Optimization of Homomorphic Comparison Algorithm on RNS-CKKS Scheme

Eunsang Lee, Joon-Woo Lee, Young-Sik Kim, Jong-Seon No

***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. However, homomorphic comparison, max function, and ReLU function algorithms that use this approximation method have not yet been successfully implemented on the residue number system variant CKKS (RNS-CKKS) scheme, and the sets of degrees of the component polynomials used by the algorithms are not optimized for the RNS-CKKS scheme. In this paper, we propose 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 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 α. We reduce the depth consumption of the homomorphic comparison, max function, and ReLU function algorithms by one depth for several α. In addition, the numerical analysis demonstrates that the proposed homomorphic comparison, max function, and ReLU function algorithms reduce running time by 6%, 7%, and 6% on average compared with the previous best-performing algorithms, 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, and Gentry proposed the first FHE scheme using bootstrapping in [1]. 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 (RNS) 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 $\operatorname{comp}(a,b)$, 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 $\operatorname{sgn}(x)$, that is,

$$\begin{aligned}\operatorname{comp}(a,b) &= \frac{1}{2}(\operatorname{sgn}(a-b)+1),\\ \max(a,b) &= \frac{1}{2}(a+b+(a-b)\operatorname{sgn}(a-b)),\\ \operatorname{ReLU}(x) &= \frac{1}{2}(x+x\operatorname{sgn}(x)),\end{aligned}$$

where $\operatorname{sgn}(x) = x/|x|$ 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 $\operatorname{sgn}(x)$ using the composition of component

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). (Corresponding author: Young-Sik Kim.)

E.S. Lee, J.-W. Lee, and J.-S. No are with the Department of Electrical and Computer Engineering, INMC, Seoul National University, Seoul 08826, South Korea (e-mail: eslee3209@ccl.snu.ac.kr; joonwoo3511@ccl.snu.ac.kr; jsno@snu.ac.kr).

Y.-S. Kim is with the Department of Information and Communication Engineering, Chosun University, Gwangju 61452, South Korea (e-mail: iamyskim@Chosun.ac.kr).

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 $\mathrm{sgn}(x)$ with the minimum number of non-scalar multiplications using a composition of minimax approximate polynomials.

Although evaluation of sign function on the CKKS scheme has been studied well [9], [19], there is no study of sign function evaluation on the RNS-CKKS scheme yet. First, since the rescaling error is somewhat large in the RNS-CKKS scheme, unlike in the CKKS scheme, it is required to study the homomorphic comparison operation and max function on the RNS-CKKS scheme that deal with this somewhat large rescaling error and achieve low approximation failure rates. In addition, although the best-performing homomorphic comparison or max function algorithms on the CKKS scheme [9] uses the degrees of the component polynomials that minimize the number of non-scalar multiplications, it does not minimize the running time on the RNS-CKKS scheme. That is, because the running time of a non-scalar multiplication changes with the current ciphertext level on the RNS-CKKS scheme, the minimization of the number of non-scalar multiplications does not necessarily correspond to that of running time, unlike the CKKS scheme. Thus, further research on the degrees of the component polynomials optimized for the RNS-CKKS scheme will improve the performance 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 on the RNS-CKKS scheme with a low failure rate ($< 2^{-15}$) by applying the scaling factor management technique [20] and using 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 α is 9 or 14 (resp. when α 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).
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 algorithm for inverse minimax approximation error. That is, we propose an algorithm that finds the optimal set of degrees with the minimum running time itself instead of the number of non-scalar multiplications. Using the optimal set of degrees 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 best-performing algorithms up to now on the RNS-CKKS scheme library SEAL [12].

B. 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 $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ and $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$ be the polynomial rings, where N is a power-of-two integer. Let $\mathcal{C} = \{q_0, q_1, \cdots, q_{\ell-1}\}$ be the set of positive integers that are coprime each other. Then, for $a \in \mathbb{Z}_Q$, where $\mathbb{Z}_Q$ is the set of integers modulo Q and $Q = \prod_{i=0}^{\ell-1} q_i$, we denote the RNS representation of a with regard to $\mathcal{C}$ by $[a]_{\mathcal{C}} = ([a]_{q_0}, \cdots, [a]_{q_{\ell-1}}) \in \mathbb{Z}_{q_0} \times \cdots \times \mathbb{Z}_{q_{\ell-1}}$. For the set of real numbers $\mathbb{R}$ and the set of complex numbers $\mathbb{C}$, a field isomorphism $\bar{\tau} : \mathbb{R}[X]/(X^N + 1) \to \mathbb{C}^{N/2}$ is defined by $\bar{\tau} : r(X) \mapsto (r(\bar{\zeta}^{5^j}))_{0 \le j < N/2}$, where $\bar{\zeta} = \exp(-\pi i/N)$ is a $(2N)$-th root of unity in $\mathbb{C}$. $\mathcal{HWT}_N(h)$ 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 $\tilde{R}_{a,b}$. In particular, if $a = 1 - \tau$ and $b = 1 + \tau$ for some $\tau \in (0, 1)$, then $\tilde{R}_{a,b} = \tilde{R}_{1-\tau,1+\tau}$ is denoted by R_τ. $|\{(n_1, n_2, \cdots, n_i); S(n_1, \cdots, n_i)\}|$ denotes the number of tuples $(n_1, \cdots, n_i)$ such that the statement $S(n_1, \cdots, n_i)$ is true. $\alpha_{\max}, \ell_{\max}, m_{\max}, n_{\max}$, and $t_{\max}$ denote the upper-bound of precision α, 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 $\alpha_{\max} = 20, \ell_{\max} = 30, m_{\max} = 70, n_{\max} = 40$, and $t_{\max} = 240$ in this paper. $d_{\max}$ denotes the upper-bound of degrees of component polynomials, and $d_{\max}$ 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 $\mathcal{B} = \{p_0, \cdots, p_{k-1}\}$, $\mathcal{C} = \{q_0, \cdots, q_{\ell-1}\}$, and $\mathcal{D} = \{p_0, \cdots, p_{k-1}, q_0, \cdots, q_{\ell-1}\}$, where p_i and q_j are all distinct primes.

– $\underline{\texttt{Conv}_{\mathcal{C}\to\mathcal{B}}}$: For $[a]_\mathcal{C} = (a^{(0)}, a^{(1)}, \cdots, a^{(\ell-1)}) \in \mathbb{Z}_{q_0} \times \cdots \times \mathbb{Z}_{q_{\ell-1}}$, output

$$\texttt{Conv}_{\mathcal{C}\to\mathcal{B}}([a]_\mathcal{C}) = \left(\sum_{j=0}^{\ell-1} [a^{(j)} \cdot \hat{q}_j^{-1}]_{q_j} \cdot \hat{q}_j \bmod p_i\right)_{0\le i<k},$$

where $\hat{q}_j = \prod_{j'\neq j} q_{j'} \in \mathbb{Z}$. This algorithm over integers $\texttt{Conv}_{\mathcal{C}\to\mathcal{B}}(\cdot) : \prod_{j=0}^{\ell-1} \mathbb{Z}_{q_j} \to \prod_{i=0}^{k-1} \mathbb{Z}_{p_i}$ can be extended to an algorithm over the polynomial rings as $\texttt{Conv}_{\mathcal{C}\to\mathcal{B}}(\cdot) : \prod_{j=0}^{\ell-1} \mathcal{R}_{q_j} \to \prod_{i=0}^{k-1} \mathcal{R}_{p_i}$ by applying it coefficient-wise.

