Ph.D. DISSERTATION

Improved Fully Homomorphic Encryption and Practical Implementation of Privacy-Preserving Deep Neural Networks

완전동형암호의 개선 및 정보보호 심층신경망 모델의
실용적 구현

BY

LEE JOON-WOO

AUGUST 2022

DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Ph.D. DISSERTATION

Improved Fully Homomorphic Encryption and Practical Implementation of Privacy-Preserving Deep Neural Networks

완전동형암호의 개선 및 정보보호 심층신경망 모델의 실용적 구현

BY

LEE JOON-WOO

AUGUST 2022

DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Improved Fully Homomorphic Encryption and Practical Implementation of Privacy-Preserving Deep Neural Networks

완전동형암호의 개선 및 정보보호 심층신경망 모델의 실용적 구현

지도교수 노 종 선

이 논문을 공학박사 학위논문으로 제출함

2022년 8월

서울대학교 대학원

전기·정보 공학부

이 준 우

이준우의 공학박사 학위 논문을 인준함

2022년 8월

Abstract

Fully homomorphic encryption (FHE) is an encryption scheme that enables the server to process encrypted data sent by the clients without decrypting them in the client-server model. It allows the clients to outsource processing the sensitive data to the distrustful server without leaking any information about the data. Although several FHE schemes are constructed and improved in various aspects since the first construction of FHE in 2009, the existing FHE schemes are not sufficiently efficient for being applied to practical application. In this dissertation, four main crucial topics on FHE schemes are dealt with to make FHE schemes appropriate for practical applications: Bootstrapping of approximate homomorphic encryption, Galois key management, privacy-preserving deep neural network, and sorting for FHE'ed data.

First, the message precision in the bootstrapping operation of the residue number system variant Cheon-Kim-Kim-Song (RNS-CKKS) scheme is improved. Since the homomorphic modular reduction process is one of the most important steps in determining the precision of the bootstrapping, the homomorphic modular reduction process is focused on. I propose an *improved multi-interval Remez algorithm* obtaining the optimal minimax approximate polynomial of modular reduction function and the scaled sine/cosine function over the union of the approximation regions. Next, I propose the *composite function method using the inverse sine function* to reduce the difference between the scaling factor used in the bootstrapping and the default scaling factor. With these methods, the approximation error in the bootstrapping of the RNS-CKKS scheme is reduced by 1/1176$\sim$1/42 (5.4$\sim$10.2-bit precision improvement) for each parameter setting. While the bootstrapping without the composite function method has 27.2$\sim$30.3-bit precision at maximum, the bootstrapping with the composite function method has 32.6$\sim$40.5-bit precision.

Second, using the state-of-the-art techniques regarding FHE schemes, the ResNet-

20 network with the CKKS scheme is implemented with having almost the same classification accuracy as the plaintext counterpart without any modification of models or retraining. To further improve the performance, the total bootstrapping runtime is minimized using *multiplexed parallel convolution* that collects sparse output data for multiple channels compactly. The *imaginary-removing bootstrapping* is also proposed to prevent the deep neural networks from catastrophic divergence during approximate ReLU operations. In addition, level consumptions are optimized with lighter and tighter parameters. Simulation results show that the proposed implementation has 4.67$\times$ lower inference latency and 134$\times$ less amortized runtime (runtime per image) for ResNet-20 compared to the implementation without the multiplex convolution technique, with achieving standard 128-bit security. Furthermore, ResNet-32, 44, 56, 110 for CIFAR-10 dataset and ResNet-32 for CIFAR-100 dataset are successfully implemeneted, and the classification accuracy is similar to that of plaintext counterparts. The maximum classification accuracy is 92.9% at maximum, which is the highest accuracy among the similar privacy-preserving setting.

Third, a new concept of *hierarchical Galois key generation method* for homomorphic encryption is proposed to reduce the burdens of the clients and the server running BFV and CKKS schemes. The main concept in the proposed method is the hierarchical Galois keys, such that after the client generates and transmits a few Galois keys in the highest key level to the server, the server can generate any required Galois keys from the public key and the smaller set of Galois keys in the higher key level. This proposed method significantly reduces the number of the clients' operations for Galois key generation and the communication cost for the Galois key transmission. For example, if the standard ResNet-20 network for the CIFAR-10 dataset and the ResNet-18 network for the ImageNet dataset are implemented with pre-trained parameters of the CKKS scheme with the polynomial modulus degree $N = 2^{16}$ and $N = 2^{17}$, respectively, the server requires 265 and 617 Galois keys, which occupy 105.6GB and 197.6GB of memory, respectively. If the proposed three-level hierarchical Galois key system is used, the Galois key size generated and transmitted by the client can be reduced from 105.6GB to 3.4GB for ResNet-20 model for CIFAR-10, and reduced from 197.6GB to 3.9GB for ResNet-18 model for ImageNet.

Fourth, in order for the Shell sort algorithm to be used on the FHE data, the Shell sort is modified with an additional parameter α, allowing exponentially small sorting failure probability. For a gap sequence of powers of two, the modified Shell sort with input array length n is found to have the trade-off between the running time complexity of $O(n^{3/2}\sqrt{\alpha+\log\log n})$ and the sorting failure probability of $2^{-\alpha}$. Its running time complexity is close to the intended running time complexity of $O(n^{3/2})$ and the sorting failure probability can be made very low with slightly increased running time. Further, the near-optimal window length of the modified Shell sort is also derived via convex optimization. The proposed analysis of the modified Shell sort is numerically confirmed by using randomly generated arrays. For the practical aspect, the modification can be applied to any gap sequence, and it is shown that Ciura's gap sequence, which is known to have good practical performance, is also practically effective when the modified Shell sort is applied. The modified Shell sort is compared with other sorting algorithms with the FHE over the torus (TFHE) library, and it is shown that this modified Shell sort has the best performance in running time among in-place sorting algorithms on homomorphic encryption scheme.

keywords: Bootstrapping, Brakerski/Fan-Vercauteran scheme (BFV), Cheon-Kim-Kim-Song scheme (CKKS), Composite function approximation, Fully homomorhpic encryption (FHE), Fully homomorphic encryption with torus (TFHE), Hierarchical Galois key generation, Imaginary-removing bootstrapping, Improved multi-interval Remez algorithm, Modified Shell sort, Multiplexed convolution, Privacy-preserving machine learning, ResNet models, RNS-variant CKKS scheme (RNS-CKKS)

student number: 2016-24750

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

Fully homomorphic encryption (FHE) is the encryption scheme enabling any logical operations [11, 25, 33, 36, 71] or arithmetic operations [19, 21] with encrypted data. The FHE scheme makes it possible to preserve security in data processing. However, in the traditional encryption schemes, they are not encrypted to enable the processing of encrypted data, which causes clients to be dissuaded from receiving services and prevents companies from developing various related systems because of the lack of clients' privacy. FHE solves this problem clearly so that clients can receive many services by ensuring their privacy.

Since Gentry first designed fully homomorphic encryption in 2009, various optimizations have been made for fully homomorphic encryption. The hard computational problems on which fully homomorphic encryption is based have also varied from Approximate GCD Problem, LWE, Ring-LWE, etc., and the supported types of messages range from only one bit to finite ring in large modulus, real numbers, and complex numbers. In terms of time performance, the fully homomorphic encryption scheme, which was only theoretically possible but practically difficult to use, has improved rapidly and has come close to a practical level, leading to practical application and application to artificial intelligence systems. Recently, as research on fully homomorphic encryption is conducted to accelerate fully homomorphic encryption with accelerators such as GPU and FPGA, the implementation of fully homomorphic encryption is improved in terms of engineering sense.

