Received January 15, 2022, accepted February 17, 2022, date of publication March 14, 2022, date of current version March 22, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3159694

Privacy-Preserving Machine Learning With Fully Homomorphic Encryption for Deep Neural Network

JOON-WOO LEE[1], HYUNGCHUL KANG[2], YONGWOO LEE[2], WOOSUK CHOI[2], JIEUN EOM[2], MAXIM DERYABIN[2], EUNSANG LEE[1], JUNGHYUN LEE[1], (Graduate Student Member, IEEE), DONGHOON YOO[2], YOUNG-SIK KIM[3], (Member, IEEE), AND JONG-SEON NO[1], (Fellow, IEEE)

[1]Department of Electrical and Computer Engineering, INMC, Seoul National University, Seoul 08826, Republic of Korea
[2]Samsung Advanced Institute of Technology, Suwon 16678, Republic of Korea
[3]Department of Information and Communication Engineering, Chosun University, Gwangju 61452, Republic of Korea

Corresponding author: Young-Sik Kim (iamyskim@chosun.ac.kr)

This work was supported by Samsung Advanced Institute of Technology.

ABSTRACT Fully homomorphic encryption (FHE) is a prospective tool for privacy-preserving machine learning (PPML). Several PPML models have been proposed based on various FHE schemes and approaches. Although FHE schemes are suitable as tools for implementing PPML models, previous PPML models based on FHE, such as CryptoNet, SEALion, and CryptoDL, are limited to simple and nonstandard types of machine learning models; they have not proven to be efficient and accurate with more practical and advanced datasets. Previous PPML schemes replaced non-arithmetic activation functions with simple arithmetic functions instead of adopting approximation methods and did not use bootstrapping, which enables continuous homomorphic evaluations. Thus, they could neither use standard activation functions nor employ large numbers of layers. In this work, we first implement the standard ResNet-20 model with the RNS-CKKS FHE with bootstrapping and verify the implemented model with the CIFAR-10 dataset and plaintext model parameters. Instead of replacing the non-arithmetic functions with simple arithmetic functions, we use state-of-the-art approximation methods to evaluate these non-arithmetic functions, such as ReLU and Softmax, with sufficient precision. Further, for the first time, we use the bootstrapping technique of the RNS-CKKS scheme in the proposed model, which enables us to evaluate an arbitrary deep learning model on encrypted data. We numerically verify that the proposed model with the CIFAR-10 dataset shows 98.43% identical results to the original ResNet-20 model with non-encrypted data. The classification accuracy of the proposed model is 92.43%±2.65%, which is quite close to that of the original ResNet-20 CNN model (91.89%). It takes approximately 3 h for inference on a dual Intel Xeon Platinum 8280 CPU (112 cores) with 172 GB of memory. We believe that this opens the possibility of applying FHE to an advanced deep PPML model.

INDEX TERMS Privacy-preserving machine learning, ResNet-20, RNS-CKKS FHE scheme, SEAL library, software implementation.

I. INTRODUCTION

The privacy-preserving issue is one of the most practical problems for machine learning recently. Fully homomorphic encryption (FHE) is the most appropriate tool for privacy-preserving machine learning (PPML) to ensure strong security in the cryptographic sense and satisfy the succinctness of communication. FHE is an encryption scheme in which ciphertexts can be processed with any deep Boolean or arithmetic circuits without access to the data. The security of FHE is usually defined as the indistinguishability under chosen-plaintext attack (IND-CPA) security, which is

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

a standard cryptographic security definition. If the client sends the public keys and encrypted data with an FHE scheme to the PPML server, the server can perform all the computations required in the desired service before sending the encrypted output to the client. Therefore, the application of FHEs to PPML has been extensively researched before now.

The most successful PPML model on homomorphically encrypted data prior to now was constructed using the Fast Fully Homomorphic Encryption over the Torus homomorphic encryption scheme (TFHE) by Lou and Jiang [1], but it used the leveled version of the TFHE scheme without bootstrapping rather than an FHE version. In other words, they chose in advance the parameters that can be used to perform the desired network without bootstrapping. If we want to design a deeper neural network with the leveled homomorphic encryption scheme, impractically large parameters must be used, which causes a heavy runtime or memory overhead. Furthermore, because the packing technique cannot be applied easily in the TFHE scheme, it can cause additional inefficiency with regard to the running time and memory overhead if we want to process many data simultaneously. Thus, it is desirable to use FHE with moderate parameters and bootstrapping, which naturally supports the packing technique in the PPML model.

Applicable FHE schemes with this property are word-wise FHE schemes, such as the Brakerski-Fan-Vercauteren (BFV) scheme [2] or Cheon-Kim-Kim-Song (CKKS) scheme [3], [4]. In particular, the CKKS scheme has gained considerable interest as a suitable tool for PPML implementation because it can deal with encrypted real numbers naturally. However, these schemes support only homomorphic arithmetic operations such as homomorphic addition and homomorphic multiplication. Unfortunately, popular activation functions are usually non-arithmetic functions, such as ReLU, sigmoid, leaky ReLU, and ELU. Thus, these activation functions cannot be directly evaluated using a word-wise FHE scheme. When previous machine learning models using FHE replaced the non-arithmetic activation function with simple polynomials, these models were not proven to show high accuracy for advanced classification tasks beyond the MNIST dataset.

Although many machine learning models require multiple deep layers for high accuracy, there is no choice but to use a small number of layers in previous FHE-based deep learning models until the fast and accurate bootstrapping techniques of FHE schemes have very recently become available. The bootstrapping technique transforms a ciphertext that cannot further support homomorphic multiplication into a fresh ciphertext by extending the levels of the ciphertext [5], [6]. However, the bootstrapping technique has been actively improved with regard to algorithmic time complexity [7]–[9], precision [10], and implementation [11], making bootstrapping more practical. The PPML model with many layers must be implemented using a precise and efficient bootstrapping technique in the FHE. In addition, because the training process is generally quite expensive as it requires many images and a large running time, it is more desirable to use the pre-trained parameters trained for the original standard plaintext machine learning model without any additional training process.

A. OUR CONTRIBUTION

For the first time, we implement the ResNet-20 model for the CIFAR-10 dataset [12] using the residue number system CKKS (RNS-CKKS) [4] FHE scheme, which is a variant of the CKKS scheme using the SEAL library 3.6.1 version [13], one of the most reliable libraries implementing the RNS-CKKS scheme. In addition, we implement bootstrapping of the RNS-CKKS scheme in the SEAL library according to [6]–[10] to support a large number of homomorphic operations for a deep neural network, as the SEAL library does not support the bootstrapping operation. ResNets are historic convolutional neural network (CNN) models that enable a very deep neural network with high accuracy for complex datasets such as CIFAR-10 and ImageNet. Many high-performance methods for image classification are based on ResNets because these models can achieve sufficiently high classification accuracy by stacking more layers. We first apply the ReLU function based on the composition of minimax approximate polynomials [14] to the encrypted data. Using the results, we show the possibility of applying FHE with bootstrapping to the standard deep machine learning model by implementing ResNet-20 over the RNS-CKKS scheme. The implemented bootstrapping can support a sufficiently high precision to successfully use bootstrapping in ResNet-20 with the RNS-CKKS scheme for the CIFAR-10 dataset.

Boemer *et al.* [15] pointed out that all existing PPML models based on FHE or multi-party computation (MPC) are vulnerable to model-extraction attacks. One of the reasons for this problem is that previous PPML methods with the FHE scheme do not evaluate Softmax with the FHE scheme. It simply sends the result before the Softmax function, and then it is assumed that the client computes Softmax by itself. Thus, information about the model can be extracted with many input-output pairs to the client. It is desirable for the server to evaluate the Softmax function with FHE. We first implement the Softmax function in the machine learning model using the method in [3], and this is the first implementation of a privacy-preserving machine learning model based on FHE mitigating the model extraction attack.

We prepare the pretrained model parameters by training the original ResNet-20 model with the CIFAR-10 plaintext dataset and perform privacy-preserving ResNet-20 with these plaintext pretrained model parameters and encrypted input images. We find that the inference result of the proposed privacy-preserving ResNet-20 is 98.43% identical to that of the original ResNet-20. It achieves 92.43%±2.65% classification accuracy, which is close to the original accuracy of 91.89%. Thus, we verify that the proposed implemented PPML model successfully performs ResNet-20 on encrypted data, even with the model parameters trained for the plaintext model.

B. RELATED WORKS

1) HE-FRIENDLY NETWORK

Some previous works re-designed the machine learning model to be compatible with the HE scheme by replacing the standard activation functions with simple nonlinear polynomials [16]–[20], called the HE-friendly network. Although the highest classification accuracy of the HE-friendly CNN with the simple polynomial activation function implemented by word-wise HE is 91.5% for the CIFAR-10 dataset [20], a better PPML machine learning model has not been demonstrated until now. This suggests that these machine learning models are usually successful only for a simple dataset and cannot achieve sufficiently high accuracy for an advanced dataset. Because the choice of activation functions is sensitive in the advanced machine learning model, it may not be desirable to replace the standard and famous activation functions with simple arithmetic functions. Moreover, an additional pre-training process must be conducted before the PPML service is provided. Because the training process is quite time-consuming and requires a large amount of data, it is preferable to use the standard model parameters of ResNets and VGGNets trained for plaintext data when the privacy of the testing dataset has to be preserved.

2) HYBRID MODEL WITH FHE AND MPC

Some previous studies evaluated non-arithmetic activation functions using the multiparty computation technique to implement the standard well-known machine learning model that preserves privacy [15], [21]–[24]. Although this method can accurately evaluate even non-arithmetic functions, the privacy of the model information can be disclosed. In other words, the client should know the activation function used in the model, which is undesirable for PPML servers. In addition, because communication with clients is not succinct, clients must be involved in the computation, which is not desirable for clients.

