, D. Micciancio, and Y. Polyakov, “Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping,” Cryptol. ePrint Arch., Tech. Rep. 2021/1337, 2021.

[26] W.-J. Lu, Z. Huang, C. Hong, Y. Ma, and H. Qu, “PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption,” in Proceedings of IEEE Symposium on Security and Privacy (SP), 2021, pp. 1057–1073.

[27] A. Kim, A. Papadimitriou, and Y. Polyakov, “Approximate homomorphic encryption with reduced approximation error,” Cryptol. ePrint Arch., Tech. Rep. 2020/1118, vol. 2020, 2020.

[28] J.-W. Lee, E. Lee, Y.-W. Lee, and J.-S. No, “Optimal minimax polynomial approximation of modular reduction for bootstrapping of approximate homomorphic encryption,” Cryptol. ePrint Arch., Tech. Rep. 2020/5/2/2020/0083:084202, 2020.

[29] E. W. Cheney, “Introduction to approximation theory,” 1966.
JONG-SEON NO (Fellow, IEEE), received the B.S. and M.S.E.E. degrees in electronics engineering from Seoul National University, Seoul, South Korea, in 1981 and 1984, respectively, and the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in 1988. He was a Senior MTS with Hughes Network Systems, from 1988 to 1990. He was an Associate Professor with the Department of Electronic Engineering, Konkuk University, Seoul, from 1990 to 1999. He joined the faculty of the Department of Electrical and Computer Engineering, Seoul National University, in 1999, where he is currently a professor. His research interests include error-correcting codes, cryptography, sequences, LDPC codes, interference alignment, and wireless communication systems. He became an IEEE Fellow through the IEEE Information Theory Society, in 2012. He became a member of the National Academy of Engineering of Korea (NAEK), in 2015, where he served as the Division Chair of Electrical, Electronic, and Information Engineering from 2019 to 2020. He was a recipient of the IEEE Information Theory Society Chapter of the Year Award, in 2007. From 1996 to 2008, he served as the Founding Chair of the Seoul Chapter of the IEEE Information Theory Society. He was the General Chair of Sequence and Their Applications 2004 (SETA 2004), Seoul. He also served as the General Co-Chair of the International Symposium on Information Theory and Its Applications 2006 (ISITA 2006) and the International Symposium on Information Theory, 2009 (ISIT 2009), Seoul. He served as the Co-Editor-in-Chief for the IEEE JOURNAL OF COMMUNICATIONS AND NETWORKS, from 2012 to 2013.

***