However, optimization is still essential in various ways to use fully homomorphic encryption for practical use. Since the bootstrapping operation is essential to perform deep computational circuits with fully homomorphic encryption, optimization of the bootstrapping operation is also essential. Since the number of consecutive multiplication operations in most fully homomorphic encryption is limited to a finite number, unlimited use of the multiplication operations requires the conversion of ciphertexts that can no longer be multiplied into ciphertexts that can be multiplied. This type of operation is called bootstrapping. Since the bootstrapping operation is known as the most time-consuming among homomorphic operations, attempts to reduce bootstrapping time have continued in early studies of bootstrapping. Reducing the running time of bootstrapping is important, but increasing the precision of bootstrapping operations also plays an important role in improving bootstrapping performance in that the precision of bootstrapping is also lower than other homomorphic operations. If a lot of low-accuracy bootstrapping is used for deep operations, the amplification of the error may become severe and cause a catastrophic difference from the actual result value.

Recently, fully homomorphic encryption schemes have been actively used in the field of privacy-preserving machine learning area. The CKKS homomorphic encryption is known to be suitable for machine learning systems that mainly deal with real number data. Until now, however, doubting about the practicality of bootstrapping in CKKS homomorphic encryption scheme, almost all studies only deal with shallow-depth neural network models or activation functions with low-degree polynomials to avoid the use of the bootstrapping operations. For this reason, it was not proven whether a deep standard machine learning model verified by many studies in artificial intelligence academia could also be used with homomorphic encryption scheme. Therefore, it remains as an open problem whether validated deep standard machine learning models such as ResNet or VGGNet can be performed with the fully homomorphic encryption schemes.

Another important obstacle to fully homomorphic encryption is heavy Galois keys. The BFV and CKKS schemes support rotation operation which corresponds to a cyclic shift of message data within ciphertext. Many applications that require important operations such as bootstrapping, matrix multiplication, and convolution in convolutional neural networks can be achieved using this rotation. The Galois keys are evaluation keys for the homomorphic rotation operation, which is the cyclic shift operations for rows of the encrypted matrix in one ciphertext of the BFV scheme and for encrypted message vector in that of the CKKS scheme. The homomorphic rotation operation is inevitable if it is required to operate data with different positions in one ciphertext, such as the bootstrapping [8, 16, 20, 56, 58], the matrix multiplication [47], and the convolution in convolutional neural networks [49]. Since different Galois keys are required for all the different cyclic shift values for the homomorphic rotation operation, the number and the total size of Galois keys can be significantly large for the complex computational model. For example, if the standard ResNet-20 network for the CIFAR-10 dataset is implemeneted with pre-trained parameters with the CKKS scheme with the polynomial modulus degree $N = 2^{16}$, the server requires 265 Galois keys, which occupies 105.6GB of memory in the server. If the ResNet-18 network for the ImageNet dataset is designed using the same techniques, 617 Galois keys are required and it occupies 197.6GB of memory in the server.

Since the server processing encrypted data using fully homomorphic encryption does not know information on encrypted data, the time complexity of an algorithm cannot be reduced by changing each actions of the algorithm according to the data. Therefore, it is usually difficult to optimize the homomorphic encryption algorithm by the existing algorithm optimization method. A representative example is a sorting algorithm. For example, Quick sort is known as one of the fastest algorithm among sorting algorithms. This has the worst-case time complexity of $O(n^2)$, but in most cases, the data is sorted fast as the behavior of the algorithm is appropriately determined by the state of the data. However, in situations where data are encrypted with fully homomorphic encryption, the state of the data cannot be checked. Thus, each operation of the algorithm cannot be changed according to the encrypted data, and the optimization method of the quick sort cannot be applied. Due to this problem, it is necessary to appropriately modify the algorithm previously applied to the plaintext so that it can be effectively applied on the fully homomorphic encryption.

1.1 Contributions of Dissertation

High Precision Bootstrapping for RNS-CKKS FHE Scheme I propose two methods to improve the bootstrapping operation of the RNS-CKKS scheme. Firstly, I devise a fast algorithm, called an improved multi-interval Remez algorithm, obtaining the optimal minimax approximate polynomial of any continuous functions over any union of the finite number of intervals, which include the modular reduction function and the scaled sine/cosine function over the union of the approximation regions. Although the previous works have suggested methods to obtain polynomials that approximate the scaled sine/cosine function well from the minimax perspective, which are used to approximate the modular reduction function, these methods cannot obtain the optimal minimax approximate polynomial.

The original multi-interval Remez algorithm is not theoretically proven to obtain the minimax approximate polynomial, and it is only practically used for two or three approximation regions in the finite impulse response filter design, while it is required to approximate functions over the union of tens of intervals. Furthermore, it takes impractically much time if this algorithm is used without further improvement to obtain a polynomial that can be used for the bootstrapping. To make the multi-interval Remez algorithm practical, the multi-interval Remez algorithm is modified as the improved multi-interval Remez algorithm. Then the correctness of the improved multi-interval Remez algorithm is proved, including the original multi-interval Remez algorithm, for the union of any finite number of intervals. Since it can obtain the optimal minimax approximate polynomial in seconds, one can even adaptively obtain the polynomial when it is required to abruptly change some parameters on processing the ciphertexts so that it is required to update the approximate polynomial. All polynomial approximation methods proposed in previous works for bootstrapping in the CKKS scheme can be replaced with the improved multi-interval Remez algorithm, which ensures the best quality of the approximation. It ensures to use the least degree of the approximate polynomial for a given amount of error.

Next, the composite function method is proposed to enlarge the approximation region in the homomorphic modular reduction process using the inverse sine function. The crucial point in the bootstrapping precision is that the difference between the modular reduction function and the sine/cosine function gives a significant precision loss. All previous works have used methods that approximate the modular reduction function as a part of the sine/cosine functions. This approximation has an inherent approximation error so that the limitation of the precision occurs. Besides, to ensure that these two functions are significantly close to each other, the approximation region has to be reduced significantly. They set the half-width of one interval in the approximation region as 2^{-10}, which is equal to the ratio of default scaling factor to the scaling factor used in the bootstrapping. The message has to be scaled by multiplying 2^{-10} to make the message into the approximation region, and it is scaled by multiplying 2^{10} at the end of the bootstrapping. Thus, the precision error in the computation for bootstrapping is amplified by 2^{10}, and the 10-bit precision loss occurs. If it is tried to reduce this precision loss by enlarging the approximation region, the approximation error by the sine/cosine function becomes large, and thus the overall precision becomes lower than before.

Therefore, It is proposed to compose the optimal approximate polynomial of the inverse sine function to the sine/cosine function, since composing the inverse sine function to the sine/cosine function extends the approximation region of the modular reduction function, which makes it possible to improve the precision of the bootstrapping. Note that the inverse sine function used in this situation has only one interval in the approximation region, and thus the small approximate error can be reached with relatively low degree polynomials. the minimax approximate polynomials for the scaled cosine function and the inverse sine function are obtained with sufficiently small minimax error by the improved multi-interval Remez algorithm. these polynomials are applied in the homomorphic modular reduction process by homomorphically evaluating the approximate polynomial for the scaled cosine function, several double-angle formulas, and the approximate polynomial for the inverse sine function. This enables me to minimize the inevitable precision loss by approximating the modular reduction function to the sine/cosine function.

Since the previous works do not focus on the maximum precision of the bootstrapping of the RNS-CKKS scheme, the maximum precision of the bootstrapping with the previous techniques is checked. The detailed relation with the precision of the bootstrapping and various parameters is analyzed with `SEAL` library. With the proposed methods, the approximation error in the bootstrapping of the RNS-CKKS scheme is reduced by 1/1176$\sim$1/42 (5.4$\sim$10.2-bit precision improvement) for each parameter setting. While the bootstrapping without the composite function method has 27.2$\sim$30.3-bit precision at maximum, the bootstrapping with the proposed composite function method has 32.6$\sim$40.5-bit precision, which is better precision than 32-bit fixed-point precision.

Privacy-Preserving Deep Neural Network For the first time, the ResNet-20 model for the CIFAR-10 dataset [53] is implemented using the residue number system CKKS (RNS-CKKS) [21] FHE scheme, which is a variant of the CKKS scheme using the SEAL library 3.6.1 version [68], one of the most reliable libraries implementing the RNS-CKKS scheme. In addition, the bootstrapping of the RNS-CKKS scheme is implemented in the SEAL library according to [8,16,20,44,56] 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. the ReLU function is firstly applied based on the composition of minimax approximate polynomials [57] to the encrypted data. Using the results, the possibility of applying FHE with bootstrapping to the standard deep machine learning model is shown 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.