3) PPML WITH LEVELED HOMOMORPHIC ENCRYPTION

Some studies have used a leveled homomorphic encryption scheme to implement a standard machine learning model. A representative example is the work of Lou and Jiang [1], which implements ResNet-20 for the CIFAR-10 dataset or ResNet-18 for the ImageNet dataset with a leveled version of the TFHE scheme. When using a leveled homomorphic encryption scheme, we should set parameters capable of depth consumption for the desired circuit. Thus, to homomorphically evaluate deeper circuits, we must set large parameters. This property of the leveled homomorphic encryption scheme makes it difficult to evaluate a more deep learning model because the required parameters may be impractical to the general computing environment. Furthermore, the running time of each homomorphic encryption becomes larger, and thus, the total running time can be asymptotically larger than the linear time with the circuit depth. However, the FHE scheme uses practical parameters with a fixed size regardless of the circuit depth, and the total running time can be linearly proportional to the circuit depth. Therefore, for practical deep-learning models with large circuit depths, the implementation of a deep-learning model using the FHE scheme is an important research topic.

II. PRELIMINARIES

A. RNS-CKKS SCHEME

The CKKS scheme [3] is an FHE scheme that supports arithmetic operations on encrypted data over real or complex numbers. The structure of these encrypted data is a one-dimensional vector, where each component of this vector is called a slot. Users with a public key can process encrypted real or complex data using the CKKS scheme without knowing any private information. The security of the CKKS scheme is based on the ring-LWE hardness assumption. The supported homomorphic operations are the addition, scalar multiplication, non-scalar multiplication, rotation, and complex conjugation operations, and each operation except the homomorphic rotation operation is applied component-wise. While scalar multiplication is multiplication with plaintext, non-scalar multiplication is multiplication with ciphertext. The rotation operation homomorphically performs a cyclic shift of the vector in several steps. The non-scalar multiplication, rotation, and complex conjugation operations in the CKKS scheme require additional evaluation keys and key-switching procedures.

Each real amount of data is scaled with a large integer, called the scaling factor, and then rounded to the integer before encrypting the data. When the two data encrypted with the CKKS scheme are multiplied homomorphically, the scaling factors of the two data sets are also multiplied. This scaling factor should be reduced to the original value by using the rescaling operation.

Because the CKKS scheme requires somewhat large integers, the original CKKS scheme uses a multi-precision library, which requires a higher computational complexity. To reduce the complexity, a residue number system variant of the CKKS scheme [4], called the RNS-CKKS scheme, was also proposed. In the residue number system, a large integer is split into several small integers, and the addition and multiplication of the original large integers are equivalent to the corresponding component-wise operations of the small integers. The RNS-CKKS scheme was used in this study.

In this study, we denote homomorphic addition, homomorphic scalar multiplication, and homomorphic non-scalar multiplication as $\oplus, \odot, \otimes$. The homomorphic rotation operation for the left rotation with r steps is denoted as $\texttt{rot}(\mathsf{ct}, r)$.

B. BABY-STEP GIANT-STEP POLYNOMIAL EVALUATION

To utilize homomorphic encryption, many nonarithmetic operations of ResNets and bootstrapping must be approximated using high-order polynomials. When we evaluate a polynomial for the encrypted input with the RNS-CKKS scheme, it is important to reduce the number of non-scalar multiplications and depth consumption as much as possible. A well-known polynomial evaluation method that is efficient in nonscalar multiplications and depth consumption is the baby-step giant-step polynomial evaluation method.

Bossuat *et al.* [9] suggested a variant of the baby-step giant-step polynomial evaluation method that guarantees optimal depth consumption with a small additional non-scalar multiplication. Because depth consumption is generally more sensitive than the number of non-scalar multiplications because of the number of bootstrapping, we use this variant as a default method for homomorphic polynomial evaluation. Lee *et al.* [25] suggested an efficient method for polynomials with only odd-degree terms. These algorithms are elaborated upon with the proposed implementation based on a binary tree in Section III-A.

C. BOOTSTRAPPING OF CKKS SCHEME

The rescaling operation reduces both the scaling factor and ciphertext modulus, which are necessary for each homomorphic multiplication. After several consecutive multiplications, the ciphertext modulus cannot be further reduced. The bootstrapping operation of the CKKS scheme [6] transforms a ciphertext with a small modulus into a fresh ciphertext with a large modulus without changing the message. Therefore, any arithmetic circuit with a large multiplicative depth can be obtained using bootstrapping.

Bootstrapping of the CKKS scheme starts with an increase in the modulus of the ciphertext. Because the message polynomial becomes $m + q_0 I$, where q_0 is the modulus before bootstrapping and I is an unknown integer polynomial, the modular reduction of the coefficients of the message polynomial should be performed homomorphically to remove the $q_0 I$ part.

To move the coefficients of the message polynomials into the slots, CoeffToSlot is performed to the raised ciphertext. The core part of this operation is matrix multiplication with a Vandermonde matrix. Specifically, if we have U_0 and U_1 as:

$$U_0 = \begin{bmatrix} 1 & \zeta_0 & \cdots & \zeta_0^{N/2-1} \\ 1 & \zeta_1 & \cdots & \zeta_1^{N/2-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \zeta_{N/2-1} & \cdots & \zeta_{N/2-1}^{N/2-1} \end{bmatrix},$$

$$U_1 = \begin{bmatrix} \zeta_0^{N/2} & \zeta_0^{N/2+1} & \cdots & \zeta_0^{N-1} \\ \zeta_1^{N/2} & \zeta_1^{N/2+1} & \cdots & \zeta_1^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ \zeta_{N/2-1}^{N/2} & \zeta_{N/2-1}^{N/2+1} & \cdots & \zeta_{N/2-1}^{N-1} \end{bmatrix}$$

Then, the CoeffToSlot operation is the homomorphic evaluation of the two formulas, $\mathbf{z}_i' = \frac{1}{N}(\bar{U}_k^T \cdot \mathbf{z} + U_k^T \cdot \bar{\mathbf{z}})$ for $k = 0, 1$. Chen *et al.* [7] proposed an FFT-like optimization technique for this operation. They observed that these matrices are FFT-friendly in that some kind of butterfly structure can be applied to the operation. To provide a trade-off between the running time and depth consumption, they also proposed a level collapsing technique, where several layers in the butterfly structure were merged to reduce the depth consumption to the desired value. We used this FFT-like optimization technique and the level-collapsing technique in our implementation.

Next, we perform homomorphic modular reduction on the converted ciphertext, called ModReduction. There are several techniques, and we elaborate only on the techniques that we used to implement. Lee *et al.* [10] proposed that modular reduction is represented by the composition of several functions $h_3 \circ h_2^{\ell} \circ h_1$ such that

$$h_1(x) = \cos\left(\frac{2\pi}{2^{\ell}}\left(x - \frac{1}{4}\right)\right),\ h_2(x) = 2x^2 - 1,$$
$$h_3(x) = \frac{1}{2\pi}\arcsin x.$$

Then, h_1 and h_3 are approximated using minimax approximate polynomials in the approximation regions. Lee *et al.* also proposed an improved multi-interval Remez algorithm to obtain a minimax approximate polynomial for piecewise continuous functions. This approximation required several parameters. K is the parameter that determines the number of approximation intervals and ϵ is the half width of each interval. The approximation region was $\cup_{i=-(K-1)}^{K-1}[i - \epsilon, i + \epsilon]$. While the parameter ϵ is related to the range and precision of the input message data, the parameter K is related to bootstrapping failure. The additional parameters are the polynomial degrees of the approximate polynomials of the h_1 and h_3 functions. The homomorphic evaluation of polynomials is performed using the baby-step giant-step polynomial evaluation algorithm in Section II-B.

Then, the modular-reduced slots are reverted to the coefficients of the message polynomial using SlotToCoeff. The SlotToCoeff operation is a homomorphic evaluation of the formula $\mathbf{z} = U_0 \cdot z_0 + U_1 \cdot z_1$. This operation can also be optimized with FFT-like optimization and level-collapsing techniques [7].

D. MINIMAX COMPOSITION OF ReLU

Lee *et al.* [14] showed that the ReLU function had to be approximated with sufficiently high precision if we use pre-trained model parameters with the original ResNet-20 model. A polynomial with a large degree is required if a single minimax polynomial approximates the ReLU function, and a large running time is required to evaluate homomorphically. Instead of using a single minimax polynomial for the ReLU function, they used the formula $\mathrm{ReLU}(x) = \frac{1}{2}x(1 + \mathrm{sign}(x))$ and approximated $\mathrm{sign}(x)$ by the minimax composition of the small degree polynomials [26]. It reduced the running time of the homomorphic evaluation of the ReLU function, and this approximation method made the homomorphic evaluation of non-arithmetic functions, such as the ReLU function, more practical.

Lee *et al.* [26] specified a method for determining the optimal composite polynomials for the sign function. When each polynomial composing the composite polynomial was found, the range of the previous polynomial was used as the approximation domain for the next polynomial. If each polynomial is a minimax approximate polynomial of the sign function for each domain, the range of each polynomial is always two intervals symmetric to the origin. Each degree of the element polynomial requiring minimal nonscalar multiplications for the desired precision is determined by a dynamic programming algorithm.

III. NEW CONSIDERATION FOR ResNet-20 ON RNS-CKKS SCHEME

To implement the ResNet-20 model with the RNS-CKKS scheme, three new points must be considered: binary tree-based implementation for polynomial evaluation, natural implementation for the strided convolution, and implementation of the Softmax function.