– $\underline{\texttt{ModUp}_{\mathcal{C}\to\mathcal{D}}}$: For $[a]_\mathcal{C} \in \prod_{j=0}^{\ell-1} \mathcal{R}_{q_j}$, output

$$(\texttt{Conv}_{\mathcal{C}\to\mathcal{B}}([a]_\mathcal{C}), [a]_\mathcal{C}) \in \prod_{i=0}^{k-1} \mathcal{R}_{p_i} \times \prod_{j=0}^{\ell-1} \mathcal{R}_{q_j}.$$

– $\underline{\texttt{ModDown}_{\mathcal{D}\to\mathcal{C}}}$: For $([a]_\mathcal{B}, [b]_\mathcal{C}) \in \prod_{i=0}^{k-1} \mathcal{R}_{p_i} \times \prod_{j=0}^{\ell-1} \mathcal{R}_{q_j}$, output

$$([b]_\mathcal{C} - \texttt{Conv}_{\mathcal{B}\to\mathcal{C}}([a]_\mathcal{B})) \cdot [P^{-1}]_\mathcal{C},$$

where $P = \prod_{i=0}^{k-1} p_i$.

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

– $\underline{\texttt{Setup}(\lambda; \Delta, L)}$: For a security parameter λ, a scaling factor Δ, and the number of levels L (also called the maximum level), we set some parameters. The polynomial degree N of $\mathcal{R}$ is chosen so that the number of levels L can be supported with the security λ. A secret key distribution $\chi_{\textsf{key}}$, an error distribution $\chi_{\textsf{err}}$ over $\mathcal{R}$, and an encryption key distribution $\chi_{\textsf{enc}}$ are chosen according to the security parameter λ. Bases with prime numbers $\mathcal{B} = \{p_0, p_1, \cdots, p_{k-1}\}$ and $\mathcal{C} = \{q_0, q_1, \cdots, q_L\}$ are selected so that $p_i \equiv 1 \bmod 2N$ for $0 \le i \le k-1$ and $q_j \equiv 1 \bmod 2N$ for $0 \le j \le L$. q_0 is usually set close to 2^{60}, and $q_j - \Delta$ are as small as possible for $1 \le j \le L$. All prime numbers are distinct. Let $\mathcal{D} = \mathcal{B} \cup \mathcal{C}$. Let $\mathcal{C}_\ell = \{q_0, q_1, \cdots, q_\ell\}$ and $\mathcal{D}_\ell = \mathcal{B} \cup \mathcal{C}_\ell$ for $0 \le \ell \le L$. Let $P = \prod_{i=0}^{k-1} p_i$ and $Q = \prod_{j=0}^{L} q_j$. Let $\hat{p}_i = \prod_{0\le i'\le k-1, i'\neq i} p_{i'}$ for $0 \le i \le k-1$, and $\hat{q}_{\ell,j} = \prod_{0\le j'\le \ell, j'\neq j} q_{j'}$ for $0 \le j \le \ell \le L$. Then, the following numbers are computed as:
  - $[P^{-1}]_{q_j}$ for $0 \le j \le L$
  - $[\hat{p}_i]_{q_j}$ and $[\hat{p}_i^{-1}]_{p_i}$ for $0 \le i \le k-1$, $0 \le j \le L$
  - $[\hat{q}_{\ell,j}]_{p_i}$ and $[\hat{q}_{\ell,j}^{-1}]_{q_j}$ for $0 \le i \le k-1$, $0 \le j \le \ell \le L$

– $\underline{\texttt{Ecd}(\mathbf{z}; \Delta)}$: For $\mathbf{z} \in \mathbb{C}^{N/2}$, output $\lfloor \bar{\tau}^{-1}(\Delta\mathbf{z}) \rceil \in \mathcal{R}$.

– $\underline{\texttt{Dcd}(m; \Delta)}$: For $m \in \mathcal{R}$, output $\Delta^{-1} \cdot \bar{\tau}(m) \in \mathbb{C}^{N/2}$.

– $\underline{\texttt{KSGen}(s_1, s_2)}$: This algorithm generates the switching key for switching the secret key s_1 to s_2. First, sample $(a'^{(0)}, \cdots, a'^{(k+L)}) \leftarrow U(\prod_{i=0}^{k-1} \mathcal{R}_{p_i} \times \prod_{j=0}^{L} \mathcal{R}_{q_j})$ and $e' \leftarrow \chi_{\textsf{err}}$. Then, for given $s_1, s_2 \in \mathcal{R}$, output the switching key $\textsf{swk} = (\textsf{swk}^{(0)} = (b'^{(0)}, a'^{(0)}), \cdots, \textsf{swk}^{(k+L)} = (b'^{(k+L)}, a'^{(k+L)})) \in \prod_{i=0}^{k-1} \mathcal{R}_{p_i}^2 \times \prod_{j=0}^{L} \mathcal{R}_{q_j}^2$, where $b'^{(i)} \leftarrow -a'^{(i)} \cdot s_2 + e' \bmod p_i$ for $0 \le i \le k-1$ and $b'^{(k+j)} \leftarrow -a'^{(k+j)} \cdot s_2 + [P]_{q_j} \cdot s_1 + e' \bmod q_j$ for $0 \le j \le L$.

– $\underline{\texttt{KeyGen}(\lambda)}$: This algorithm generates the secret key, public key, and the evaluation key. First, sample $s \leftarrow \chi_{\textsf{key}}$ and set the secret key $\textsf{sk} \leftarrow (1, s)$. Sample $(a^{(0)}, \cdots, a^{(L)}) \leftarrow U(\prod_{j=0}^{L} \mathcal{R}_{q_j})$ and $e \leftarrow \chi_{\textsf{err}}$. Then, the public key is $\textsf{pk} \leftarrow (\textsf{pk}^{(j)} = (b^{(j)}, a^{(j)}) \in \mathcal{R}_{q_j}^2)_{0\le j\le L}$, where $b^{(j)} \leftarrow -a^{(j)} \cdot s + e \bmod q_j$ for $0 \le j \le L$. The evaluation key is $\textsf{evk} \leftarrow \texttt{KSGen}(s^2, s)$.

– $\underline{\texttt{Enc}(\mathbf{z}; \textsf{pk}, \Delta)}$: For a message slot $\mathbf{z} \in \mathbb{C}^{N/2}$, compute $m = \texttt{Ecd}(\mathbf{z}; \Delta)$. Then, sample $v \leftarrow \chi_{\textsf{enc}}$ and $e_0, e_1 \leftarrow \chi_{\textsf{err}}$. Then, output the ciphertext $\textsf{ct} = (\textsf{ct}^{(j)})_{0\le j\le L} \in \prod_{j=0}^{L} \mathcal{R}_{q_j}^2$, where $\textsf{ct}^{(j)} \leftarrow v \cdot \textsf{pk}^{(j)} + (m + e_0, e_1) \bmod q_j$ for $0 \le j \le L$.