Even though the implementation by simply combining the existing techniques is an important step toward very-deep standard convolutional neural networks (VDSCNN)s, deeper CNNs than ResNet-20 have not yet been implemented. In addition, the implementation has high latency of 10,602s even with 64 CPU threads. One major reason for this high latency comes from its inefficient data packing. The practical PPML for VDSCNNs is implemented for the first time by resolving the efficiency and feasibility problems. My contributions are summarized as follows:

- the bootstrapping runtime is effectively reduced by using a multiplexed packing method (i.e., packing data of multiple channels into one ciphertext in a compact manner);
- A *multiplexed convolution* algorithm is proposed to perform convolutions for multiplexed input tensors, which also supports strided convolutions. A faster *multiplexed parallel convolution* algorithm is also proposed, which reduces the number of required rotations in the multiplexed convolution algorithm by 62% by utilizing full slots of ciphertext;

- It is found that a catastrophic divergence phenomenon occurs when implementing VDSCNNs using approximate ReLU (APR). The *imaginary-removing bootstrapping* is proposed to prevent this phenomenon so as to maintain the accuracy of PPML for VDSCNNs;
- Level consumptions are optimized with lighter and tighter parameters to achieve faster inference and the standard 128-bit security level;
- The ResNet-20 on the RNS-CKKS scheme is implemented using the SEAL library [68] with a latency of 3,972s with only one CPU thread, which is $4.67\times$ lower than that without multiplexed packing technique using 64 threads. Also, amortized runtime (runtime per image) of the proposed implementation is $134\times$ smaller due to a significant reduction of the number of operations;
- The ResNet-32/44/56/110 on the RNS-CKKS scheme are implemented with high accuracies close to those of backbone CNNs.

Hierarchical Galois Key Generation In the BFV and CKKS schemes, it is observed that the Galois keys can be generated from other Galois keys using key-switching operation. The crucial observation is that the Galois key can be regareded as a set of ciphertexts. If the key-switching operation is performed to each ciphertext in a Galois key, new Galois key for other cyclic shift can be derived. Since the key-switching operation requires a key-switching key with larger modulus (i.e., in the higher key level) than the ciphertext, the key-switching key for this Galois key generation should have higher level than the newly generated Galois key. This high-level key-switching key is also in the form of the Galois key, and thus it can also be generated by another higher-level key-switching key. Thus, a chain of Galois keys for various levels can be defined, where each Galois key may be used as a key-switching key for generating a lower-level Galois key.

From the above observations, a hierarchical Galois key generation system is proposed, which makes it possible to generate a lower-level Galois key using higher-level Galois key in the server. In this client-server model, clients can generate only a small set of the highest-level Galois keys such as Galois keys for only power-of-2 cyclic shifts. Then they send the small set of Galois keys to the key management server (KMS) or the server. The server can generate a large set of lower-level Galois keys using the received set of Galois keys without any help from the clients and finally, a set of level-zero Galois keys is generated, which corresponds to the set of conventional Galois keys for the cyclic shifts of message data within a ciphertext in the server. In the server, inert Galois keys can be temporarily removed and re-generated only when needed to efficiently manage the storage of Galois keys. This proposed method can significantly reduce the computational burdens of the client, the communication cost between the client and server, and storage cost of all Galois keys in the server. To further optimize this Galois key generation, several optimization techniques are proposed, such as the hoisted Galois key generation and the reduction to graph-theoretic algorithms.

I present a general protocol capable of efficient Galois key management reflecting the activity of the clients using multi-level Galois key generation scheme. When a client frequently uses the service, it is important to generate the desired Galois keys quickly so that the service should not be delayed due to the Galois key generation. On the other hand, in the case of clients who do not use the service frequently, it may be better to store only the minimal Galois key set and reserve memory in the server for other services to active users. However, it is required to prepare the inert client to become an active client at any time. In Figure 5.1 of three-level Galois key generation scheme, the client generates and transmits the minimum number of the level-2 Galois keys, and the server generates and retains an appropriate number of the level-1 Galois keys from the level-2 Galois keys reflecting how often the client uses services, where server and KMS can be collocated. With these level-1 Galois keys, the server can generate the level-0 Galois keys more efficiently. The role of the level-1 Galois keys is to give a trade-off between the efficiency of generating the level-0 Galois keys when requested and the memory used for storing Galois keys, and these level-1 Galois keys can be updated only by the server without any help from the client. The proposed protocol can enable this fine key management system to adjust in detail the trade-off between the memory usage of the Galois keys and the computational complexity of Galois key generation in the clients and the server.

I conduct the simulation with the proposed Galois key generation system for ResNet models with an appropriate computing environment for the client-server model. If a three-level hierarchical Galois key system is used, the Galois key size generated and transmitted by the client can be reduced from 105.6GB for 265 Galois keys to 3.4GB for 8 Galois keys for the ResNet-20 for CIFAR-10, and reduced from 197.6GB for 617 Galois keys to 3.9GB for 8 Galois keys for the ResNet-18 for ImageNet. While the generation of Galois keys for the ResNet-20 and the ResNet-18 by the client takes 368.5s and 786.0s in the conventional system, it is reduced to 12.1s (30$\times$) and 15.7s (50$\times$) in the three-level hierarchical Galois key system, respectively. The server with GPU accelerator only needs 25.3s and 22.0s to generate all required Galois keys in the online phase.

Modified Shell Sort for FHE The Shell sort [69, 70], which is one of the oldest sorting algorithms, is the generalized version of the insertion sort. The Shell sort algorithm is an in-place algorithm, which is fast and easy to implement, and thus, many systems use it as a sorting algorithm. It is known that Shell sort uses insertion sort as a subroutine algorithm, and insertion sort can be performed on the FHE data [14, 15]. However, the Shell sort should be modified to be used in the FHE setting. If any error in sorting is not allowed, then insertion sort is expected to be quite conservative, i.e., the number of operations for sorting must be set for the worst case, because the insertion sort algorithm in the FHE setting is an oblivious algorithm. Thus, if the insertion sort is used in the Shell sort, the running time complexity of Shell sort in the FHE setting must be $O(n^2)$, which makes the use of Shell sort ineffective. Therefore, it is important to devise a sorting algorithm that is better than the Shell sort on the FHE data in terms of running time complexity.

I devise a method to modify the Shell sort in the FHE setting using the window technique, which is proved to be effective in the theoretical aspect and the practical aspect. It is referred to as a "modified Shell sort". The window technique in [15] is applied to each subroutine insertion sort in the modified Shell sort for FHE setting. Note that the role of the window technique in the modified algorithm is different from the original use of the window technique. The algorithm does not reduce the bootstrapping itself compared to the number of the homomorphic gates, but the number of the comparison operations is reduced with the window technique. For this reason, the homomorphic comparison operation in the proposed sorting algorithm does not generate a comparison error.

For theoretical view, the running time complexity of the modified Shell sort is $O(n^{3/2}\sqrt{\alpha + \log\log n})$ with sorting failure probability (SFP) $2^{-\alpha}$ when the gap sequence is powers-of-two, which is close to the average-case time complexity $O(n^{3/2})$ of the original Shell sort. The value of α is an additional parameter that controls the trade-off between the running time and the SFP. This trade-off is quite effective because the SFP is decreased exponentially with α but the running time is proportional only to $\sqrt{\alpha}$. To this end, the exact distribution of window lengths of subarrays in each gap for successful sorting in the Shell sort is used. If the length of the subarray for the insertion sort in some gap is s, it is discovered that the average of the required window length for successful sorting is proportional to $\sqrt{s}$, and the right tail of its probability distribution is very thin. In the sorting process, the window length is provided as a constant multiple of $\sqrt{s}$, which ensures a negligible SFP. If the window length is close to $\beta\sqrt{s}$, the SFP decays as $e^{-\beta^2}$, which signifies a very fast-decaying function. Therefore, with a fixed negligible SFP, a small window length can be set so that the running time is asymptotically faster than that of the naive version of the Shell sort on the FHE data.