A. BINARY TREE BASED IMPLEMENTATION OF POLYNOMIAL EVALUATION

For a more intuitive and systematic implementation, we modify the baby-step giant-step polynomial evaluation algorithm using a binary tree data structure. There is a precomputation process for recursively dividing by the division algorithm for the polynomial, which is shown in Algorithm 1. The output of `DividePoly` is a binary tree useful in the homomorphic polynomial evaluation process.

Algorithm 2 shows the binary tree-based baby-step giant-step polynomial evaluation algorithm. For optimal depth consumption, we may further divide the leftmost leaf node as by Bossuat *et al.* [9] in Lines 3–13. We generalize the giant step degree as an arbitrary integer rather than a power-of-two integer, as in [25].

Then, Lines 15–18 homomorphically evaluate the polynomial in the non-leaf nodes and leaf nodes. $T_n(x)$ denotes the nth Chebyshev polynomial. The Chebyshev polynomials have the following recursive formula:

$$T_{m+n}(x) - T_{m-n}(x) = 2T_m(x)T_n(x),$$

where $m \geq n$. When we homomorphically evaluate the Chebyshev polynomials in Lines 15 and 17, we use the formula $m = n$, where $T_0(x)$ is 1. When we homomorphically evaluate other Chebyshev polynomials in Line 16, we set m as the largest power-of-two integer less than the degree, and n as the difference between the degree and m.

Lines 19–26 reduce the binary tree until it has only the root node by homomorphically evaluating the polynomials for non-leaf nodes with two leaf nodes. This implementation is essentially the same as the method in [9]; however, it is easier to design the implementation for the algorithm.

Lee *et al.* [25] suggested a method for polynomials with only odd-degree terms. They observed that, if k is even, there is no need to evaluate the Chebyshev polynomials with an even degree that is not a power-of-two integer in Line 16. If we denote `OddPolyEval` rather than `PolyEval` in the following section, we omit these polynomial evaluation processes.

Algorithm 1: `DividePoly(p; k)`

```
Input  : A degree-$d$ polynomial $p$, a giant step
         parameter $k$
Output: A binary tree $P$ with leaf having polynomials
1  if $d < k$ then
2      return a binary tree $P$ with a single root node having
       $p$
3  else
4      Find $m$ such that $k \cdot 2^{m-1} < d \leq k \cdot 2^m$.
5      Generate a binary tree $P$ with a single root node
        having $T_{k \cdot 2^{m-1}}$.
6      Divide $p$ by $T_{k \cdot 2^{m-1}}$ to obtain the quotient $q$ and the
        remainder $r$.
7      Generate a binary tree $Q$ using DividePoly($q$; $k$).
8      Generate a binary tree $R$ using DividePoly($r$; $k$).
9      Append $Q$, $R$ to the left child and the right child of
        the root in $P$, respectively.
10     return $P$
11 end
```

B. STRIDED CONVOLUTION

Juvekar *et al.* [21] proposed an efficient convolution operation for a packing structure in an FHE scheme. They also proposed a strided convolution operation on the homomorphic encryption scheme by decomposing the strided convolution into a sum of nonstrided convolutions. However, their proposed strided convolution operation is not natural for the packing structure in the RNS-CKKS scheme. Furthermore, the following operations after their strided convolution are difficult to perform on the RNS-CKKS scheme.

We propose an efficient and natural method for strided convolution in the RNS-CKKS scheme. Instead of decomposing the strided convolution, we regard the output of the strided convolution as part of the non-strided convolution, as in fact the output data for the non-strided convolution includes the output data for the strided convolution. If we perform non-strided convolution, there are some gaps between the required output data for the strided convolution, which is not completely uniform in the regular sense. Thus, after performing the non-strided convolution, we perform homomorphic scalar multiplication with a window kernel that reflects these gaps. The slot structure of the output data of the strided convolution in the output slots of the nonstrided convolution is shown in Fig. 1.

We also find that this slot structure with regular gaps is compatible with the following ReLU functions, non-strided convolution operations, and even strided convolution operations. Because the ReLU function is evaluated componentwise, this slot structure does not consider the ReLU function. The non-strided convolution to the slot structure after the

```
Algorithm 2: PolyEval(ct, $p$; $k$)
  Input : A ciphertext $\mathsf{ct} = \mathsf{Enc}(x)$, a degree-$d$ polynomial $p$
  Output: A ciphertext of $p(x)$
1 Generate a binary tree $P$ using DividePoly($p$; $k$).
2 $l \leftarrow \lceil \log k \rceil - 1$
3 if $P$ is a full binary tree and the leftmost leaf polynomial has degree more than $2^l$ then
4     $p_0 \leftarrow$ the leftmost leaf polynomial
5     $V \leftarrow$ the leftmost leaf node
6     while the polynomial in $V$ has degree more than $2^l$ do
7         Replace $p_0$ with $T_{2^l}$ in $V$.
8         Divide $p_0$ by $T_{2^l}$ to obtain the quotient $q$ and the remainder $r$.
9         Append $q$, $r$ to the left child and the right child of $V$.
10        $V \leftarrow$ the left child of $V$
11        $l \leftarrow l - 1$
12    end
13 end
14 $l \leftarrow \lceil \log k \rceil - 1$
15 Homomorphically evaluate $T_2(x), T_4(x), \cdots, T_{2^l}(x)$ using $T_{2n}(x) = 2T_n(x)^2 - 1$.
16 Homomorphically evaluate other $T_n(x)$ for $3 \leq n \leq k$.
17 Homomorphically evaluate $T_{2k}(x), \cdots, T_{2^{m-1}\cdot k}(x)$.
18 Evaluate all of leaf node polynomials using the pre-computed ciphertexts.
19 while $P$ has only a root node do
20     $V \leftarrow$ one of the non-leaf nodes that have two leaf child
21     $\mathsf{ct}_T \leftarrow$ ciphertext for the polynomial $(T(x))$ in $V$ (pre-computed in Line 15, 17)
22     $\mathsf{ct}_q \leftarrow$ ciphertext for the polynomial $(q(x))$ in left child of $V$
23     $\mathsf{ct}_r \leftarrow$ ciphertext for the polynomial $(r(x))$ in right child of $V$
24     $\mathsf{ct}_T \leftarrow \mathsf{ct}_q \otimes \mathsf{ct}_T \oplus \mathsf{ct}_r$
25     Replace $T(x)$ with $q(x)T(x) + r(x)$ in node $V$ and remove the childs of $V$
26 end
27 return $\mathsf{ct}_p$ for input polynomial $p(x)$
```

proposed strided convolution can be performed with Gazelle's convolution method [21], with all rotation steps doubled. Additional strided convolution to the slot structure after the strided convolution can be performed with the non-strided convolution for this slot structure, followed by additional filtering. With these convolution methods, we can perform non-strided and strided convolution operations, even after several strided convolutions.

In ResNet-20, we only use convolution with a stride of one or two, and thus we assume that the strided convolution is convolution with stride two. Each convolution operation should be given an additional parameter slotstr, which represents the slot structure for meaningful data in the input ciphertext of each convolution. The parameter slotstr is stored in each ciphertext for each channel and initialized with zero, and it is added by one only when the strided convolution is performed. If the non-strided convolution is performed, we apply Gazelle's convolution method with the steps multiplied by 2^{slotstr}. If the strided convolution is performed, we perform the same procedure as the non-strided convolution, except for the following filtering. A specific algorithm for the strided convolution is presented in Section V-C.

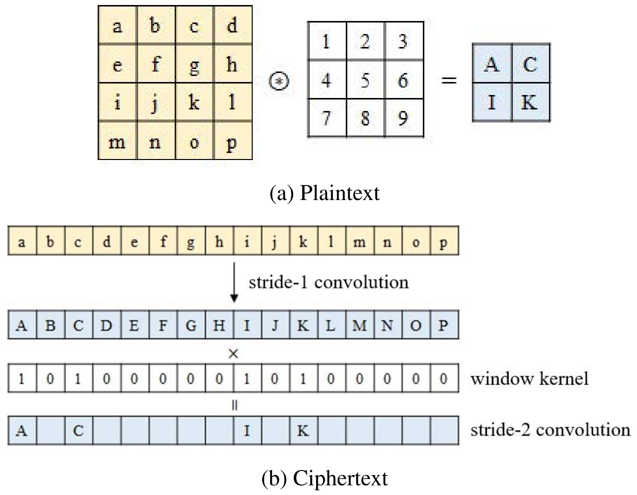


FIGURE 1. Stride-2 convolution.

C. APPROXIMATION FOR SOFTMAX

The inverse function of the Softmax function is unstable, that is, if one tries to recover the inputs to Softmax from the erroneous Softmax outputs, the recovered input may be quite different because of the amplification of noise in the output. Various inherent noises in homomorphic computations occur in the RNS-CKKS scheme, and thus, the input to Softmax will be difficult to recover. Thus, we implemented the Softmax function in our privacy-preserving ResNet-20 implementation for security against the model extraction attack. The Softmax function is $e^{x_i} / \sum_{j=0}^{T-1} e^{x_j}$ for each $i = 0, \cdots, T-1$, where T denotes the number of classification types. Because the Softmax function was not implemented in previous works for the PPML with homomorphic encryption, the approximation method for the Softmax function should be newly designed. There are two non-arithmetic operations in the Softmax function: an exponential function and an inverse function.

We use different approximation techniques for these non-arithmetic functions because of the differences in the characteristics of the input values. The absolute input values of the exponential function are dozens, but the output values of that function are unstable. However, the input values of the inverse function are unstable because each scale of the input value is different from the input value. Based on these characteristics, we chose the following approximation methods, and the entire algorithm is suggested in Section V-G.