– $\underline{\texttt{Dec}(\textsf{ct}; \textsf{sk}, \Delta)}$: For a ciphertext $\textsf{ct} = (\textsf{ct}^{(j)})_{0\le j\le \ell} \in \prod_{j=0}^{\ell} \mathcal{R}_{q_j}^2$, obtain $\tilde{m} = \langle \textsf{ct}^{(0)}, \textsf{sk} \rangle \bmod q_0$. Then, output $\mathbf{z} = \texttt{Dcd}(\tilde{m}; \Delta)$.

– $\underline{\texttt{Add}(\textsf{ct}_1, \textsf{ct}_2)}$: For two ciphertexts $\textsf{ct}_r = (\textsf{ct}_r^{(j)})_{0\le j\le \ell}$ for $r = 1, 2$, output the ciphertext $\textsf{ct}_{\textsf{add}} = (\textsf{ct}_{\textsf{add}}^{(j)})_{0\le j\le \ell}$, where $\textsf{ct}_{\textsf{add}}^{(j)} \leftarrow \textsf{ct}_1^{(j)} + \textsf{ct}_2^{(j)} \bmod q_j$ for $0 \le j \le \ell$.

– $\underline{\texttt{Mult}(\textsf{ct}_1, \textsf{ct}_2; \textsf{evk})}$: For two ciphertexts $\textsf{ct}_r = (\textsf{ct}_r^{(j)} = (c_{r0}^{(j)}, c_{r1}^{(j)}))_{0\le j\le \ell}$ for $r = 1, 2$, compute the followings:
  - $d_0^{(j)} = c_{00}^{(j)} c_{10}^{(j)} \bmod q_j$, $d_1^{(j)} = c_{00}^{(j)} c_{11}^{(j)} + c_{01}^{(j)} c_{10}^{(j)} \bmod q_j$, and $d_2^{(j)} = c_{01}^{(j)} c_{11}^{(j)} \bmod q_j$ for $0 \le j \le \ell$.
  - $\texttt{ModUp}_{C_\ell \to D_\ell}(d_2^{(0)}, \cdots, d_2^{(\ell)}) = (\tilde{d}_2^{(0)}, \cdots, \tilde{d}_2^{(k-1)}, d_2^{(0)}, \cdots, d_2^{(\ell)})$.
  - $\tilde{ct} = (\tilde{ct}^{(k+\ell)} = (\tilde{c}_0^{(j)}, \tilde{c}_1^{(j)}))_{0\le j\le k+\ell}$, where $\tilde{ct}^{(i)} = \tilde{d}_2^{(i)} \cdot \textsf{evk}^{(i)} \bmod p_i$ and $\tilde{ct}^{(k+j)} = d_2^{(j)} \cdot \textsf{evk}^{(k+j)} \bmod q_j$ for $0 \le i \le k-1$ and $0 \le j \le \ell$.
  - $(\hat{c}_r^{(0)}, \cdots, \hat{c}_r^{(\ell)}) = \texttt{ModDown}_{D_\ell \to C_\ell}(\tilde{c}_r^{(0)}, \cdots, \tilde{c}_r^{(k+\ell)})$ for $r = 0, 1$.
  - $\textsf{ct}_{\textsf{mult}} = (\textsf{ct}_{\textsf{mult}}^{(j)})_{0\le j\le \ell}$, where $\textsf{ct}_{\textsf{mult}}^{(j)} = (\hat{c}_0^{(j)} + d_0^{(j)}, \hat{c}_1^{(j)} + d_1^{(j)}) \bmod q_j$ for $0 \le j \le \ell$.

  Then, output the ciphertext $\textsf{ct}_{\textsf{mult}}$.

– $\underline{\texttt{RS}(\textsf{ct})}$: For a ciphertext $\textsf{ct} = (\textsf{ct}^{(j)} = (c_0^{(j)}, c_1^{(j)}))_{0\le j\le \ell}$, output the ciphertext $\textsf{ct}' = (\textsf{ct}'^{(j)} = (c'^{(j)}_0, c'^{(j)}_1))_{0\le j\le \ell-1}$, where $c'^{(j)}_r = q_\ell^{-1} \cdot (c_r^{(j)} - c_r^{(\ell)}) \bmod q_j$ for $r = 0, 1$ and $0 \le j \le \ell - 1$.

In this paper, we set the key distribution $\chi_{\textsf{key}} = \mathcal{HWT}_N(256)$, which samples an element in $\mathcal{R}$ with ternary coefficients that have 256 nonzero values uniformly at random.

C. Scaling Factor Management

A technique of eliminating the large rescaling error in the RNS-CKKS scheme was proposed in [20], where different scaling factors in different levels were used instead of using the same scaling factor for each level. If the maximum level is L, and the ciphertext modulus for level i is q_i, the scaling factor for each level is set as follows: $\Delta_L = q_L$ and $\Delta_i = \Delta_{i+1}^2 / q_{i+1}$ for $i = 0, \cdots, L-1$.

When two ciphertexts at the same level are multiplied homomorphically, they do not introduce approximate rescaling error. Then, we consider when two ciphertexts are in the different levels: levels i and j such that $i > j$. In this case, the moduli $q_i, q_{i-1}, \cdots, q_{j+1}$ in the first ciphertext

TABLE I
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 [10] AND THE ODD BABY-STEP GIANT-STEP ALGORITHM [21].

polynomial degree d	depth consumption $\mathsf{dep}(d)$	multiplications $\mathsf{mult}(d)$
3	2	2
5	3	3
7	3	5
9	4	5
11	4	6
13	4	7
15	4	8
17	5	8
19	5	8
21	5	9
23	5	9
25	5	10
27	5	10
29	5	11
31	5	12
33	6	11
35	6	11
37	6	11
39	6	11
41	6	12
43	6	12
45	6	13
47	6	13
49	6	14
51	6	14
53	6	14
55	6	14
57	6	15
59	6	15
61	6	16
63	6	17

are dropped, and then the first ciphertext is multiplied by a constant $\lfloor \frac{\Delta_j q_{j+1}}{\Delta_i} \rceil$. Then, we rescale the first ciphertext by q_{j+1}. Since both ciphertexts are now at the same level, conventional homomorphic multiplication can be performed. Also, the approximate rescaling error is decreased in this way.

D. Homomorphic Comparison Operation using Minimax Composite Polynomial

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

The minimax approximate polynomial of degree at most d on D for $\mathrm{sgn}(x)$ is denoted by $\mathsf{MP}(D; d)$. In addition, for the minimax approximate polynomial $p(x) = \mathsf{MP}(D; d)$, the minimax approximation error $\max_D \|p(x) - \mathrm{sgn}(x)\|_\infty$ is denoted by $\mathsf{ME}(D; d)$. It is known that for any continuous function f on D, the minimax approximate polynomial of degree at most d on D is unique [22]. Furthermore, the minimax approximate polynomial can be obtained using the improved multi-interval Remez algorithm [23].

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

- $p_1 = \mathsf{MP}(D; d_1)$
- $p_i = \mathsf{MP}(p_{i-1} \circ p_{i-2} \circ \cdots \circ p_1(D); d_i)$ for $i, 2 \le i \le k$.