For the practical view, the running time of the modified Shell sort is effectively reduced even in the small arrays, compared to the basic in-place sorting algorithms on the FHE data, bubble sort, and insertion sort.

Chapter 2

PRELIMINARIES

2.1 Fully Homomorphic Encryption

The fully homomorphic encryption is a public-key encryption scheme, which supports an arbitrary number of additions and multiplications of plaintext without decryption so that anyone without the decryption key can operate the circuit with any ciphertext without leaking the information of its plaintext.

Gentry suggested the bootstrapping technique to transform a somewhat homomorphic encryption scheme, which allows only a finite number of operations on the encrypted data, to a fully homomorphic encryption scheme [35]. The bootstrapping operation has enabled several researchers to construct the FHE schemes [20, 35], which involves implementing the decryption circuit on encrypted data using the evaluation algorithm, that is, the addition and multiplication algorithms in the FHE setting. All of the FHE schemes suggested thus far ensure security by adding the plaintext to an LWE sample or a ring-LWE sample, which is known as pseudorandom samples. For security reasons, the LWE sample or the ring-LWE sample includes some errors. As the addition and multiplication operations are repeated, the total number of errors increases, and if the total number of errors exceeds a certain limit, a decryption failure occurs. Thus, the errors need to be removed after a certain number of operations on the encrypted data, so that the ciphertexts can be further evaluated. The purpose of the bootstrapping operation is to reset the errors in the ciphertext when the errors are too large to be decrypted.

As bootstrapping utilizes a considerable amount of computation during the processing of the FHE, the number of bootstrapping operations significantly affects the total number of operations of the FHE. In fact, the number of bootstrapping operations depends on the multiplicative depth of the circuit. The lower the depth of a circuit, the fewer the number of bootstrapping operations. Thus, it is crucial to consider the number of the bootstrapping operations for each element, when bootstrapping is implemented in the FHE schemes. If the total number of operations in an algorithm is fixed, it is better to evenly distribute the operations on the inputs. Furthermore, to stably address errors, deterministic algorithms are better than randomized algorithms. This is because the error size of each element can be predicted in deterministic algorithms ensuring that these errors are handled easily and error control is optimized adequately.

2.1.1 TFHE Scheme

The TFHE homomorphic encryption scheme [25] is the most practical bit-wise homomorphic encryption scheme now. There are two types of the TFHE scheme: the leveled homomorphic encryption and the fully homomorphic encryption. Since the fully homomorphic encryption version of the TFHE scheme is used, this version is only dealt with in this subsection. Its basic elements are the bootstrapped homomorphic gates, which perform each gate followed by the bootstrapping. Although the noise in the ciphertext grows when the homomorphic gate is performed without the bootstrapping, the bootstrapping refreshes the noise independent of the input noise. Hence, any large-depth boolean circuits can be performed without noise growth of the ciphertext using the TFHE scheme.

The secret key $\mathbf{s}$ is a vector of length n in $\{0,1\}^n$ uniformly sampled, and the ciphertext is formed by $(\mathbf{a}, b) \in \mathbb{T}^n \times \mathbb{T}$, where $b = \mathbf{a} \cdot \mathbf{s} + e + \mu$ and $\mu \in \{-\frac{1}{8}, \frac{1}{8}\}$ is encoded by $\mu = \frac{1}{4}(b - \frac{1}{2})$ with the message bit $b \in \{0, 1\}$. The bootstrapping procedure makes the encoded message $\frac{1}{8}$ when the input encoded message is in $(0, \frac{1}{2})$ and makes the encoded message $-\frac{1}{8}$ when the input encoded message is in $(\frac{1}{2}, 1)$. Before the bootstrapping for each bootstrapped homomorphic gate, each matched linear operation is processed so that the encoded message is in $(0, \frac{1}{2})$ when the output bit is 1 and is in $(\frac{1}{2}, 1)$ when the output bit is 0. The linear operations can easily be performed homomorphically since the LWE ciphertext has the linear property. For example, the homomorphic NAND gate performs $\frac{1}{8} - a - b$ homomorphically where a and b are the encoded message of the two input ciphertexts before the bootstrapping. All boolean gates can be designed by this method, and thus any boolean circuits can be composed with these bootstrapped homomorphic gates. Each linear operation for each homomorphic gate and the detailed bootstrapping procedure can be referred to [25].

2.1.2 BFV and CKKS Schemes

Fully homomorphic encryption, abbreviated as homomorphic encryption, is an encryption system designed to enable arbitrary arithmetic operations on encrypted data. Homomorphic encryption was initially defined as a bit-wise encryption scheme capable of performing all boolean circuits while encrypted. The definition was mitigated to include word-wise encryption schemes capable of arbitrary arithmetic circuits for encrypted integer or complex number data. The homomorphic encryptions covered in this dissertation are BFV and CKKS schemes. These homomorphic encryption schemes support arithmetic operations with the single-instruction multiple-data (SIMD) manner, allowing multiple independent data to be encrypted and operated at once in a single ciphertext with a single homomorphic operation. In the case of BFV, the data storing structure is a matrix in which the number of rows is two, and in the case of CKKS, it is a one-dimensional vector. In addition, in the case of BFV, integer data is encrypted, and in the case of CKKS, complex data is encrypted. Although each scheme has several variants, the schemes will be addressed with the following encoding and encryption methods.

- BFV scheme: The packing structure is a $2 \times N/2$-matrix $(v_{ij}) \in \mathbb{Z}_t^{2 \times N/2}$. Let ω be a $2N$-th root of unity in $\mathbb{Z}_t$. Then, $m(X) \in R_t$ is obtained such that $m(\omega^{\alpha_{ij}}) = v_{ij}$ for $\alpha_{ij} = (-1)^i \cdot 5^j \mod 2N$ and encrypt as $u \cdot (b, a) + (Q/t \cdot m + e_0, e_1)$, where $u, e_0, e_1 \leftarrow \chi$.
- CKKS scheme: The packing structure is a vector of length $N/2$, $(v_i) \in \mathbb{R}^{N/2}$. Let ζ be a $2N$-th root of unity in $\mathbb{C}$. Then, $m(X) \in \mathbb{R}[X]/(X^N + 1)$ is obtained such that $m(\zeta^{\alpha_i}) = v_i$ for $\alpha_i = 5^j \mod 2N$ and encrypt it as $u \cdot (b, a) + (\lfloor \Delta \cdot m \rceil + e_0, e_1)$, where $u, e_0, e_1 \leftarrow \chi$.

I assume that the residue number system variants of BFV and CKKS schemes [3, 21] are used. In these variants, the ciphertext modulus is chosen as the product of large primes, and the ciphertext is represented as a vector of remainders for the primes rather than one large remainder for the ciphertext modulus. By the Chinese remainder theorem (CRT), each vector of remainders for the primes has a one-to-one correspondence to the large remainder of the large modulus. The element-wise addition and multiplication between two vectors of remainders also correspond to those between two corresponding remainders of the product of the primes. The non-trivial operations in these RNS variants are the ModUp and $\mathsf{ModDown}$ operations. The ModUp operation raises the modulus with remaining the remainder, and the $\mathsf{ModDown}$ operation divides the modulus and the remainder by the product of some prime moduli and round the output. These operations include many heavy NTT/INTT operations and CRT merge processes, and thus these are one of the most time-consuming low-level operations in the BFV and CKKS. Since the decomposition process in the key-switching operation requires several ModUp processes, reducing the decomposition process is important in minimizing homomorphic operations. The specific ModUp and $\mathsf{ModDwon}$ operations are described in Algorithms 1 and 2, where it is assumed that each ring element is in the NTT form.

Algorithm 1 ModUp

Input: Two disjoint sets of primes $\mathcal{C} = \{q_0, \cdots, q_{\sigma-1}\}$, $\mathcal{B} = \{p_0, \cdots, p_{\tau-1}\}$, where $Q = \prod_i q_i$ and $P = \prod_j p_j$, and an RNS-form ring element $(a_0, \cdots, a_{\sigma-1}) \in \prod_{i=0}^{\sigma-1} R_{q_i}$ for $a \in R_Q$, where $a_i = a \mod q_i$.