1) EXPONENTIAL FUNCTION

If we simply approximate the exponential functions on a desired interval, the approximation may not be accurate, because the scales of the output for the exponential function can be too varied. Assume that we must approximate the exponential function e^x in $[-B, B]$. We can then regard the function as $(e^{x/B})^B$. Note that we can approximate e^y in $[-1, 1]$ when we set $y = x/B$, and the exponential function in this interval is easy to approximate. Thus, we approximate the exponential function in $[-1, 1]$ using the least-squares method, and we find that the approximate polynomial with degree 12 can approximate sufficiently precisely. Then, when we homomorphically evaluate the exponential function in $[-B, B]$, we divide the input by B, evaluate the approximate polynomial for the exponential function, and exponentiate it with B. If we set B as a power-of-two integer, the exponentiation with B can be implemented by repeated squaring.

2) INVERSE FUNCTION

Although the exponential function has a range with various scales, the inverse function in the Softmax function has a domain with various scales. This characteristic makes the approximation of the inverse function difficult with ordinary polynomial approximation, even with some scaling of the input. In this case, the Goldschmidt division method is appropriate for evaluating the inverse function of the input with various scales [27], [28]. In the Goldschmidt division method, the following formula is used,

$$\frac{1 - x^{2^n}}{1 - x} = \prod_{i=0}^{n-1}(1 + x^{2^i}).$$

If $|x| < 1$, where the left term of the above formula converges to $1/(1 - x)$ quickly, even with a small n. When we substitute $y = 1 - x$, the inverse function $1/y$ can be approximated as $\prod_{i=0}^{n-1}(1 + (1 - y)^{2^i})$, when $0 < y < 2$. Note that, even if y is close to zero, the approximated inverse function value is amplified to a very large number. This characteristic cannot be satisfied by using ordinary polynomial approximation methods. This characteristic can be used to reserve the role of the Softmax function in generating a one-hot vector, even when we approximate the Softmax function.

When the range of the input is $(0, 2R]$, we consider the inverse function in the range of $1/R \cdot 1/(y/R)$. In other words, the input value is multiplied by $1/R$, evaluated by the inverse function with the Goldschmidt division method, and multiplied by $1/R$ again. Note that R is a very large number, and the input may be far less than R. Even if y/R is very close to zero, the Goldschmidt method stably evaluates the inverse function, as previously mentioned.

3) GUMBEL SOFTMAX FUNCTION

If the input value of the Softmax function is large, the bound B for the range of the exponential function should be so large that the output value exceeds the capacity of the homomorphic encryption scheme. If the value of R is set to a fixed value for sufficient precision of the inverse function, the input value of the inverse function can be larger than $2R$. In this case, we can use the Gumbel Softmax technique, which evaluates the following function instead of the Softmax function:

$$\frac{e^{x_i/\lambda}}{\sum_{j=0}^{T-1} e^{x_j/\lambda}},$$

where λ is an additional parameter. If we use the Gumbel Softmax function, the output vector is still similar to the one-hot vector, and thus the model extraction attack can be sufficiently mitigated. Furthermore, the range of the exponential function is reduced from B to $B^{1/\lambda}$ and the input of the inverse function is included in $(0, 2R]$.

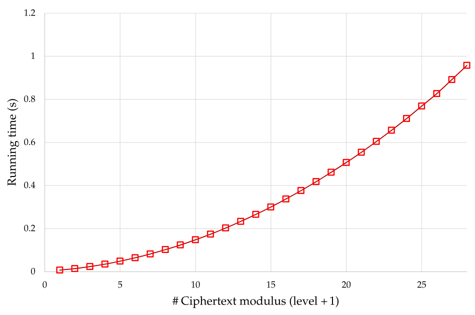
IV. POSITION OF BOOTSTRAPPING

Because we first use the bootstrapping operation in a machine learning model, we should consider performing the bootstrapping operation in the middle of the ResNet-20 model. In this section, we analyze several factors affecting the efficiency of the bootstrapping operation.

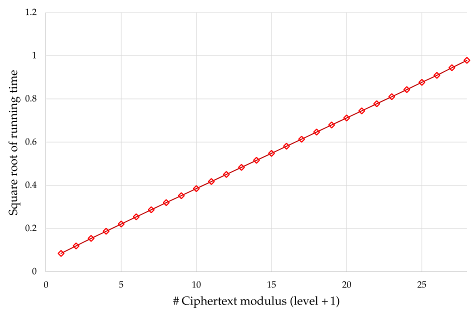
The key-switching operation is the heaviest operation in the homomorphic operations in the RNS-CKKS scheme; thus, the nonscalar multiplication, rotation, and complex conjugation requiring the key-switching operation are far heavier than the addition and scalar multiplication. Therefore, the number of key-switching operations roughly determines the total number of operations.

There is a major additional factor affecting the total number of operations and the level of ciphertext for the key-switching operation. The key-switching operation includes the decomposing, multi-sum, and mod-down operations. While the mod-down operation is linear with the level of the input ciphertext, the decomposing operation and multi-sum operation are quadratic with the level of the input ciphertext. Because the most time-consuming operation among the three procedures is the decomposing operation, the key-switching operation is a quadratic function with level. This quadratic property shows the large effect of the ciphertext level on the key-switching operation and total number of operations.

This quadratic property is numerically confirmed using the `SEAL` library, as shown in Fig. 2. Fig. 2 shows the running time of the rotation operation for the various levels of the input ciphertext, where the most part of the rotation operation is the key-switching operation, and it also shows the graph of the square root of the running time to represent the quadratic property more clearly. The square root of the running time is almost a linear function with level, which confirms the quadratic property. Thus, the sum of the squared level of each input ciphertext of each key-switching operation can be simulated much more closely than the number of key-switching operations.



(a) T_{rot}-$(\ell+1)$ graph



(b) $\sqrt{T_{\text{rot}}}$-$(\ell+1)$ graph

FIGURE 2. Running time for the rotation operation for various number of ciphertext modulus with $N = 2^{16}$ (a) T_{rot}-$(\ell+1)$ graph (b) $\sqrt{T_{\text{rot}}}$-$(\ell+1)$ graph.

Although the most time-consuming operation in ResNet-20 is the bootstrapping operation, the level of the ciphertexts for each key-switching operation in the bootstrapping is fixed, regardless of the structure of ResNet-20. Thus, it is desirable to compare the number of key-switching operations of the convolution and ReLU functions. We note that the number of key-switching operations in the convolution operation is significantly higher than that in the ReLU function because of the numerous rotation operations in the convolution operation.

This suggests that it is desirable to perform bootstrapping immediately after the convolution operation. Then, the convolution operation is performed at the lowest level of the ciphertext, and many rotation operations in the convolution operation are significantly reduced. A numerical comparison of this analysis is presented in Section VI.

V. IMPLEMENTATION DETAILS OF ResNet-20 ON RNS-CKKS

A. STRUCTURE

Fig. 3 shows the structure of the ResNet-20 model and Table 1 shows the specification of the ResNet-20. With this structure, We design our implemented structure for ResNet-20 using the RNS-CKKS scheme, as shown in Fig. 4, where it consists of convolution (Conv), batch normalization (BN), ReLU, bootstrapping (Boot), average pooling (AP), fully connected layer (FC), and Softmax. This model is virtually identical to the original ResNet-20 model, except that bootstrapping procedures are added. These procedures are described in the following subsections.

B. GENERAL SETTING FOR RNS-CKKS SCHEME

1) PARAMETERS

We set the ciphertext polynomial degree to 2^{16} and the secret key Hamming weight to 64. The bit lengths of the base modulus (q_0), special modulus, and default modulus are set to 60, 60, and 50, respectively. The bit length of the modulus in the bootstrapping range is the same as that of q_0. The numbers of levels for the general homomorphic operations and bootstrapping are set to 11 and 13, respectively. The maximum bit length of the modulus is 1450, which satisfies the 111.6-bit security. The security level λ is computed based on Cheon *et al.*'s hybrid dual attack [29], which is the fastest attack on the LWE with a sparse key. Table 2 lists the parameters.

2) DATA PACKING

The message is a $32 \times 32 \times 3$ CIFAR-10 RGB image, and one single image is processed at a time. We can use 2^{15} message slots in one ciphertext with our parameters, which is a half of polynomial degree. Rather than using the full slots of the ciphertext, we employ the sparse packing method [6] to pack a channel of a CIFAR-10 image in one ciphertext using only 2^{10} sparse slots. This is because the bootstrapping of sparsely packed ciphertext takes much less time than that of fully packed ciphertext, and convolution operations can be performed more smoothly with minimal rotation operations.

We construct a structure for the encrypted tensor. In our implementation, it is not sufficient to have ciphertexts composing the encrypted data, but we must store the slot structure parameter generated by the strided convolution. For ease of understanding, we also store the dimensions of the tensor in the encrypted tensor. An encrypted tensor $\mathsf{Tensorct}$ for a tensor in $\mathbb{R}^{\ell\times\ell\times h}$ is in the form of $(\{\mathsf{ct}_k\}_k, \ell, \mathsf{slotstr}, h)$, where $\{\mathsf{ct}_k\}_k$ is an array of ciphertexts comprising the encrypted tensor, and $\mathsf{slotstr}$ is the slot structure parameter. Algorithm 3 shows the detailed algorithm for encrypting image tensors.