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

Algorithm 1: $\texttt{AppIMEbinary}(\tau'; d, \bar{i})$ [9]

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

Algorithm 2: $\texttt{ComputehG}(\tau)$ [9]

```
Input: An input $\tau$ and an odd maximum degree $d_{\max}$
Output: 2-dimensional tables $\tilde{h}$ and $\tilde{G}$
Generate 2-dimensional tables $\tilde{h}$ and $\tilde{G}$, both of which
  have size of $(m_{\max} + 1) \times (n_{\max} + 1)$.
for $m \leftarrow 0$ to $m_{\max}$ do
    for $n \leftarrow 0$ to $n_{\max}$ do
        if $m \le 1$ or $n \le 1$ then
            $\tilde{h}(m, n) \leftarrow \tau$
            $\tilde{G}(m, n) \leftarrow \phi$
        else
            $j \leftarrow \underset{\substack{1\le i\le \frac{d_{\max}-1}{2} \\ \mathsf{mult}(2i+1)\le m \\ \mathsf{dep}(2i+1)\le n}}{\operatorname{argmax}} \mathsf{IME}(\tilde{h}(m - \mathsf{mult}(2i+1), n - \mathsf{dep}(2i+1)); 2i+1)$
            $\tilde{h}(m, n) \leftarrow \mathsf{IME}(\tilde{h}(m - \mathsf{mult}(2j+1), n - \mathsf{dep}(2j+1)); 2j+1)$
            $\tilde{G}(m, n) \leftarrow \{2j+1\} \cup \tilde{G}(m - \mathsf{mult}(2j+1), n - \mathsf{dep}(2j+1))$
        end
    end
end
```

Algorithm 3: $\texttt{ComputeMinDep}(\alpha, \epsilon)$ [9]

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

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

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

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 α, ϵ and inputs $a, b \in [0, 1]$ is summarized as follows:

1) Obtain the minimum depth consumption M_{dep} from $\texttt{ComputeMinDep}$ algorithm in Algorithm 3.
2) Choose depth consumption $D(\geq M_{\text{dep}})$ and obtain the optimal set of degrees M_{degs} from $\texttt{ComputeMinMultDegs}$ algorithm in Algorithm 4.
3) For some appropriate margin η, perform the homomorphic comparison algorithm $\mathsf{MinimaxComp}$ in Algorithm 5 using M_{degs}.

$\texttt{ComputeMinDep}$ and $\texttt{ComputeMinMultDegs}$ algorithms use $\texttt{ComputehG}$ algorithm as a subroutine, and $\mathsf{MinimaxComp}$ algorithm uses $\texttt{ComputeMinMultDegs}$ algorithm as a subroutine. Then, the output of $\mathsf{MinimaxComp}$ algorithm, $\tilde{p}(a, b) = \frac{p_k \circ p_{k-1} \circ \cdots \circ p_1(a-b)+1}{2}$ satisfies the following comparison operation error condition:

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

The set of degrees $M_{\text{degs}} = \{d_1, \cdots, d_k\}$ obtained from the $\texttt{ComputeMinMultDegs}$ algorithm satisfies $\deg(p_i) = d_i, 1 \leq i \leq k$. M_{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 $\texttt{ComputehG}$ algorithm in Algorithm 2 [9] should be performed to obtain the optimal set of degrees from the $\texttt{ComputeMinMultDegs}$ algorithm. For performing the $\texttt{ComputehG}$ algorithm, the *inverse minimax approximation error* $\mathsf{IME}(\tau'; d)$ should be computed many times. That is, the $\texttt{AppIMEbinary}$ algorithm [9] determining an approximate value of $\mathsf{IME}(\tau'; d)$ should be called many times. However, a single call of $\texttt{AppIMEbinary}$ algorithm also requires multiple computations of $\mathsf{ME}(R_\tau; d)$, that is, several calls for the improved multi-interval Remez algorithm [23]. As a result, the $\texttt{ComputehG}$ algorithm requires significant running time. Specifically, the number of computations of $\mathsf{IME}(\tau'; d)$ in $\texttt{ComputehG}$ for each precision parameter α is given as follows:

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

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

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

$$\begin{aligned} &15(m_{\max}+1)(n_{\max}+1) - (m_{\max}+1)\cdot 64 \\ &- (n_{\max}+1)\cdot 113 + 64 \cdot 113 \\ =&15 \cdot 71 \cdot 41 - 71 \cdot 64 - 41 \cdot 113 + 64 \cdot 113 \\ =&41,720. \end{aligned}$$

If we want to use $d_{\max} = 63$, the number of computations of $\mathsf{IME}(\tau'; d)$ is $117,675$.

To obtain a precise approximate value of $\mathsf{IME}(\tau'; d)$ using the AppIMEbinary algorithm, it is required to iterate at least ten times. Then, the expected number of computations of $\mathsf{ME}(R_\tau; d)$ in ComputehG 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 α, where the input τ of ComputehG algorithm is $2^{1-\alpha}$. To perform the ComputehG algorithm for α from 4 to 20, around $1,176,750 \times 17 = 20,004,750$ calls for $\mathsf{ME}(R_\tau; d)$ are required. Because of the large number of calls for $\mathsf{ME}(R_\tau; d)$, the value of $d_{\max}$ larger than 31 could not be used in [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 $\mathsf{IME}(\tau'; d)$.

A. Improved Algorithm for Inverse Minimax Approximation Error Using Interpolation

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

1) Sample the values of τ at moderate intervals.
2) Compute the values of $\mathsf{ME}(R_\tau; d)$ for the sampled τ.
3) For $\tau' \in (0,1)$, obtain an approximate value of $\mathsf{IME}(\tau'; d)$ by interpolation using the computed sample values of $\mathsf{ME}(R_\tau; d)$.

Before describing the proposed algorithm in detail, we define $\mathsf{realtoexp}(\tau)$ and $\mathsf{exptoreal}(x)$ for $\tau \in (0,1)$ and $x \in \mathbb{R}$ as follows:

$$\mathsf{realtoexp}(\tau) = \begin{cases} \log_2(\tau), & \text{if } \tau < 0.5 \\ -\log_2(1-\tau), & \text{otherwise,} \end{cases}$$

$$\mathsf{exptoreal}(x) = \begin{cases} 2^x, & \text{if } x < 0 \\ 1 - 2^{-x}, & \text{otherwise.} \end{cases}$$

For $\alpha_{\max}$, which is the upper-bound of α, we consider sampling τ from $2^{-\alpha_{\max}-1}$ to $1 - 2^{-\alpha_{\max}-1}$. If τ is close to zero or one, sampling should be very dense. Because sampling the whole range densely from $2^{-\alpha_{\max}-1}$ to $1 - 2^{-\alpha_{\max}-1}$ requires a large number of samples, it is desirable to sample densely when τ is close to zero or one, and sparsely, otherwise.