Output: An RNS-form ring element $(\bar{a}_0, \cdots, \bar{a}_{\sigma-1}, \tilde{a}_0, \cdots, \tilde{a}_{\tau-1}) \in \prod_{i=0}^{\sigma-1} R_{q_i} \times \prod_{j=0}^{\tau-1} R_{p_i}$ for $a' \in R_{PQ}$, where $\bar{a}_i = a' \mod q_i$, $\tilde{a}_j = a' \mod p_j$, and $a' = a + Q \cdot e$ for small e.

```
INTT operation to $(a_0, \cdots, a_{\sigma-1})$.
for $i \leftarrow 0$ to $\sigma - 1$ do
    $\bar{a}_i \leftarrow a_i$
    $b_i \leftarrow a_i \cdot [\hat{q}_i^{-1}]_{q_i} \in R_{q_i}$
for $j \leftarrow 0$ to $\tau - 1$ do
    $\tilde{a}_j \leftarrow 0$
    for $i \leftarrow 0$ to $\sigma - 1$ do
        $\tilde{a}_j \leftarrow \tilde{a}_j + b_i \cdot [\hat{q}_i]_{p_j} \in R_{p_j}$
NTT operation to $(\bar{a}_0, \cdots, \bar{a}_{\sigma-1}, \tilde{a}_0, \cdots, \tilde{a}_{\tau-1})$.
return $(\bar{a}_0, \cdots, \bar{a}_{\sigma-1}, \tilde{a}_0, \cdots, \tilde{a}_{\tau-1}) \in \prod_{i=0}^{\sigma-1} R_{q_i} \times \prod_{j=0}^{\tau-1} R_{p_i}$
```

Algorithm 2 ModDown

Input: Two disjoint sets of primes $\mathcal{C} = \{q_0, \cdots, q_{\sigma-1}\}$, $\mathcal{B} = \{p_0, \cdots, p_{\tau-1}\}$, where $Q = \prod_i q_i$ and $P = \prod_j p_j$, and an RNS-form ring element $(\bar{a}_0, \cdots, \bar{a}_{\sigma-1}, \tilde{a}_0, \cdots, \tilde{a}_{\tau-1}) \in \prod_{i=0}^{\sigma-1} R_{q_i} \times \prod_{j=0}^{\tau-1} R_{p_i}$ for $a \in R_{PQ}$, where $\bar{a}_i = a \mod q_i, \tilde{a}_j = a \mod p_j$.

Output: An RNS-form ring element $(a'_0, \cdots, a'_{\sigma-1}) \in \prod_{i=0}^{\sigma-1} R_{q_i}$ for $a' \in R_Q$, where $a'_i = a' \mod q_i$ and $a' = \lfloor P^{-1} \cdot a \rceil + e$ for small e.

```
for $i \leftarrow 0$ to $\sigma - 1$ do
    $\bar{b}_i \leftarrow \bar{a}_i + [\lfloor P/2 \rfloor]_{q_i} \in R_{q_i}$
for $j \leftarrow 0$ to $\tau - 1$ do
    $\tilde{b}_j \leftarrow \tilde{a}_j + [\lfloor P/2 \rfloor]_{p_j} \in R_{p_j}$
$(\bar{b}'_0, \cdots, \bar{b}'_{\sigma-1}, \tilde{b}'_0, \cdots, \tilde{b}'_{\tau-1}) \leftarrow$ ModUp for $(\tilde{b}_0, \cdots, \tilde{b}_{\tau-1})$ from $\prod_{j=0}^{\tau-1} R_{p_i}$ to $\prod_{i=0}^{\sigma-1} R_{q_i} \times \prod_{j=0}^{\tau-1} R_{p_i}$
for $i \leftarrow 0$ to $\sigma - 1$ do
    $a'_i \leftarrow [P^{-1}]_{q_i} \cdot (\bar{b}_i - \bar{b}'_i) \in R_{q_i}$
return $(a'_0, \cdots, a'_{\sigma-1}) \in \prod_{i=0}^{\sigma-1} R_{q_i}$
```

This pair of ring elements (b, a) is the public key of each scheme. Although there are evaluation keys for homomorphic operations that are opened to the public domain, these pairs (b, a) for the encryption process are represented as the public key in this dissertation. For the CKKS scheme, the level of a ciphertext is the maximum number of multiplications that can be performed on the ciphertext without bootstrapping. In the RNS-CKKS scheme, if the level of a ciphertext is ℓ, there is $\ell + 1$ number of RNS moduli for the ciphertext. Each size of RNS moduli is determined by the required precision of the multiplication in each level. Since the other homomorphic operations rather than the rotation operation of the BFV and CKKS schemes are not relevant to understanding this dissertation, the rotation operation for $m(X)$ is only dealt with, corresponding to cyclic shift of the message vector. The detailed explanations of other operations can be found in [12, 19, 21, 33]. The rotation operation in the BFV scheme is an operation mapping $(v_{i,j}) \mapsto (v_{i,(j+r)})$ while encrypted, where the addition operation of the subscript is in modulo $N/2$. The rotation operation of the CKKS scheme is an operation mapping $(v_i) \mapsto (v_{i+r})$ while encrypted. In terms of ring elements, these operations can be unified as operations mapping $m(X) \mapsto m(X^{5^r})$.

For these operations, an operation of $(b(X), a(X)) \mapsto (b(X^{5^r}), a(X^{5^r}))$ operation is first performed. This processed ciphertext satisfies $b(X^{5^r}) + a(X^{5^r}) \cdot s(X^{5^r}) \approx m(X^{5^r})$, which means that it is a ciphertext with a plaintext of $m(X^{5^r})$ with the secret key $s(X^{5^r})$. this ciphertext should be converted to a ciphertext with the same plaintext and the original secret key. This is done by taking the key-switching operation from $s(X^{5^r})$ to $s(X)$.

2.1.3 RNS-CKKS Scheme

It is known that the CKKS scheme supports several operations for encrypted data of real numbers or complex numbers. Since it deals with usually real numbers, the noise that ensures the security of the CKKS scheme can be embraced in the outside of the significant figures of the data, which is the crucial concept of the CKKS scheme.

The RNS-CKKS scheme [21] uses the RNS form to represent the ciphertexts and to perform the homomorphic operations efficiently. While the power-of-two modulus is used in the CKKS scheme, the product of large primes is used for ciphertext modulus in the RNS-CKKS scheme so that the RNS system can be applied. These large primes are chosen to be similar to the scaling factor, which is some power-of-two integer. There is a crucial difference in the rescaling operation between the CKKS scheme and the RNS-CKKS scheme. While the CKKS scheme can rescale the ciphertext by the exact scaling factor, the RNS-CKKS scheme has to rescale the ciphertext by one of the RNS modulus, which is not equal to the scaling factor. Thus, the RNS-CKKS scheme allows approximation in the rescaling procedure. Detailed procedures in the RNS-CKKS scheme are described as follows.

Several independent messages are encoded into one polynomial by the canonical embedding before encryption. The canonical embedding σ embeds $a \in \mathbb{Q}[X]/\left<\Phi_M(X)\right>$ into an element of $\mathbb{C}^N$ whose elements are values of a evaluated at the distinct roots of $\Phi_M(X)$. It is a well-known fact that the roots of $\Phi_M(X)$ are exactly the power of odd integers of the M-th root of unity, and $\mathbb{Z}_M^* = \langle -1, 5\rangle$. Let $\mathbb{H} = \{(z_j)_{j\in\mathbb{Z}_M^*} : z_j = \overline{z_{-j}}\}$, and π be a natural projection from $\mathbb{H}$ to $\mathbb{C}^{N/2}$. Then, it is easily known that the range of σ is exactly $\mathbb{H}$. When $N/2$ complex number messages constitute an element in $\mathbb{C}^{N/2}$, each coordinate is called a slot. The encoding and decoding procedures are given as follows.