Algorithm 3: $\texttt{EncTensor}(A \in \mathbb{R}^{L\times L\times H})$

```
Input  : A tensor $A \in \mathbb{R}^{L\times L\times H}$
Output: An encrypted tensor $\mathsf{Tensorct}$
1  for $k = 0$ to $H-1$ do
2      $v_k \leftarrow \mathbf{0} \in \mathbb{R}^{L^2}$
3      for $i = 0$ to $L-1$ do
4          for $j = 0$ to $L-1$ do
5              $x \leftarrow i \cdot L + j$
6              $v_k[x] \leftarrow A[i,j,k]$
7          end
8      end
9      $\mathsf{ct}_k \leftarrow \mathsf{Enc}(v_k; N, L^2)$
10 end
11 return $(\{\mathsf{ct}_k\}_{k=0,\cdots,H-1}, L, 0, H)$
```

TABLE 1. **The specification of the ResNet-20 (CIFAR-10).**

Layer		Input Size	#Inputs	Filter Size	#Filters	Output Size	#Outputs
Conv1		32×32	3	3×3	16	32×32	16
Conv2	2-1	32×32	16	3×3	16	32×32	16
	2-2	32×32	16	3×3	16	32×32	16
	2-3	32×32	16	3×3	16	32×32	16
Conv3	3-1-1	32×32	16	3×3	32	16×16	32
	3-1-2	16×16	32	3×3	32	16×16	32
	3-1-s	32×32	16	1×1	32	16×16	32
	3-2	16×16	32	3×3	32	16×16	32
	3-3	16×16	32	3×3	32	16×16	32
Conv4	4-1-1	16×16	32	3×3	64	8×8	64
	4-1-2	8×8	64	3×3	64	8×8	64
	4-1-s	16×16	32	1×1	64	8×8	64
	4-2	8×8	64	3×3	64	8×8	64
	4-3	8×8	64	3×3	64	8×8	64
Average Pooling		8×8	64	8×8	64	-	64
Fully Connected		64×1	1	-	-	-	10

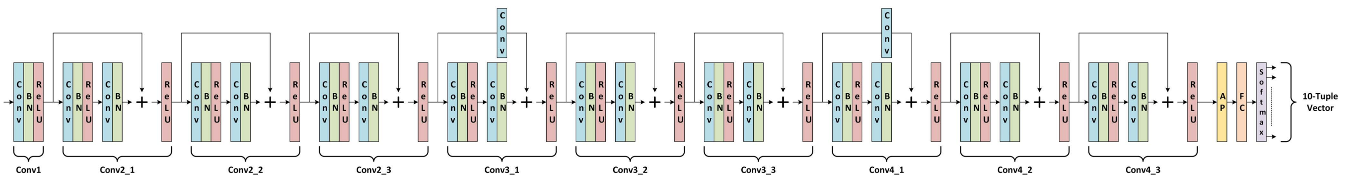


FIGURE 3. **Structure of ResNet-20.**

TABLE 2. **RNS-CKKS parameter settings.**

λ	Hamming Weight	Degree	Modulus Q	q_0	Special Prime	Scaling Factor	Evaluation Level	Bootstrapping Level
111.6	64	2^{16}	1450 bits	60 bits	60 bits	50 bits	11	13

3) DATA RANGE AND PRECISION

Any polynomial can approximate continuous functions only in certain bounded sets. If even one value in the message slots exceeds this bounded set, the absolute value of the output diverges to a large value, leading to complete classification failure. Because FHE can only handle arithmetic operations, polynomial approximation should be used for non-arithmetic operations, such as the ReLU function, bootstrapping, and Softmax functions. Therefore, the inputs for these procedures should be within the bounded approximation region. We analyze the absolute input values for ReLU, bootstrapping, and Softmax when performing ResNet-20 with several images. Because the observed maximum absolute input value for these procedures is 37.1, we conjecture that the absolute input values for these procedures are less than 40 with a very high probability. This observation is used in the implementation of each procedure. We also empirically find that the precision of the approximate polynomial or the function should be at least 16bits below the decimal point; thus, we approximate each non-arithmetic function with 16-bit average precision.

4) OPTIMIZATION FOR PRECISION OF HOMOMORPHIC OPERATIONS

We apply several methods to reduce the rescaling and relinearization errors and ensure the precision of the resultant message, such as scaling factor management in [30], lazy rescaling, and lazy relinearization [31], [32]. Lazy rescaling and relinearization can also be applied to reduce the computation time, as they require significant computation owing to the number-theoretic transformation (NTT) and gadget decomposition.

C. CONVOLUTION AND BATCH NORMALIZATION

Most of the operations in ResNet-20 are convolutions with zero-padded inputs to maintain their size. We use the packed single-input single-output (SISO) convolution with stride 1 used in Gazelle [21]. Strided convolution with stride 2 is also required to perform downsampling, and it is performed by the method proposed in Section III-B. Algorithm 4 shows the detailed algorithm for convolution, which includes both non-strided convolution and strided

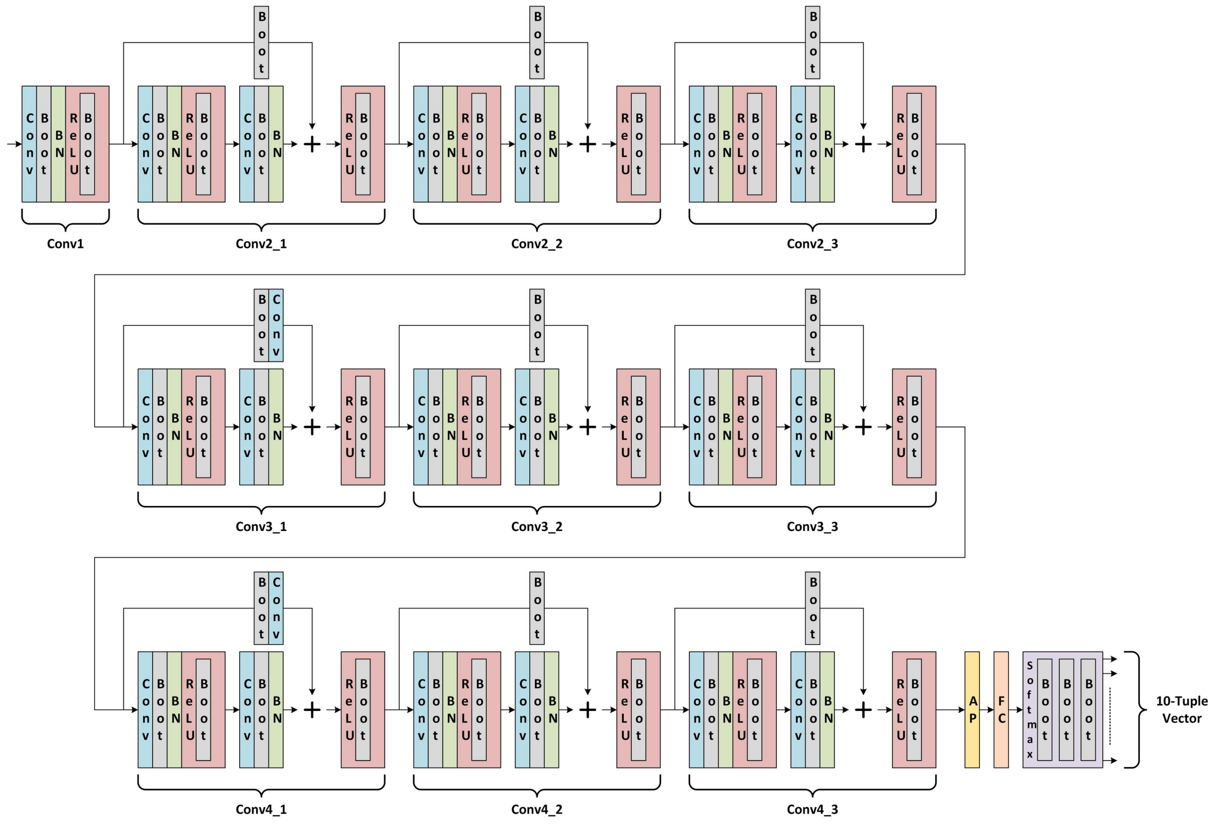


FIGURE 4. Proposed structure of ResNet-20 over RNS-CKKS scheme.

convolution. Non-strided convolution is performed when str is 1, and strided convolution is performed when str is 2. Each rotation step is multiplied by the value of slotstr, as discussed in Section III-B.

Because the batch normalization procedure is a simple linear function with constant coefficients, it can be implemented using homomorphic addition and homomorphic scalar multiplication.

D. ReLU

For the first time, we implement the ReLU function in ResNet-20 with the RNS-CKKS scheme using the composition of the minimax polynomial approximation proposed by Lee *et al.* [14]. To find an appropriate precision value, we repeatedly perform a ResNet-20 simulation over the RNS-CKKS scheme while changing the precision, and we find that the minimum 16-bit precision shows good performance on average.