We propose to sample α from $-\alpha_{\max} - 1$ to $\alpha_{\max} + 1$ uniformly and compute $\mathsf{ME}(R_{\mathsf{exptoreal}(\alpha)}; d)$ for the sampled α. Interpolating using these samples, we can achieve a precise approximation of $\mathsf{IME}(\tau'; d)$ with a smaller number of samples. Precision $\bar{n}$ determines how frequently the values of α are sampled, and we set $\bar{n} = 10$. For a given maximum degree $d_{\max}$ and a precision $\bar{n}$, StoreME algorithm in Algorithm 6 stores α and $\mathsf{realtoexp}(\mathsf{ME}(R_{\mathsf{exptoreal}(\alpha)}; d))$ in two-dimensional tables $\tilde{X}$ and $\tilde{Y}$, respectively, for the sampled α and various d. The number of calls for $\mathsf{ME}(R_\tau; d)$ in StoreME algorithm is $(d_{\max} - 1)(\bar{n}\alpha_{\max} + 1)$. For $\bar{n} = 10$ and $\alpha_{\max} = 20$, the number of calls for $\mathsf{ME}(R_\tau; d)$ is $6,030$ for $d_{\max} = 31$ and $12,462$ for $d_{\max} = 63$.

Algorithm 6: StoreME($d_{\max}, \bar{n}$)

```
   Input: Maximum degree $d_{\max}$ and precision $\bar{n}$
   Output: 2-dimensional tables $\tilde{X}$ and $\tilde{Y}$
 1 Generate 2-dimensional tables $\tilde{X}$ and $\tilde{Y}$, both of
     which have size of $\frac{d_{\max}-1}{2} \times 2(\bar{n}\alpha_{\max}+1)$
 2 for $i \leftarrow 0$ to $\frac{d_{\max}-3}{2}$ do
 3     for $j \leftarrow 0$ to $\bar{n}\alpha_{\max}$ do
 4         $\alpha \leftarrow -\alpha_{\max} - 1 + j/\bar{n}$
 5         $\tilde{X}(i,j) \leftarrow \alpha$
 6         $\tilde{Y}(i,j) \leftarrow \mathsf{realtoexp}(\mathsf{ME}(R_{\mathsf{exptoreal}(\alpha)}; 2i+3))$
 7     end
 8     for $j \leftarrow 0$ to $\bar{n}\alpha_{\max}$ do
 9         $\alpha \leftarrow 1 + j/\bar{n}$
10         $\tilde{X}(i, j+\bar{n}\alpha_{\max}+1) \leftarrow \alpha$
11         $\tilde{Y}(i, j+\bar{n}\alpha_{\max}+1) \leftarrow$
             $\mathsf{realtoexp}(\mathsf{ME}(R_{\mathsf{exptoreal}(\alpha)}; 2i+3))$
12     end
13 end
14 return $\tilde{X}$ and $\tilde{Y}$
```

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

Algorithm 7: $\texttt{AppIME}(\tau, d; d_{\max}, \bar{n})$

```
Input: Target maximum error $\tau$, degree $d$, the odd maximum degree $d_{\max}$, and precision $\bar{n}$
Output: An approximate value of $\mathsf{IME}(\tau; d)$
1  $\tilde{X}, \tilde{Y} \leftarrow \texttt{StoreME}(d_{\max}, \bar{n})$
2  $\tilde{\tau} \leftarrow \mathsf{realtoexp}(\tau)$
3  for $j \leftarrow 1$ to $2\bar{n}\alpha_{\max} + 1$ do
4      if $\tilde{\tau} \leq \tilde{X}(\frac{d-3}{2}, j)$ then
5          if $\tilde{X}(\frac{d-3}{2}, j-1)\tilde{X}(\frac{d-3}{2}, j) < 0$ or $\tilde{Y}(\frac{d-3}{2}, j-1)\tilde{Y}(\frac{d-3}{2}, j) < 0$ then
6              return $\tilde{X}(\frac{d-3}{2}, j)$
7          else
8              return $\tilde{X}(\frac{d-3}{2}, j-1) - (\tilde{X}(\frac{d-3}{2}, j) - \tilde{X}(\frac{d-3}{2}, j-1)) \cdot (\tilde{\tau} - \tilde{Y}(\frac{d-3}{2}, j-1))/(\tilde{Y}(\frac{d-3}{2}, j) - \tilde{Y}(\frac{d-3}{2}, j-1))$
9          end
10     end
11 end
12 return $\mathsf{exptoreal}(\tilde{X}(\frac{d-3}{2}, 2\bar{n}\alpha_{\max} + 1))$
```

TABLE II
EXPECTED RUNNING TIME OF ComputehG ALGORITHM FOR α FROM 4 TO 20 USING THE PREVIOUS AND PROPOSED ALGORITHMS FOR INVERSE MINIMAX APPROXIMATION ERROR.

	$d_{\max}$	previous method using AppIMEbinary	proposed method using AppIME
expected number of calls for $\mathsf{ME}(R_\tau; d)$	31	7,092,400	6,030
	63	20,004,750	12,462
expected running time of ComputehG for α from 4 to 20	31	2,758 hours	2 hours
	63	27,228 hours	17 hours

B. *Comparison Between the Previous and the Proposed Algorithms for Inverse Minimax Approximation Error*

We compare the running time of ComputehG 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 $d_{\max} = 31$ and about 4.9 s on average when $d_{\max} = 63$. Then, the expected running time of ComputehG can be obtained. Table II shows the expected running time of ComputehG algorithm for α from 4 to 20 using the previous and proposed algorithms for inverse minimax approximation error. It can be seen from Table II that using the proposed AppIME algorithm requires much less running time than using the previous AppIMEbinary algorithm, enabling the execution of ComputehG algorithm for $d_{\max} = 63$. 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.

TABLE III
THE OPTIMAL SETS OF DEGREES M_{degs} AND CORRESPONDING MINIMUM DEPTH CONSUMPTION FOR HOMOMORPHIC COMPARISON OPERATION FOR $d_{\max} = 31$ AND $d_{\max} = 63$. $D_{\min}$ IS THE MINIMUM DEPTH CONSUMPTION FOR THE HOMOMORPHIC COMPARISON OPERATION.

α	the optimal set of degrees from $\texttt{ComputeMinMultDegs}(\alpha, 2^{-\alpha}, D_{\min})$			
	maximum degree $d_{\max} = 31$		maximum degree $d_{\max} = 63$	
	degrees	depth	degrees	depth
4	{27}	5	{27}	5
5	{7,13}	7	{7,13}	7
6	{15,15}	8	{15,15}	8
7	{7,7,13}	10	{7,7,13}	10
8	{7,15,15}	11	{7,15,15}	11
9	{7,7,7,13}	13	{55,55}	**12**
10	{7,7,13,15}	14	{7,7,13,15}	14
11	{7,15,15,15}	15	{7,15,15,15}	15
12	{15,15,15,15}	16	{15,15,15,15}	16
13	{15,15,15,31}	17	{15,15,15,31}	17
14	{7,7,15,15,27}	19	{55,59,59}	**18**

C. *Reduction of Depth Consumption Using the Improved Algorithm for Inverse Minimax Error*

While ComputehG algorithm in Algorithm 2 could only be performed for $d_{\max} \leq 31$ in [9], we perform ComputehG algorithm for $d_{\max} \leq 63$ using the proposed AppIME algorithm in Algorithm 7. Table III lists the optimal sets of degrees M_{degs} and the corresponding minimum depth consumption $D_{\min}$ for $d_{\max} = 31$ and $d_{\max} = 63$. From Table III, it can be seen that the depth consumption is reduced by one when α is 9 or 14. That is, for $\alpha = 9$ or $\alpha = 14$, high $d_{\max}$ 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 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 [9], we study homomorphic comparison operation in the RNS-CKKS scheme, and thus there are additional considerations. Unlike the CKKS scheme, the RNS-CKKS scheme has a somewhat large rescaling error, leading to a high failure rate in the homomorphic comparison operation using minimax composite polynomial [9]. Scaling factor according to level is determined as follows: $\Delta_L = q_L$ and $\Delta_i = \Delta_{i+1}^2/q_{i+1}$ for $i = 0, \cdots, L-1$. If two ciphertexts of different levels are added or multiplied with each other, a large error can occur because the scaling factors of the two ciphertexts are not the same. We apply the scaling factor management technique proposed in [20] to the homomorphic comparison operation in the RNS-CKKS scheme. It can be seen in Section VI that a low failure rate is achieved using this technique and appropriate parameter sets.