$\texttt{Ecd}(\boldsymbol{z};\Delta)$: For a vector $\boldsymbol{z} \in \mathbb{C}^{N/2}$, return

$$m(X) = \sigma^{-1}\left(\left\lfloor \Delta \cdot \pi^{-1}(\boldsymbol{z})\right\rceil_{\sigma(\mathcal{R})}\right) \in \mathcal{R},$$

where Δ is the scaling factor and $\left\lfloor \pi^{-1}(\boldsymbol{z})\right\rceil_{\sigma(\mathcal{R})}$ denotes the discretization (rounding) of $\pi^{-1}(\boldsymbol{z})$ into an element of $\sigma(\mathcal{R})$.

$\texttt{Dcd}(m;\Delta)$: For a polynomial $m(X) \in \mathcal{R}$, return a vector $\boldsymbol{z} \in \mathbb{C}^{N/2}$ whose entry of index j is $z_j = \left\lfloor \Delta^{-1} \cdot m(\zeta_M^{5^j})\right\rceil$ for $j \in \{0, 1, \cdots, N/2-1\}$, where ζ_M is the M-th root of unity.

Before describing the RNS-CKKS scheme, several basic operations for RNS system is defined: `Conv`, `ModUp`, and `ModDown`. Let $\mathcal{B} = \{p_0, p_1, \cdots, p_{k-1}\}, \mathcal{C} = \{q_0, q_1, \cdots, q_{\ell-1}\}$, and $\mathcal{D} = \{p_0, p_1, \cdots, p_{k-1}, q_0, q_1, \cdots, q_{\ell-1}\}$, where p_i's and q_j's are all distinct primes.

$\texttt{Conv}_{\mathcal{C}\to\mathcal{B}}$: It converts the RNS bases from $\mathcal{C}$ to $\mathcal{B}$ without the merge process of Chinese remainder theorem, which is defined as

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

where $[a]_{\mathcal{C}} = (a^{(0)}, \cdots, a^{(\ell-1)}) \in \mathbb{Z}_{q_0} \times \cdots \times \mathbb{Z}_{q_{\ell-1}}$ and $\hat{q}_j = \prod_{j'\neq j} q_{j'} \in \mathbb{Z}$.

$\texttt{ModUp}_{\mathcal{C}\to\mathcal{D}}$: It adds other moduli in $\mathcal{B}$ to the current RNS bases to expand the modulus space without changing the value as

$$\begin{aligned}\texttt{ModUp}_{\mathcal{C}\to\mathcal{D}}(\cdot) &: \prod_{j=0}^{\ell-1} R_{q_j} \to \prod_{i=0}^{k-1} R_{p_i} \times \prod_{j=0}^{\ell-1} R_{q_j} \\ &: [a]_{\mathcal{C}} \to (\texttt{Conv}_{\mathcal{C}\to\mathcal{B}}([a]_{\mathcal{C}}), [a]_{\mathcal{C}}).\end{aligned}$$

$\texttt{ModDown}_{\mathcal{D}\to\mathcal{C}}$: It removes the moduli in $\mathcal{B}$ from the current RNS bases with dividing the value by $P = \prod_{i=0}^{k-1} p_i$ as

$$\begin{aligned}\texttt{ModDown}_{\mathcal{D}\to\mathcal{C}}(\cdot) &: \prod_{i=0}^{k-1} R_{p_i} \times \prod_{j=0}^{\ell-1} R_{q_j} \to \prod_{j=0}^{\ell-1} R_{q_j} \\ &: ([a]_{\mathcal{B}}, [b]_{\mathcal{C}}) \to ([b]_{\mathcal{C}} - \texttt{Conv}_{\mathcal{B}\to\mathcal{C}}([a]_{\mathcal{B}})) \cdot [P^{-1}]_{\mathcal{C}}.\end{aligned}$$

Then, each procedure of the RNS-CKKS scheme is given as follows.

$\texttt{Setup}(q, L; 1^\lambda)$: Given a scaling factor Δ, the number of levels L, and a security parameter λ, several parameters are chosen as follows.

- A power-of-two degree N of the polynomial modulus of the ring is chosen so that the number of level L can be supported with the security parameter λ.

- A secret key distribution χ_{key}, an encryption key distribution χ_{enc}, and an error distribution χ_{err} over R are chosen considering the security parameter λ.
- A basis with prime numbers $\mathcal{B} = \{p_0, p_1, \cdots, p_{k-1}\}$ and $\mathcal{C} = \{q_0, q_1, \cdots, q_L\}$ is chosen so that $p_i \equiv 1 \mod 2N$, $q_j \equiv 1 \mod 2N$ for $0 \le i \le k-1, 0 \le j \le L$, and $|q_i - \Delta|$ is as small as possible. All prime numbers are distinct and $\mathcal{D} = \mathcal{B} \cup \mathcal{C}$. 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 $P = \prod_{i=0}^{k-1} p_i, Q = \prod_{j=0}^{L} q_j, \hat{p}_i = \prod_{0 \le i' \le k-1, i' \ne i} p_{i'}$ for $0 \le i \le k-1$, and $\hat{q}_{\ell,j} = \prod_{0 \le j' \le \ell, j' \ne j} q_{j'}$ for $0 \le j \le \ell \le L$. Then, the following numbers are computed.

- $[\hat{p}_i]_{q_j}$ and $[\hat{p}_i^{-1}]_{p_i}$ for $0 \le i \le k-1, 0 \le j \le L$
- $[P^{-1}]_{q_j}$ for $0 \le j \le L$
- $[\hat{q}_{\ell,j}]_{p_i}$ and $[\hat{q}_{\ell,j}^{-1}]_{q_j}$ for $0 \le i \le k-1, 0 \le j \le \ell \le L$

$\texttt{KSGen}(s_1, s_2)$**:** This procedure generates the switching key for switching the secret key s_1 to s_2 without changing the message in a ciphertext. Given $s_1, s_2 \in R$, sample $(a'^{(0)}, \cdots, a'^{(k+L)}) \leftarrow U\left(\prod_{i=0}^{k-1} R_{p_i} \times \prod_{j=0}^{L} R_{q_j}\right)$ and an error $e' \leftarrow \chi_{\mathsf{err}}$, and generate the switching key swk as

$$\left(\mathsf{swk}^{(0)} = (b'^{(0)}, a'^{(0)}), \cdots, \mathsf{swk}^{(k+L)} = (b'^{(k+L)}, a'^{(k+L)})\right) \in \prod_{i=0}^{k-1} R_{p_i}^2 \times \prod_{j=0}^{L} R_{q_j}^2,$$

where $b'^{(i)} \leftarrow -a'^{(i)} \cdot s_2 + e' \mod p_i$ for $0 \le i \le k-1$ and $b'^{(k+j)} \leftarrow -a'^{(k+j)} \cdot s_2 + [P]_{q_j} \cdot s_1 + e' \mod q_j$ for $0 \le j \le L$.

$\texttt{KeyGen}$**:** This procedure generates the secret key, the evaluation key, and the public key. Sample $s \leftarrow \chi_{\mathsf{key}}$ and set $\mathsf{sk} \leftarrow (1, s)$ as the secret key. The evaluation key is set by $\mathsf{evk} \leftarrow \texttt{KSGen}(s^2, s)$. Also, sample $(a^{(0)}, a^{(1)}, \cdots, a^{(L)}) \leftarrow$

$U\left(\prod_{j=0}^{L} R_{q_j}\right)$ and $e \leftarrow \chi_{\mathsf{err}}$ and the public key is generated as

$$\mathsf{pk} \leftarrow \left(\mathsf{pk}^{(j)} = (b^j, a^{(j)}) \in R_{q_j}^2\right)_{0 \le j \le L},$$

where $b^{(j)} \leftarrow -a^{(j)} \cdot s + e \mod q_j$ for $0 \le j \le L$.