To synthesize the sign function for the ReLU approximation, we generate the composition of the small minimax approximate polynomials with precision parameter $\alpha = 12$ using three minimax approximate polynomials with degrees 7, 15, and 27. Algorithm 5 generates composite polynomials approximating the sign function [26]. GenMinimax(f, d, D) in Algorithm 5 is an algorithm that generates the minimax approximate polynomial with degree d for function f on domain D, and we implement

Algorithm 4: Conv(Tensorct, W, stride)

```
Input  : An encrypted tensor
         Tensorct = ($\{ct_k\}_{k=0,\cdots,t-1}$, $\ell$, slotstr, $t$),
         weight parameters $W \in \mathbb{R}^{c\times c\times t\times t'}$ ($c$ is an odd
         integer), and the stride of the convolution
         operation str
Output: An output encrypted tensor Tensorct'
1  $L \leftarrow \ell \cdot$ slotstr
2  for $h = 0$ to $t' - 1$ do
3      $ct'_h \leftarrow \mathbf{0}$
4      for $k = 0$ to $t - 1$ do
5          for $(i, j) = (0, 0)$ to $(c - 1, c - 1)$ do
6              $w \leftarrow \mathbf{0} \in \mathbb{R}^{L^2}$
7              for $(i', j') = (0, 0)$ to $(\ell - 1, \ell - 1)$ do
8                  if $0 \le i' + i - \lfloor c/2 \rfloor \le \ell - 1$ and $0 \le$
                    $j' + j - \lfloor c/2 \rfloor \le \ell - 1$ then
9                      $w[(i' \cdot L + j') \cdot \text{slotstr} \cdot \text{str}] \leftarrow$
                         $W[i, j, k, h]$
10                 end
11             end
12             $r \leftarrow (i - \lfloor c/2 \rfloor) \cdot L + (j - \lfloor c/2 \rfloor)$
13             $ct'_h \leftarrow ct'_h \oplus (w \odot \texttt{rot}(ct_k, r \cdot \text{slotstr}))$
14         end
15     end
16 end
17 return ($\{ct'_h\}_{h=0,\cdots,t'-1}$, $\ell/\text{str}$, slotstr $\cdot$ str, $t'$)
```

Algorithm 5: `GenSignPoly`(α, $\{d_i\}_i$)

```
Input  : Precision parameter of sign function $\alpha$,
         sequence of composite polynomial degrees
         $\{d_i\}_{i=0,\cdots,s-1}$
Output: Sequence of composite polynomials for sign
         function $\{p_i\}_{i=0,\cdots,s-1}$ where
         $p_{s-1} \circ \cdots \circ p_0(x) \approx \text{sign}(x)$
1 for $i = 0$ to $s-1$ do
2     if $i = 0$ then
3         $D_0 \leftarrow [-1, -2^{-\alpha}] \cup [2^{-\alpha}, 1]$
4     else
5         $D_i \leftarrow \text{Range}(p_{i-1}, D_{i-1})$
6     end
7     $p_i \leftarrow \text{GenMinimax}(\text{sign}, d_i, D_i)$
8 end
```

Algorithm 6: `ReLU`(Tensorct, $\{p_i\}_i$)

```
Input  : An encrypted tensor
         Tensorct $= (\{\text{ct}_k\}_{k=0,\cdots,t-1}, \ell, \text{slotstr}, t)$,
         sequence of composite polynomials for sign
         function $\{p_i\}_{i=0,\cdots,s-1}$
Output: An activated encrypted tensor with ReLU
         Tensorct$'$
1 for $k = 0$ to $t-1$ do
2     $\text{ct}'_k \leftarrow \text{ct}_k$
3     for $i = 0$ to $s-1$ do
4         $\text{ct}'_k \leftarrow$ OddPolyEval($\text{ct}'_k, p_i$)
5     end
6     $\text{ct}'_k \leftarrow (0.5 \odot \text{ct}_k) \otimes (1 + \text{ct}'_k)$
7 end
```

this algorithm using the multi-interval Remez algorithm [10]. Range(f, D) denotes the range of f in the domain D.

Algorithm 6 shows the homomorphic evaluation method for the ReLU function using the composite polynomials generated by Algorithm 5 as the input. After homomorphically evaluating the p_i's in order, we homomorphically evaluate $x(1 + \text{sign}(x))/2$.

This composition of polynomials ensures that the average approximation precision is approximately 16-bit precision. The homomorphic evaluation of the polynomials is performed using the odd baby-giant method in [25] and the optimal level consumption method in [9]. Because the homomorphic evaluation of polynomial compositions consumes many depths, it is impossible to complete it without bootstrapping. Thus, we use bootstrapping twice in a layer, once in the middle, and once at the end of evaluating the ReLU function.

E. BOOTSTRAPPING

Because we have to consume many depths to implement ResNet-20 in the RNS-CKKS scheme, many bootstrapping procedures are required to ensure sufficient homomorphic multiplications. For the first time, we apply the bootstrapping technique to perform deep neural networks such as ResNet-20 on encrypted data and prove that the FHE scheme with state-of-the-art bootstrapping can be successfully applied to privacy-preserving deep neural networks. Because the `SEAL` library does not support any bootstrapping technique, we implement the most advanced bootstrapping with the `SEAL` library [9]–[11]. CoeffToSlot and SlotToCoeff are implemented using a collapsed FFT structure [7] with a depth of 2. The ModReduction is implemented using the composition of the cosine function, two double-angle formulas, and the inverse sine function [8], [10], where the cosine function and the inverse sine function are approximated using the multi-interval Remez algorithm as in [10].

The most crucial issue when using bootstrapping in the RNS-CKKS scheme is bootstrapping failure. More than a thousand bootstrapping procedures are required in our model, and the result of the entire neural network can be largely distorted if even one of the bootstrapping procedures fails. Bootstrapping failure occurs when one of the slots in the input ciphertext of the ModReduction procedure is not within the approximation region. The approximation interval can be controlled by bootstrapping parameters (K, ϵ), where the approximation region is $\cup_{i=-(K-1)}^{K-1}[i-\epsilon, i+\epsilon]$ [6]. While parameter ϵ is related to the range and precision of the input message data, parameter K is related to the values composing the ciphertext and is not related to the input data. Because the values contained in the ciphertext are not predictable, we must investigate the relationship between the bootstrapping failure probability and the parameter K.

We describe how bootstrapping failure affects the entire ResNet evaluation and propose a method to reduce the bootstrapping failure probability. As CKKS bootstrapping is based on the sparsity of the secret key, there is a failure probability of bootstrapping.

The decryption formula for a ciphertext (a, b) of the CKKS scheme is given as $a \cdot s + b = m + e \pmod{\mathcal{R}_q}$ for secret key s; hence, $a \cdot s + b \approx m + q \cdot I \pmod{\mathcal{R}_Q}$, where the Hamming weight of s is h. As the coefficients of a and b are in $[-\frac{q_0}{2}, \frac{q_0}{2})$, the coefficients of $a \cdot s + b$ have an absolute value less than $\frac{q_0(h+1)}{2}$. However, based on the ring-LWE assumption, the coefficients of $a \cdot s + b$ follow a scaled Irwin-Hall distribution and it is assumed that the coefficients of $I < K = O(\sqrt{h})$ [32]. Because the modular reduction function is approximated in the domain $\cup_{i=-(K-1)}^{K-1}[i-\epsilon, i+\epsilon]$, If a coefficient of I has a value greater than or equal to K, the modular reduction returns a useless value and thus fails. This is why the approximated modular reduction in the previous CKKS bootstrapping has a certain failure probability.

TABLE 3. Boundary of approximation region given key Hamming weight and failure probability of modular reduction.

$\Pr(\lvert I_i \rvert \geq K)$	$h = 64$	$h = 128$	$h = 192$
2^{-23} [9]	12	17	21
2^{-30}	14	20	24
2^{-40}	16	23	28

Even though $O(\sqrt{h})$ is a reasonable upper bound for a single bootstrapping, it is not sufficient when the number of slots is large and there are many bootstrappings. Let p be the probability of modular reduction failure, $\Pr(|I_i| \geq K)$. If there are n slots in the ciphertext, then there are $2n$ coefficients to perform modular reduction. Hence, the failure probability of single bootstrapping is $1-(1-p)^{2n} \approx 2n \cdot p$. Similarly, when N_b bootstrappings exist in the evaluation of the entire network, the failure probability of the entire network is $2N_b \cdot n \cdot p$. As there are many slots in our ciphertext and thousands of bootstrapping are performed, the failure probability is very high when using previous approximate polynomials.

In Table 3, we present several bounds for the input message and its failure probability. A larger bound means that a higher degree of the approximate polynomial is required; hence, more computations are required. Using the new bound for the approximation in Table 3, we can offer a trade-off between the evaluation time and failure probability of the entire network. Following [9], [32], the approximated modular reduction in the CKKS bootstrapping thus far has a failure probability $\approx 2^{-23}$, but it is not sufficiently small because we have to perform many bootstrapping procedures for ResNet-20. Thus, the bootstrapping failure probability is set to less than 2^{-40} in our implementation. The Hamming weight of the secret key is set to 64, and $(K, \epsilon) = (17, 2^{-10})$. The corresponding degree for the minimax polynomial for the cosine function is 45, and that of the inverse sine function is 1, which is obtained using the multi-interval Remez algorithm [10]. The number of double-angle formulas ℓ is set to two.

F. AVERAGE POOLING AND FULLY CONNECTED LAYER

The size of the tensor after all convolutional layers are performed is $8 \times 8 \times 64$. We perform average pooling on each channel to obtain a vector of length 64 and a fully connected layer to obtain a vector of length 10. The form of the output for average pooling is an array of ciphertexts with a length of 64, and each element of the ciphertext array has corresponding data in the first slot. Because all data are separated into other ciphertexts, no rotation operation is required when we perform the fully connected layer. Algorithm 7 shows the detailed procedures for average pooling and the fully connected layers.

```
Algorithm 7: AvgPoolFullCon(Tensorct, $W$)
Input  : An encrypted tensor
         Tensorct $= (\{ct_k\}_{k=0,\cdots,t-1}, \ell, slotstr, t)$,
         weight parameters for fully connected layer
         $W \in \mathbb{R}^{T\times t}$
Output: An array of ciphertexts $\{ct'_k\}_{k=0,\cdots,T-1}$
1  for $k = 0$ to $t-1$ do
2      $\bar{ct}_k \leftarrow ct_k$
3      for $i = 0$ to $\log \ell - 1$ do
4          tmpct $\leftarrow$ rot($\bar{ct}_k$, slotstr $\cdot 2^i$)
5          $\bar{ct}_k \leftarrow \bar{ct}_k \oplus$ tmpct
6      end
7      for $j = 0$ to $\log \ell - 1$ do
8          tmpct $\leftarrow$ rot($\bar{ct}_k$, slotstr $\cdot \ell \cdot 2^i$)
9          $\bar{ct}_k \leftarrow \bar{ct}_k \oplus$ tmpct
10     end
11 end
12 for $u = 0$ to $T-1$ do
13     $ct'_u \leftarrow \mathbf{0}$
14     for $k = 0$ to $t-1$ do
15         $ct'_u \leftarrow ct'_u \oplus (W[u,k] \odot \bar{ct}_k)$
16     end
17 end
18 return $\{ct'_k\}_{k=0,\cdots,T-1}$
```

G. SOFTMAX

We use the Softmax method proposed in Section III-C. The bound parameters B and R are set to 64 and 10^4, respectively, and the Gumbel Softmax parameter λ is set to 4. The approximation parameter in Goldschmidt's division algorithm is set to 16. Although a parameter B greater than 40 is sufficient, as discussed in Section III-C, we use 64 because a power-of-two B is better for implementation. T is the number of classification types, which is 10 for the CIFAR-10 dataset. Algorithm 8 shows the detailed procedure for the Softmax function.