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_{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.

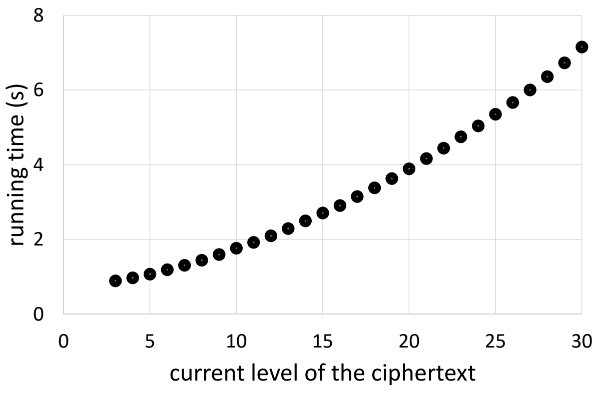

Fig. 1. The running time of a polynomial of degree seven according to the current level of the ciphertext on the RNS-CKKS scheme library SEAL [12].

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 $M_{\text{degs}} = \{7, 7, 31, 31\}$ 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 that minimizes the running time itself rather than the number of non-scalar multiplications. Specifically, we modify the previous ComputehG and ComputeMinMultDegs algorithms so that the modified algorithms can find the set of degrees M_{degs} that minimizes the running time.

Now, we define $C[i][\ell][\ell']$ for $0 \le i \le \frac{d_{\max}-3}{2}$, $0 \le \ell \le \ell_{\max}$, $0 \le \ell' \le \ell_{\max}$. First, we set up for the maximum level ℓ. Then, any polynomial of degree $2i+3$ in level ℓ' is evaluated using the optimal level consumption technique [10] and odd baby-step giant-step algorithm [21]. If t is the running time of the polynomial evaluation in milliseconds, then we define $C[i][\ell][\ell']$ as $C[i][\ell][\ell'] = \lfloor \frac{t}{100} \rceil$. Here, if $\ell < \ell'$ or $\ell' < \lceil \log_2(2i+3) \rceil$, the polynomial evaluation is infeasible, and thus, $C[i][\ell][\ell']$ is set to large enough value $100,000$ in this case. We obtain the values of $C[i][\ell][\ell']$ by performing polynomial evaluation on encrypted data, and this computation is done on a Linux PC with Intel Core i7-10700 CPU at 2.90GHZ with one thread. Then, $u_{\tau,L}(m,n)$ and $V_{\tau,L}(m,n)$ are defined recursively using the values of $C[i][\ell][\ell']$ as follows:

$$u_{\tau,L}(m,n) = \begin{cases} \tau, & \text{if } m \le 1 \text{ or } n \le 1 \\ \mathsf{IME}(u_{\tau,L}(m - C[j_{m,n}-1][L][n], n - \\ \mathsf{dep}(2j_{m,n}+1)); 2j_{m,n}+1), & \text{otherwise,} \end{cases}$$

$$V_{\tau,L}(m,n) = \begin{cases} \phi, & \text{if } m \le 1 \text{ or } n \le 1 \\ \{2j_{m,n}+1\} \cup V_{\tau,L}(m - C[j_{m,n}-1][L][n], \\ n - \mathsf{dep}(2j_{m,n}+1)), & \text{otherwise,} \end{cases}$$

where $j_{m,n} =$

$$\underset{\substack{1 \le i \\ C[i-1][L][n] \le m \\ \mathsf{dep}(2i+1) \le n}}{\operatorname{argmax}} \mathsf{IME}(u_{\tau,L}(m - C[i-1][L][n], n - \mathsf{dep}(2i+1)); 2i+1).$$

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

$$\begin{aligned} &\mathsf{MCE}(R_{\tau'}; \{d_i\}_{1 \le i \le k}) \le \tau \\ &\sum_{i=1}^{k} \mathsf{dep}(d_i) \le n \\ &\sum_{i=1}^{k} C[\frac{d_i - 3}{2}][L][L - \sum_{j=1}^{i-1} \mathsf{dep}(d_j)] \le m, \end{aligned}$$

where $\sum_{i=1}^{k} C[\frac{d_i-3}{2}][L][L - \sum_{j=1}^{i-1} \mathsf{dep}(d_j)]$ is the running time of the homomorphic comparison operation using set of degrees $M_{\text{degs}} = \{d_i\}_{1 \le i \le k}$. In addition, for $V_{\tau,L}(m,n) = \{d'_i\}_{1 \le i \le k'}$, we have $\mathsf{MCE}(R_{u_{\tau,L}(m,n)}; \{d'_i\}_{1 \le i \le k'}) \le \tau$, $\sum_{i=1}^{k'} \mathsf{dep}(d'_i) \le n$, and $\sum_{i=1}^{k} C[\frac{d_i-3}{2}][L][L - \sum_{j=1}^{i-1} \mathsf{dep}(d_j)] \le m$.

ComputeuV algorithm in Algorithm 8 outputs two-dimensional tables $\tilde{u}$ and $\tilde{V}$ that store the values of $u_{\tau,L}$ and $V_{\tau,L}$. This ComputuV algorithm requires many computations of $\mathsf{IME}(\tau'; d)$. However, these computations can be performed quickly using the proposed AppIME algorithm.

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

Now, we propose the homomorphic comparison algorithm OptMinimaxComp in Algorithm 10 that uses ComputeMinTimeDegs algorithm. This is the modified algorithm of the previous MinimaxComp algorithm in Algorithm 5 [9], minimizing the running time on the RNS-CKKS scheme for a given depth consumption D.

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.