$\mathtt{Enc}_{\mathsf{pk}}(m)$**:** For a message slot $\mathbf{z} \in \mathbb{C}^{N/2}$, generate the message polynomial by $m = \mathtt{Ecd}(\mathbf{z}; \Delta)$. Then, sample $v \leftarrow \chi_{\mathsf{enc}}$ and $e_0, e_1 \leftarrow \chi_{\mathsf{err}}$ and generate the ciphertext $\mathsf{ct} = (\mathsf{ct}^{(j)})_{0 \le j \le L} \in \prod_{j=0}^{L} R_{q_j}^2$, where $\mathsf{ct}^{(j)} \leftarrow v \cdot \mathsf{pk}^{(j)} + (m + e_0, e_1) \mod q_j$ for $0 \le j \le L$.

$\mathtt{Dec}_{\mathsf{sk}}(\mathsf{ct})$**:** For a ciphertext $\mathsf{ct} = (\mathsf{ct}^{(j)})_{0 \le j \le \ell} \in \prod_{j=0}^{\ell} R_{q_j}^2$, compute $\tilde{m} = \langle \mathsf{ct}^{(0)}, \mathsf{sk} \rangle \mod q_0$ and output $\mathbf{z} = \mathtt{Dcd}(\tilde{m}; \Delta)$.

$\mathtt{Add}(\mathsf{ct}_1, \mathsf{ct}_2)$**:** For two ciphertexts $\mathsf{ct}_r = \left(\mathsf{ct}_r^{(j)}\right)_{0 \le j \le \ell}$ for $r = 1, 2$, output the ciphertext $\mathsf{ct}_{\mathsf{add}} = \left(\mathsf{ct}_{\mathsf{add}}^{(j)}\right)_{0 \le j \le \ell}$, where $\mathsf{ct}_{\mathsf{add}}^{(j)} \leftarrow \mathsf{ct}_1^{(j)} + \mathsf{ct}_2^{(j)} \mod q_j$ for $0 \le j \le \ell$.

$\mathtt{Mult}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)$**:** For two ciphertexts $\mathsf{ct}_r = \left(\mathsf{ct}_r^{(j)} = (c_{r0}^{(j)}, c_{r1}^{(j)})\right)_{0 \le j \le \ell}$, compute the followings and output the ciphertext $\mathsf{ct}_{\mathsf{mult}} \in \prod_{j=0}^{\ell} R_{q_j}^2$.

- $d_0^{(j)} = c_{00}^{(j)} c_{10}^{(j)} \mod q_j$, $d_1^{(j)} = c_{00}^{(j)} c_{11}^{(j)} + c_{01}^{(j)} c_{10}^{(j)} \mod q_j$, and $d_2^{(j)} = c_{01}^{(j)} c_{11}^{(j)} \mod q_j$ for $0 \le j \le \ell$.
- $\mathtt{ModUp}_{\mathcal{C}_\ell \to \mathcal{D}_\ell}(d_2^{(0)}, d_2^{(1)}, \cdots, d_2^{(\ell)}) = (\tilde{d}_2^{(0)}, \tilde{d}_2^{(1)}, \cdots, \tilde{d}_2^{(k-1)}, d_2^{(0)}, d_2^{(1)}, \cdots, d_2^{(\ell)})$.
- $\tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}^{(k+\ell)} = (\tilde{c}_0^{(j)}, \tilde{c}_1^{(j)}))_{0 \le j \le k+\ell}$, where $\tilde{\mathsf{ct}^{(i)}} = \tilde{d}_2^{(i)} \cdot \mathsf{evk}^{(i)} \mod p_i$ and $\tilde{\mathsf{ct}^{(k+j)}} = d_2^{(j)} \cdot \mathsf{evk}^{(k+j)} \mod q_j$ for $0 \le i \le k-1$ and $0 \le j \le \ell$.
- $\left(\hat{c}_r^{(0)}, \hat{c}_r^{(1)}, \cdots, \hat{c}_r^{(\ell)}\right) = \mathtt{ModDown}_{\mathcal{D}_\ell \to \mathcal{C}_\ell}\left(\tilde{c}_r^{(0)}, \tilde{c}_r^{(1)}, \cdots, \tilde{c}_r^{(k+\ell)}\right)$ for $r = 0, 1$.
- $\mathsf{ct}_{\mathsf{mult}} = (\mathsf{ct}_{\mathsf{mult}}^{(j)})_{0 \le j \le \ell}$, where $\mathsf{ct}_{\mathsf{mult}}^{(j)} = \left(\hat{c}_0^{(j)} + d_0^{(j)}, \hat{c}_1^{(j)} + d_1^{(j)}\right) \mod q_j$ for $0 \le j \le \ell$.

$\mathtt{RS}(\mathsf{ct})$**:** For a ciphertext $\mathsf{ct} = \left(\mathsf{ct}^{(j)} = (c_0^{(j)}, c_1(j))\right)_{0 \le j \le \ell}$, output the ciphertext $\mathsf{ct}' = \left(\mathsf{ct}'^{(j)} = (c_0'^{(j)}, c_1'^{(j)})\right)_{0 \le j \le \ell-1}$, where $c_r'^{(j)} = q_\ell^{-1} \cdot \left(c_r^{(j)} - c_r^{(\ell)}\right) \mod q_j$ for $r = 0, 1$ and $0 \le j \le \ell - 1$.

There are additional homomorphic operations, rotation, and complex conjugation, which are used for homomorphic linear transformation in the bootstrapping of the RNS-CKKS scheme. Since these operations are not used in this dissertation, these operations are omitted in this chapter.

2.2 Key-Switching Operation and Galois Key

I now explain the key-switching operation [44] in BFV and CKKS schemes. This operation converts a ciphertext (b, a) that can be decrypted by a secret key s to another ciphertext (b', a') that can be decrypted by another secret key s' without changing the messages. It requires an evaluation key called the key-switching key, which is constructed as follows. Suppose that it is required to perform the key-switching operation switching the secret key from s to s'. The RNS moduli for key-switching are Q_i for $i = 0, \cdots, \mathtt{dnum} - 1$ and the special modulus is P, where dnum is defined to be the number of the RNS moduli decomposed for the key-switching operation. In this case, the RNS bases for these RNS moduli are $\hat{Q}_i \cdot [\hat{Q}_i^{-1}]_{Q_i}$, where $\hat{Q}_i$ means $\prod_{j \neq i} Q_j$. The special modulus P should be set to be larger than all Q_i's because of the noise reduction in the key-switching operation. The key-switching key is constructed as dnum ciphertexts, each of which is $(b_i, a_i) \in R_{PQ}^2$, where $a_i \leftarrow R_{PQ}$ and $b_i = -a_i \cdot s' + e + P \cdot \hat{Q}_i \cdot [\hat{Q}_i^{-1}]_{Q_i} \cdot s$. In the key-switching operation, a is first decomposed into the RNS elements of a with the ModUp operation, which is described in Algorithm 3. Each RNS element is multiplied by the ciphertext having the corresponding RNS basis in the key-switching key and added with each other. Then, the ciphertext and the modulus are divided by the special modulus with the $\mathsf{ModDown}$ operation. The whole algorithm for key-switching operation is described in Algorithm 4. This process of temporarily raising and reducing the modulus prevents the noise from amplifying, and the special modulus should be larger than all of the dnum RNS moduli used in the key-switching operation.

Algorithm 3 Decompose

```
Input: A ring element $a \in R_Q$ in the RNS form, where $Q = \prod_{i=0}^{\delta-1} Q_i$ and $Q_i$'s are
       pairwisely coprime, and the additional modulus $P$ coprime to $Q$.
Output: A vector of ring elements $(a_0, \cdots, a_{\delta-1}) \in R_{PQ}^{\delta}$, where $a_i = [a]_{Q_i} + Q_i \cdot \tilde{e}_i$
        for small $\tilde{e}_i$'s and $a_i$'s are in the RNS form.
for $i \leftarrow 0$ to $\delta - 1$ do
    $a_i \leftarrow$ ModUp for $[a]_{Q_i} \in R_{Q_i}$ from $R_{Q_i}$ to $R_{PQ}$.
return $(a_0, \cdots, a_{\delta-1}) \in R_{PQ}^{\delta}$
```

A trade-off for various performances occurs depending on the value of dnum. As the value of dnum increases, the computation amount in the key-switching operation increases due to the increase in the number of NTT/INTT operations and the amount of inner-product computation. Also, the size of the key-switching keys increases because the number of ciphertexts in the key-switching key is dnum. On the other hand, if the value of dnum is large, each RNS modulus used in the key-switching operation is small, making the special modulus small. Since the upper bound of the size of the total modulus is fixed with the specified security level, the available modulus for homomorphic computations, except the special modulus, can be large. This can accommodate a more deep homomorphic circuit without the bootstrapping operation or reduce the number of the bootstrapping operations when a deep homomorphic circuit is performed with the bootstrapping operations. The value of dnum is selected in consideration of these trade-offs.