Because the Softmax function consumes many depths, several bootstrapping operations are used in the middle of the process. Bootstrapping is performed for each ciphertext just before the beginning of the Softmax function, just before the inverse function, and after eight iterations of the Goldschmidt division process.

VI. SIMULATION RESULT

A. SIMULATION SETTING AND MODEL PARAMETERS

We implement the proposed model using the SEAL library [13] released by Microsoft, equipped with our implementation of RNS-CKKS bootstrapping. Our simulation environment is a dual Intel Xeon Platinum 8280 CPU (112 cores) with 512GB memory. We allocate one thread per channel in each layer using the OpenMP library to improve the execution speed of ResNet-20. The memory required for this simulation is 172GB.

The model parameters are prepared using the following training method: we use 32×32 RGB images, subtract the mean of the pixels in the training dataset, and adopt a data augmentation method, such as shifting and mirroring horizontally, for training. We adopt He initialization [33] as weight initialization with no dropout. We train the model with 32×32 mini-batches and a cross-entropy loss function. The learning rate start at 0.001 divided by 10 after 80 epochs and 100 after 120 epochs during training. The classification accuracy with the trained model parameters is 91.89% and is tested using 10,000 images.

B. PERFORMANCE

Table 4 shows the agreement ratio between the classification results of the implemented privacy-preserving ResNet-20

TABLE 4. Classification accuracy of the ResNet-20 for plaintext and ciphertext and agreement ratio.

Model	ResNet-20[1]	ResNet-20[2]	PPML ResNet-20	Agreement
Accuracy	91.89%	92.95% ± 2.56%	92.43% ± 2.65%	98.43%± 1.25%

[1] Classification accuracy verified with 10,000 images.
[2] Classification accuracy verified with 383 images which are used to test ResNet-20 on encrypted images.

TABLE 5. The running time of the ResNet-20 and the percentage of time spent in each component relative to total time.

Layer	Conv	BN	ReLU	Boot	AP + FC	Softmax	Total time (s)
Time ratio	17.44%	13.55%	34.61%	31.55%	0.04%	2.81%	10,602

```
Algorithm 8: Softmax(Tensorct, $B$, $R$, $\lambda$)
Input  : An array of ciphertext $\{\mathsf{ct}_k\}_{k=0,\cdots,T-1}$, bound
         parameter $B$, $R$ ($B$ is a power-of-two integar),
         power-of-two Gumbel parameter $\lambda$,
         Goldschmidt approximation parameter $d$
Output: An encrypted one-hot vector $\{\mathsf{ct}'_k\}_{k=0,\cdots,T-1}$
1  $p_{\exp} \leftarrow \mathtt{GenApproxPoly}(e^x, [-1, 1])$
2  $\tilde{\mathsf{ct}} \leftarrow \mathbf{0}$
3  for $k = 0$ to $T-1$ do
4      $\mathsf{ct}'_k \leftarrow 1/B \odot \mathsf{ct}_k$
5      $\mathsf{ct}'_k \leftarrow \mathtt{polyeval}(\mathsf{ct}'_k, p_{\exp})$
6      for $i = 0$ to $\log B - \log\lambda$ do
7          $\mathsf{ct}'_k \leftarrow \mathsf{ct}'_k \otimes \mathsf{ct}'_k$
8      end
9      $\tilde{\mathsf{ct}} \leftarrow \tilde{\mathsf{ct}} \oplus \mathsf{ct}'_k$
10 end
11 $\tilde{\mathsf{ct}} \leftarrow 2 \ominus 1/R \odot \tilde{\mathsf{ct}}$
12 $\mathsf{tmpct} \leftarrow \tilde{\mathsf{ct}} \ominus 1$
13 for $j = 0$ to $d-1$ do
14     $\mathsf{tmpct} \leftarrow \mathsf{tmpct} \otimes \mathsf{tmpct}$
15     $\tilde{\mathsf{ct}} \leftarrow \tilde{\mathsf{ct}} \otimes (1 \oplus \mathsf{tmpct})$
16 end
17 for $k = 0$ to $T-1$ do
18     $\mathsf{ct}'_k \leftarrow \mathsf{ct}_k \otimes \tilde{\mathsf{ct}}$
19 end
20 return $\{\mathsf{ct}'_k\}_{k=0,\cdots,T-1}$
```

and that of the original ResNet-20, which shows almost identical results. We test the inference on 383 encrypted images, and the 95% confidence interval is suggested for each result. The classification accuracy of the ResNet-20 for the encrypted data is 92.43%± 2.65%, while that of the original ResNet-20 for the corresponding plaintext image is 92.95%± 2.56%. We also measure the agreement ratio, which is the percentage of the case when the output of the classification in the proposed PPML model is the same as that in the original ResNet-20 model. Our agreement ratio is 98.43%± 1.25%, which is a sufficiently high. Thus, we verify that the ResNet-20 can be successfully carried out using the RNS-CKKS scheme with sufficient accuracy for classification and the proper bootstrapping operation.

TABLE 6. Comparison of the running time of ResNet-20 for two positions of the bootstrapping.

Bootstrapping position	After conv	After ReLU
Total Time (s)	10,602	14,694

Table 5 shows the running time for the whole ResNet-20 and the portion for each component in the model. Note that we include the running time of the bootstrapping operation in BN or ReLU when the bootstrapping operation is performed in the middle of each operation. In other words, the regular bootstrapping for each layer is counted for the running time of the bootstrapping. The proposed model takes about 3 h to infer one image, and the most time-consuming components are the convolution, ReLU, and bootstrapping.

Table 6 shows the running time of ResNet-20 when the bootstrapping operation is performed after convolution operation and after ReLU function, respectively. The running time of the case when the bootstrapping is performed after convolution operation is reduced by 27.8% compared to the case when the bootstrapping is performed after ReLU function. This supports the analysis of the bootstrapping position in Section IV.

C. DISCUSSION

1) RUNNING TIME

The running time for the proposed model, which is about 3 h, is somewhat large for practical use. This study first shows the possibility of applying FHE to standard deep learning models with high accuracy, but it has to be optimized and improved in various ways to reduce the running time. Therefore, the essential future work is achieving an advanced implementation of ResNets with the RNS-CKKS scheme with various accelerators using a GPU, FPGA, or ASIC. Because research on implementing the state-of-the-art FHE scheme is advancing rapidly, ResNet-20 over encrypted data will soon be more practical. In addition, we implement the PPML model for only one image, and the running time per image can be significantly improved by properly using the packing method of the RNS-CKKS scheme. We leave this optimization for many images for future work.

2) SECURITY LEVEL

The security level of the proposed model is 111.6-bit security, which is a marginal security level that can be considered secure. Because the standard security level in most applications is 128 bits, this security level can be regarded as insecure, and we may want to raise the security level. However, this 111.6-bit security is not a hard limit of our implementation; we can easily raise the security level by changing the parameters of the RNS-CKKS scheme. This creates a trade-off between security and running time, and thus, we can reach a higher security level at the cost of longer running time.

3) CLASSIFICATION ACCURACY

Even if ML models are trained with the same hyper-parameters, the ML models have different performances because weights are initialized to random values for each training session. Thus, the ML model performance, such as the accuracy, is the average value obtained over several training sessions. However, because we focus on implementing ResNet-20 for homomorphically encrypted data, we train this model only once. Nevertheless, we have shown that the encrypted ResNet-20 operation is possible with almost the same accuracy as in the original ResNet-20 paper [34]. Furthermore, because it is implemented in the FHE with bootstrapping, it is expected that the same result will be obtained for a deeper network than Resnet-20.

VII. CONCLUSION

For the first time, we applied the RNS-CKKS scheme, a state-of-the-art FHE scheme, to the standard deep neural network ResNet-20 to implement PPML. Because the more precise approximations of the ReLU function, bootstrapping, and Softmax functions have not been applied to the PPML models until now, we applied these techniques with various fine-tuned parameters. We then showed that the implemented ResNet-20 with the RNS-CKKS scheme achieved almost the same result as the original ResNet-20 as well as the highest classification accuracy among the PPML models with the word-wise FHE scheme introduced so far. This work first suggested that the word-wise FHE with the most advanced techniques can be applied to a state-of-the-art machine learning model without re-training it.

ACKNOWLEDGMENT

(Joon-Woo Lee and HyungChul Kang contributed equally to this work.)