Algorithm 8: $\texttt{ComputeuV}(\tau; L)$

```
Input: $\tau$, maximum level $L$
Output: 2-dimensional tables $\tilde{u}$ and $\tilde{V}$
1  Generate 2-dimensional tables $\tilde{u}$ and $\tilde{V}$, both of which
     have size of $(t_{\max}+1) \times (n_{\max}+1)$
2  for $m \leftarrow 0$ to $t_{\max}$ do
3      for $n \leftarrow 0$ to $n_{\max}$ do
4          if $m \le 1$ or $n \le 1$ then
5              $\tilde{u}(m,n) \leftarrow \tau$
6              $\tilde{V}(m,n) \leftarrow \phi$
7          else
8              $j \leftarrow \underset{\substack{1\le i \\ C[i-1][L][n]\le m \\ \mathsf{dep}(2i+1)\le n}}{\operatorname{argmax}} \mathsf{IME}(\tilde{u}(m-C[i-1][L][n],$
                 $n-\mathsf{dep}(2i+1)); 2i+1)$
9              $\tilde{u}(m,n) \leftarrow \mathsf{IME}(\tilde{u}(m-C[j-1][L][n],$
                 $n-\mathsf{dep}(2j+1)); 2j+1)$
10             $\tilde{V}(m,n) \leftarrow \{2j+1\} \cup \tilde{V}(m-C[j-1][L][n],$
                 $n-\mathsf{dep}(2j+1))$
11         end
12     end
13 end
```

Algorithm 9: $\texttt{ComputeMinTimeDegs}(\alpha, \epsilon, D)$

```
Input: Precision parameters $\alpha$ and $\epsilon$, and depth
       consumption $D$
Output: Minimum running time $M_{\text{time}}$ and the
        optimal set of degrees $M_{\text{degs}}$
1  $\tilde{u}$, $\tilde{V} \leftarrow \texttt{ComputeuV}(2^{1-\alpha}; D)$
2  for $j \leftarrow 0$ to $t_{\max}$ do
3      if $\tilde{u}(j,D) \ge \delta = \frac{1-\epsilon}{1+\epsilon}$ then
4          $M_{\text{time}} \leftarrow j$
5          Go to line 11
6      end
7      if $j = t_{\max}$ then
8          return $\perp$
9      end
10 end
11 $M_{\text{degs}} \leftarrow \tilde{V}(M_{\text{time}}, D)$ ;        // $M_{\text{degs}}$: ordered set
12 return $M_{\text{time}}$ and $M_{\text{degs}}$
```

A. The Proposed Homomorphic Max and ReLU Function Algorithm

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,

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

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:

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

Algorithm 10: $\texttt{OptMinimaxComp}(a, b; \alpha, \epsilon, D, \eta)$

```
Input: Inputs $a, b \in (0,1)$, precision parameters $\alpha$ and
       $\epsilon$, depth consumption $D$, and margin $\eta$
Output: Approximate value of $\mathrm{comp}(a,b)$
1  $M_{\text{degs}} = \{d_1, d_2, \cdots, d_k\} \leftarrow$
     $\texttt{ComputeMinTimeDegs}(\alpha, \epsilon, D)$
2  $p_1 \leftarrow \mathsf{MP}(\tilde{R}_{1-\epsilon,1}; d_1)$
3  $\tau_1 \leftarrow \mathsf{ME}(\tilde{R}_{1-\epsilon,1}; d_1) + \eta$
4  for $i \leftarrow 2$ to $k$ do
5      $p_i \leftarrow \mathsf{MP}(R_{\tau_{i-1}}; d_i)$
6      $\tau_i \leftarrow \mathsf{ME}(R_{\tau_{i-1}}; d_i) + \eta$
7  end
8  return $\frac{p_k \circ p_{k-1} \circ \cdots \circ p_1(a-b)+1}{2}$
```

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

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

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

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 ζ 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.

Algorithm 11: $\texttt{OptMinimaxMax}(a, b; \alpha, \zeta, D, \eta)$

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

In addition, authors in [17] 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:

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

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

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

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

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

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 ζ as the max function algorithm for a given precision parameter α.

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

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

As explained in Section III, the proposed AppIME algorithm makes it possible to perform ComputehG algorithm for $d_{\max} = 63$, which enables obtaining a better set of degrees of component polynomials using ComputeMinMultDegs.

TABLE IV
THE OPTIMAL SET OF DEGREES M_{degs} AND CORRESPONDING MINIMUM DEPTH CONSUMPTION FOR THE HOMOMORPHIC MAX/RELU FUNCTION FOR $d_{\max} = 31$ AND $d_{\max} = 63$. $D_{\min}$ IS THE MINIMUM DEPTH CONSUMPTION FOR THE HOMOMORPHIC MAX AND RELU FUNCTION.

α	ζ	the optimal set of degrees from ComputeMinMultDegs$(\alpha, \zeta \cdot 2^{-\alpha}, D_{\min} - 1)$			
		maximum degree $d_{\max} = 31$		maximum degree $d_{\max} = 63$	
		degrees	depth	degrees	depth
4	5	{5}	4	{5}	4
5	5	{13}	5	{13}	5
6	10	{3,7}	6	{3,7}	6
7	11	{7,7}	7	{7,7}	7
8	12	{7,15}	8	{7,15}	8
9	13	{15,15}	9	{15,15}	9
10	13	{7,7,13}	11	{7,7,13}	11
11	15	{7,7,27}	12	{7,7,27}	12
12	15	{7,15,27}	13	{7,15,27}	13
13	16	{15,15,27}	14	{15,15,27}	14
14	17	{15,27,29}	15	{15,27,29}	15
15	17	{29,29,29}	16	{15,27,59}	16
16	19	{7,7,7,15,15}	18	{27,29,59}	**17**
17	19	{7,7,15,15,15}	19	{29,29,59}	**18**
18	19	{7,15,15,15,15}	20	{59,59,61}	**19**

Table IV presents the optimal set of degrees M_{degs} for max/ReLU functions and the corresponding minimum depth consumption $D_{\min}$ for $d_{\max} = 31$ and $d_{\min} = 63$. From Table IV, it can be seen that the depth consumption is reduced by one when α is 16, 17, or 18, enabling one more non-scalar multiplication per homomorphic max or ReLU function.

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 Opt MinimaxReLU 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 a Linux PC with Intel Core i7-10700 CPU at 2.90GHz with one thread.

A. Parameter Setting

The precision parameters ϵ and α 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 α. 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:

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

The scaling values w and margins η 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 V and VI, 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 (2), 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 α, 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 $\Delta = 2^{50}$ 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 $\mathcal{B} = \{p_0, p_1, \cdots, p_{k-1}\}$ and $\mathcal{C} = \{q_0, q_1, \cdots, q_L\}$ 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_{degs} from ComputeMinMultDegs for $d_{\max} = 31$. On the other hand, the proposed homomorphic comparison operation obtains M_{degs} for $d_{\max} = 63$ from ComputeMinTimeDegs. The depth consumption D should satisfy $D \geq M_{\text{dep}}$, where M_{dep} is the minimum

TABLE V
SET OF SCALING VALUES AND MARGINS FOR THE PREVIOUS HOMOMORPHIC COMPARISON ALGORITHM MinimaxComp AND THE PROPOSED ALGORITHM OptMinimaxComp.

α	depth	previous homomorphic comparison algorithm MinimaxComp		proposed homomorphic comparison algorithm OptMinimaxComp	
		scaling value w	margin η	scaling value w	margin η
8	11	{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	{1,2,2,2,1.6}	2^{-16}	{1,2,2,2,1.7}	2^{-16}
	18	{1,2,2,2,2,1.6}	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	{1,2,2,2,2,1.6}	2^{-20}	{1,2,2,2,2,1.8}	2^{-22}
	23	{1,2,2,2,2,2,1.6}	2^{-20}	{1,2,2,2,2,2,1.6}	2^{-20}
20	25	{1,2,2,2,1.6}	2^{-22}	{1,2,2,2,1.7}	2^{-24}
	26	{1,2,2,2,2,1.6}	2^{-23}	{1,2,2,2,2,1.7}	2^{-26}
	27	{1,2,2,2,2,2,2,1.6}	2^{-22}	{1,2,2,2,2,2,1.7}	2^{-29}
	28	{1,2,2,2,2,2,2,2,1.6}	2^{-22}	{1,2,2,2,2,2,2,1.6}	2^{-26}

TABLE VI
SET OF SCALING VALUES AND MARGINS FOR THE PREVIOUS HOMOMORPHIC MAX/RELU FUNCTION ALGORITHMS MinimaxMax/MinimaxReLU AND THE PROPOSED ALGORITHMS OptMinimaxMax/OptMinimaxReLU.