REFERENCES

[1] Q. Lou and L. Jiang, "SHE: A fast and accurate deep neural network for encrypted data," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2019, pp. 1–9.
[2] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptol. ePrint Arch., Bellevue, WA, USA, Tech. Rep. 2012/144, 2020. [Online]. Available: https://eprint.iacr.org/2012/144
[3] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. ASIACRYPT*, 2017, pp. 409–437.
[4] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Proc. Int. Conf. Sel. Areas Cryptogr. (SAC)*, Calgary, AB, Canada, 2018, pp. 347–368.
[5] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Theory Comput.*, 2009, pp. 169–178.
[6] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for approximate homomorphic encryption," in *Proc. EUROCRYPT*, 2018, pp. 360–384.
[7] H. Chen, I. Chillotti, and Y. Song, "Improved bootstrapping for approximate homomorphic encryption," in *Proc. EUROCRYPT*, 2019, pp. 34–54.
[8] K. Han and D. Ki, "Better bootstrapping for approximate homomorphic encryption," in *Proc. Cryptogr. Track RSA Conf.*, 2020, pp. 364–390.
[9] J.-P. Bossuat, C. Mouchet, J. Troncoso-Pastoriza, and A.-P. Hubaux, "Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys," in *Proc. EUROCRYPT*. Manhattan, NY, USA: Springer, 2021, pp. 587–617.
[10] 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 *Proc. EUROCRYPT*. Manhattan, NY, USA: Springer, 2021, pp. 618–647.
[11] W. Jung, S. Kim, J. H. Ahn, J. H. Cheon, and Y. Lee, "Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with GPUs," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2021, no. 4, pp. 114–148, Aug. 2021.
[12] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," M.S. thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2009.
[13] Microsoft. (2021). *Microsoft SEAL*. [Online]. Available: https://github.com/microsoft/SEAL
[14] J. Lee, E. Lee, J.-W. Lee, Y. Kim, Y.-S. Kim, and J.-S. No, "Precise approximation of convolutional neural networks for homomorphically encrypted data," 2021, *arXiv:2105.10879*.
[15] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, and H. Yalame, "MP2ML: A mixed-protocol machine learning framework for private inference," in *Proc. 15th Int. Conf. Availability, Rel. Secur.*, Nov. 2020, pp. 1–10.
[16] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and A. J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2016, pp. 201–210.
[17] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster CryptoNets: Leveraging sparsity for real-world encrypted inference," 2018, *arXiv:1811.09953*.
[18] T. van Elsloo, G. Patrini, and H. Ivey-Law, "SEALion: A framework for neural network inference on encrypted data," 2019, *arXiv:1904.12840*.
[19] A. A. Badawi, C. Jin, J. Lin, C. F. Mun, S. J. Jie, B. H. M. Tan, X. Nan, K. M. M. Aung, and V. R. Chandrasekhar, "Towards the AlexNet moment for homomorphic encryption: HCNN, the first homomorphic CNN on encrypted data with GPUs," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 3, pp. 1330–1343, Jul. 2021.
[20] E. Hesamifard, H. Takabi, and M. Ghasemi, "CryptoDL: Deep neural networks over encrypted data," 2017, *arXiv:1711.05189*.
[21] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 1651–1669.
[22] B. Reagen, W.-S. Choi, Y. Ko, V. T. Lee, H.-H.-S. Lee, G.-Y. Wei, and D. Brooks, "Cheetah: Optimizing and accelerating homomorphic encryption for private inference," in *Proc. IEEE Int. Symp. High-Performance Comput. Archit. (HPCA)*, Feb. 2021, pp. 26–39.
[23] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "nGraph-HE: A graph compiler for deep learning on homomorphically encrypted data," in *Proc. 16th ACM Int. Conf. Comput. Frontiers*, New York, NY, USA, Apr. 2019, pp. 3–13.
[24] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, "nGraph-HE2: A high-throughput framework for neural network inference on encrypted data," in *Proc. 7th ACM Workshop Encrypted Comput. Appl. Homomorphic Cryptogr.*, 2019, pp. 45–56.
[25] J.-W. Lee, E. Lee, Y. Lee, Y.-S. Kim, and J.-S. No, "Optimal minimax polynomial approximation of modular reduction for bootstrapping of approximate homomorphic encryption," Cryptol. ePrint Arch., Tech. Rep. 2020/552, 2nd Version, 2020. [Online]. Available: https://eprint.iacr.org/2020/552/20200803:084202
[26] E. Lee, J.-W. Lee, Y.-S. Kim, and J.-S. No, "Minimax approximation of sign function by composite polynomial for homomorphic comparison," *IEEE Trans. Dependable Secure Comput.*, early access, Aug. 18, 2021, doi: 10.1109/TDSC.2021.3105111.

[27] R. E. Goldschmidt, "Applications of division by convergence," Ph.D. dissertation, Dept. Elect. Eng., Massachusetts Inst. Technol., Cambridge, MA, USA, 1964.

[28] J. H. Cheon, D. Kim, D. Kim, H. H. Lee, and K. Lee, "Numerical method for comparison on homomorphically encrypted numbers," in *Proc. ASIACRYPT*, 2019, pp. 415–445.

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

[30] A. Kim, A. Papadimitriou, and Y. Polyakov, "Approximate homomorphic encryption with reduced approximation error," in *Proc. Cryptogr. Track RSA Conf.*, 2022, pp. 120–144.

[31] M. Blatt, A. Gusev, Y. Polyakov, K. Rohloff, and V. Vaikuntanathan, "Optimized homomorphic encryption solution for secure genome-wide association studies," *BMC Med. Genomics*, vol. 13, no. S7, pp. 1–13, Jul. 2020.

[32] Y. Lee, J. Lee, Y.-S. Kim, H. Kang, and J.-S. No, "High-precision and low-complexity approximate homomorphic encryption by error variance minimization," Cryptol. ePrint Arch., Bellevue, WA, USA, Tech. Rep. 2020/1549, 2022. [Online]. Available: https://eprint.iacr.org/2020/1549

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1026–1034.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

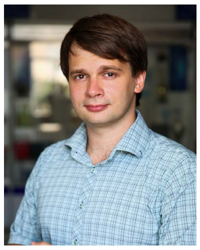
JOON-WOO LEE received the B.S. degree in electrical and computer engineering from Seoul National University, Seoul, South Korea, in 2016, where he is currently pursuing the Ph.D. degree. His current research interests include homomorphic encryption and lattice-based cryptography.

HYUNGCHUL KANG received the B.S. degree in industrial system and information on engineering and the Ph.D. degree from the Graduate School of Information Security, Korea University, Seoul, South Korea, in 2010 and 2018, respectively. He is currently a Staff Researcher at Samsung Advanced Institute of Technology, Suwon, South Korea. His research interests include symmetric cryptography, hash functions, homomorphic encryption, and privacy preserving machine learning.

YONGWOO LEE received the B.S. degree in electrical engineering and computer science from the Gwangju Institute of Science and Technology, Gwangju, South Korea, in 2015, and the M.S. and Ph.D. degrees in electrical and computer engineering from Seoul National University, South Korea, in 2017 and 2021, respectively. He is currently working as a Staff Researcher at Samsung Advanced Institute of Technology. He is also a submitter for a candidate algorithm (pqsigRM) in the first round for the NIST Post Quantum Cryptography Standardization. His current research interests include homomorphic encryption and code-based cryptography.

WOOSUK CHOI received the B.S. degree in electrical engineering from Inha University, Incheon, South Korea, in 2016, and the M.S. degree in electrical engineering from KAIST, Daejeon, South Korea, in 2019. His current research interests include homomorphic encryption and privacy preserving machine learning.

JIEUN EOM received the B.S. degree in mathematics and the M.S. and Ph.D. degrees in information security from Korea University, Seoul, in 2010, 2012, and 2019, respectively. She is currently a Staff Researcher with Samsung Advanced Institute of Technology. Her research interests include cryptography, information security, and homomorphic encryption.


MAXIM DERYABIN received the B.S. and M.S. degrees in mathematics and the Ph.D. degree in computer science from North Caucasus Federal University, Stavropol, Russia, in 2011, 2013, and 2016, respectively. He held various research and teaching positions at North Caucasus Federal University, from 2013 to 2020, including an Associate Professor in 2017. He is currently a Staff Researcher at Samsung Advanced Institute of Technology, Suwon, South Korea. One of the major topics of his research was residue number system and its applications. His current research interests include lattice-based cryptography, homomorphic encryption, computational algebra, and number theory.

EUNSANG LEE received the B.S. and Ph.D. degrees in electrical and computer engineering from Seoul National University, Seoul, South Korea, in 2014 and 2020, respectively. He is now a Postdoctoral Researcher at the Institute of New Media and Communications, Seoul National University. His current research interests include homomorphic encryption and lattice-based cryptography.

JUNGHYUN LEE (Graduate Student Member, IEEE) received the B.S. degree in statistics and the M.S. degree in mathematics from Seoul National University, Seoul, South Korea, in 2018 and 2021, respectively, where he is currently pursuing the Ph.D. degree in electrical and computer engineering. His current research interests include homomorphic encryption and lattice-based cryptography.

DONGHOON YOO received the M.S. and Ph.D. degrees in information and communication technology from the Gwangju Institute of Science and Technology, Gwangju, South Korea, in 1999 and 2005, respectively. He is currently working as a Research Master in privacy preserving computing and high-performance computing at Samsung Advanced Institute of Technology. His current research interests include homomorphic encryption, hardware acceleration, supercomputer architecture, and SW stack.

YOUNG-SIK KIM (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer science from Seoul National University, in 2001, 2003, and 2007, respectively. He joined the Semiconductor Division, Samsung Electronics, where he worked in the research and development of security hardware IPs for various embedded systems, including modular exponentiation hardware accelerator (called Tornado 2MX2) for RSA and elliptic-curve cryptography in smart-card products and mobile application processors, until 2010. He is currently a Professor with Chosun University, Gwangju, South Korea. He is also a submitter for two candidate algorithms (McNie and pqsigRM) in the first round for the NIST Post Quantum Cryptography Standardization. His research interests include post-quantum cryptography, the IoT security, physical-layer security, data hiding, channel coding, and signal design. He is selected as one of 2025's 100 Best Technology Leaders (for Crypto-Systems) by the National Academy of Engineering of Korea.

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 for 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 for 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.