α	ζ	depth	previous homomorphic max/ReLU function algorithms MinimaxMax/MinimaxReLU		proposed homomorphic max/ReLU function algorithms OptMinimaxMax/OptMinimaxReLU	
			scaling value w	margin η	scaling value w	margin η
8	12	11	{1,2,1.6}	2^{-12}	{1,1.7}	2^{-12}
12	15	14	{1,2,2,1.6}	2^{-16}	{1,2,2,1.6}	2^{-16}
16	16	18	{1,2,2,2,1.6}	2^{-20}	{1,2,2,2,1.6}	2^{-20}
		19	{1,2,2,2,2,1.6}	2^{-20}	{1,2,2,2,1.6}	2^{-20}
20	21	22	{1,2,2,2,1.6}	2^{-24}	{1,2,2,2,1.8]	2^{-24}
		23	{1,2,2,2,2,1.6}	2^{-24}	{1,2,2,2,2,1.6}	2^{-24}
		24	{1,2,2,2,2,2,1.6}	2^{-24}	{1,2,2,2,2,2,1.6}	2^{-24}

TABLE VII
COMPARISON BETWEEN THE RUNNING TIME (AMORTIZED RUNNING TIME) OF THE PREVIOUS HOMOMORPHIC COMPARISON ALGORITHM MinimaxComp AND THAT OF THE PROPOSED ALGORITHM OptMinimaxComp IN THE RNS-CKKS SCHEME.

α	depth	previous homomorphic comparison algorithm MinimaxComp		proposed homomorphic comparison algorithm OptMinimaxComp	
		degrees	running time	degrees	running time
8	11	{7,15,15}	4.21 s (0.12 ms)	{3,15,31}	4.08 s (0.12 ms)
12	16	{15,15,15,15}	9.92 s (0.30 ms)	{7,15,15,31}	9.61 s (0.29 ms)
	17	{7,7,7,13,13}	9.91 s (0.30 ms)	{3,3,13,15,31}	9.40 s (0.28 ms)
	18	{3,5,7,7,7,13}	9.67 s (0.29 ms)	{5,5,5,15,29}	9.12 s (0.27 ms)
16	21	{15,15,15,15,27}	18.45 s (0.56 ms)	{7,13,15,15,59}	17.49 s (0.53 ms)
	22	{7,7,7,13,13,27}	18.23 s (0.55 ms)	{5,3,7,15,15,59}	16.79 s (0.51 ms)
	23	{5,7,7,7,7,13,13}	18.08 s (0.55 ms)	{5,3,5,7,7,15,31}	16.73 s (0.51 ms)
20	25	{29,31,31,31,31}	32.04 s (0.97 ms)	{15,15,31,59,63}	30.73 s (0.93 ms)
	26	{13,15,15,15,27,27}	29.59 s (0.90 ms)	{5,15,15,15,29,61}	28.27 s (0.86 ms)
	27	{7,7,7,7,7,7,15,27}	29.19 s (0.89 ms)	{5,5,7,15,15,15,59}	27.18 s (0.82 ms)
	28	{5,7,7,7,7,7,7,7,15}	29.13 s (0.88 ms)	{5,5,5,7,7,15,15,31}	26.77 s (0.81 ms)

TABLE VIII
COMPARISON BETWEEN THE RUNNING TIMES (AMORTIZED RUNNING TIMES) OF THE PREVIOUS HOMOMORPHIC MAX/RELU FUNCTION ALGORITHMS MinimaxMax/MinimaxReLU AND THOSE OF THE PROPOSED ALGORITHMS OptMinimaxMax/OptMinimaxReLU IN THE RNS-CKKS SCHEME.

α	ζ	depth	previous homomorphic max/ReLU function algorithms MinimaxMax/MinimaxReLU			proposed homomorphic max/ReLU function algorithms OptMinimaxMax/OptMinimaxReLU		
			degrees	running time		degrees	running time	
				MinimaxMax	MinimaxReLU		OptMinimaxMax	OptMinimaxReLU
8	12	11	{3,7,7}	2.26 s (0.069 ms)	2.27s (0.069 ms)	{5,23}	2.14 s (0.065 ms)	2.17 s (0.066 ms)
12	15	14	{5,7,7,15}	5.71 s (0.17 ms)	5.74 s (0.17 ms)	{3,5,7,29}	5.53 s (0.16 ms)	5.55 s (0.16 ms)
16	16	18	{7,7,7,15,15}	11.46 s (0.34 ms)	11.37 s (0.34 ms)	{3,3,15,15,31}	10.88 s (0.33 ms)	10.95 s (0.33 ms)
		19	{3,7,7,7,7,13}	11.76 s (0.35 ms)	11.79 s (0.36 ms)	{5,5,7,13,29}	10.65 s (0.32 ms)	10.71 s (0.32 ms)
20	21	22	{15,15,15,15,27}	21.72 s (0.66 ms)	20.38 s (0.62 ms)	{7,13,15,15,59}	19.26 s (0.58 ms)	19.40 s (0.59 ms)
		23	{7,7,7,13,13,27}	19.79 s (0.60 ms)	19.81 s (0.60 ms)	{5,7,7,13,15,31}	18.59 s (0.56 ms)	18.57 s (0.56 ms)
		24	{5,7,7,7,7,7,23}	19.77 s (0.60 ms)	19.81 s (0.60 ms)	{3,5,5,7,7,15,31}	18.38 s (0.56 ms)	18.37 s (0.56 ms)

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 VII. 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 VII does not include this case. Table VII also does not include cases when the previous and proposed algorithms use the same set of degrees M_{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_{degs} from ComputeMinTimeDegs for $d_{\max} = 63$. The used sets of degrees and running times (amortized running times) of the previous homomorphic max/ReLU function algorithms MinimaxMax/MinimaxReLU and the proposed algorithms OptMinimaxMax/OptMinimaxReLU are shown in Table VIII. 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 VIII 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_{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 a low failure rate ($< 2^{-15}$) and provided the parameter sets according to the precision parameter α. 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 α is 9 or 14 (resp. when α 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, "nGraph-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. Minihold, 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. Bossuat, 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.

[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:2106.07229*, 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] A. Kim, A. Papadimitriou, and Y. Polyakov, "Approximate homomorphic encryption with reduced approximation error," *Cryptol. ePrint Arch., Tech. Rep. 2020/1118*, vol. 2020, 2020.

[21] 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. 2020/552/20200803:084202*, 2020.

[22] E. W. Cheney, "*Introduction to approximation theory*," 1966.

[23] J.-W. Lee, E. Lee, Y. Lee, Y.-S. Kim, and J.-S. No, "High-precision bootstrapping of rns-ckks homomorphic encryption using optimal minimax polynomial approximation and inverse sine function," in *International Conference on the Theory and Applications of Cryptographic Techniques*, 2021.