If it is required to perform the rotation operation for cyclic shift r, the key-switching key for this operation can be constructed as above for $s' = s(X^{5^r})$. This key-switching key is called a Galois key for cyclic shift r of the corresponding message vector because this key is used for performing Galois automorphism $m(X) \mapsto m(X^{5^k})$ to encrypted message polynomial, which is equivalent to the rotation operations. The specific algorithm for the key-switching operation is shown in Algorithm 4.

Note that in this algorithm, a general case is dealt with when the modulus $\bar{Q}$ of a ciphertext is a divisor of the maximum evaluation modulus Q. The modulus Q can be simply replaced with $\bar{Q}$ in the key-switching operation with the same decomposed RNS moduli except the last RNS modulus. The non-trivial point is that

$$\{([b^{(i)}]_{P\bar{Q}}, [a^{(i)}]_{P\bar{Q}})\}_{i=0,\cdots,\mu-1} \in (R_{P\bar{Q}})^{\mu}$$

is a valid Galois key for the evaluation modulus $\bar{Q}$ and the special modulus P. For ease of understanding, the proof for this fact is proved in the following theorem.

Assume that

$$\{(b^{(i)}, a^{(i)})\}_{i=0,\cdots,\mathtt{dnum}-1} \in (R_{PQ})^{\mathtt{dnum}}$$

is a valid shift-r Galois key for the evaluation modulus Q and the special modulus P. Let $\bar{Q} = (\prod_{i=0}^{\mu-2} Q_i) \cdot \bar{Q}_{\mu-1}$, where $\bar{Q}_{\mu-1}$ is a divisor of $Q_{\mu-1}$ and $\mu \leq \mathtt{dnum}$. Then, the shift-r Galois key

$$\{([b^{(i)}]_{P\bar{Q}}, [a^{(i)}]_{P\bar{Q}})\}_{i=0,\cdots,\mu-1} \in (R_{P\bar{Q}})^{\mu}$$

is valid for the evaluation modulus $\bar{Q}$ and the special modulus P.

Proof. Since the Galois key

$$\{(b^{(i)}, a^{(i)})\}_{i=0,\cdots,\mathtt{dnum}-1} \in (R_{PQ})^{\mathtt{dnum}}$$

is valid, it is obtained

$$b^{(i)} + a^{(i)} \cdot s = P \cdot \hat{Q}_i \cdot [\hat{Q}_i^{-1}]_{Q_i} \cdot s(X^{5^r}) + e_i \in R_{PQ}$$

for all i and small error e_i's. If the modular reduction is performed to $b^{(i)} + a^{(i)} \cdot s$ by each Q_j for $0 \leq j \leq \mu - 1$, it is obtained

$$[b^{(i)} + a^{(i)} \cdot s]_{Q_j} = \begin{cases} [P]_{Q_i} \cdot s(X^{5^r}) + e_i & \text{if } i = j \\ e_i & \text{if } i \neq j. \end{cases} \tag{2.1}$$

If the modular reduction is performed to $b^{(i)} + a^{(i)} \cdot s$ by each P, it is obtained $[b^{(i)} + a^{(i)} \cdot s]_P = e_i$. Since $\bar{Q}_{\mu-1}$ is a divisor of $Q_{\mu-1}$, $Q_{\mu-1}$ can be replaced in (2.1) with $\bar{Q}_{\mu-1}$ for all i and j.

On the other hand, the following ring element

$$P \cdot \hat{\bar{Q}}_i \cdot [\hat{\bar{Q}}_i^{-1}]_{\bar{Q}_i} \cdot s(X^{5^r}) + e_i \in R_{P\bar{Q}}$$

is considered, where $\bar{Q}_i = Q_i$ for $0 \le i \le \mu - 2$ and $\hat{\bar{Q}}_i = \prod_{j=0, j \neq i}^{\mu-1} \bar{Q}_i$. Note that it is obtained

$$[P \cdot \hat{\bar{Q}}_i \cdot [\hat{\bar{Q}}_i^{-1}]_{\bar{Q}_i} \cdot s(X^{5^r}) + e_i]_{\bar{Q}_j} = \begin{cases} [P]_{\bar{Q}_i} \cdot s(X^{5^r}) + e_i & \text{if } i = j \\ e_i & \text{if } i \neq j. \end{cases}$$

If the modular reduction is performed to $P \cdot \hat{\bar{Q}}_i \cdot [\hat{\bar{Q}}_i^{-1}]_{\bar{Q}_i} \cdot s(X^{5^r}) + e_i$ by each P, it is obtained $[P \cdot \hat{\bar{Q}}_i \cdot [\hat{\bar{Q}}_i^{-1}]_{\bar{Q}_i} \cdot s(X^{5^r}) + e_i]_P = e_i$.

Since $b^{(i)} + a^{(i)} \cdot s$ and $P \cdot \hat{\bar{Q}}_i \cdot [\hat{\bar{Q}}_i^{-1}]_{\bar{Q}_i} \cdot s(X^{5^r}) + e_i$ have the same remainders for all $\bar{Q}_i$'s and P, the two values is equal to each other in modulo $P\bar{Q}$ by the Chinese remainder theorem. Thus, it is obtained

$$\begin{aligned} [b^{(i)}]_{P\bar{Q}} + [a^{(i)}]_{P\bar{Q}} \cdot s &= [b^{(i)} + a^{(i)} \cdot s]_{P\bar{Q}} \\ &= P \cdot \hat{\bar{Q}}_i \cdot [\hat{\bar{Q}}_i^{-1}]_{\bar{Q}_i} \cdot s(X^{5^r}) + e_i \end{aligned}$$

for all i's. Thus, the shift-r Galois key

$$\{([b^{(i)}]_{P\bar{Q}}, [a^{(i)}]_{P\bar{Q}})\}_{i=0,\cdots,\mu-1} \in (R_{P\bar{Q}})^{\mu}$$

is valid for the evaluation modulus $\bar{Q}$ and the special modulus P. □

2.3 Bootstrapping of CKKS Scheme

The framework of the bootstrapping of the CKKS scheme was introduced in [19], which is the same as the case of the RNS-CKKS scheme. The purpose of bootstrapping is to refresh the ciphertext of level 0, whose multiplication cannot be performed anymore, to the fresh ciphertext of level L having the same messages. Bootstrapping is composed of the following four steps:

Algorithm 4 Key-Switching Operation [44]

```
Input: A key-switching key from $s$ to $s'$, $\mathsf{swk} = \{(b^{(i)}, a^{(i)}\}_{i=0,\cdots,\mathtt{dnum}-1} \in (R_{PQ}^2)^{\mathtt{dnum}}$ for $Q = \prod_{i=0}^{\mathtt{dnum}-1} Q_i$, and a ciphertext $(b, a) \in R_{\bar{Q}}^2$ encrypted with secret key $s \in R$ for $\bar{Q} = (\prod_{i=0}^{\mu-2} Q_i) \cdot \bar{Q}_{\mu-1}$, where $\bar{Q}_{\mu-1}$ is a divisor of $Q_{\mu-1}$ and $\mu \leq \mathtt{dnum}$.
Output: A ciphertext $(b', a') \in R_{\bar{Q}}^2$ encrypted with secret key $s' \in R$
Decompose $a$ into a vector $(a_0, \cdots, a_{\mu-1}) \in R_{P\bar{Q}}^{\mu}$, where $a_i = [a]_{Q_i} + Q_i \cdot \tilde{e}_i$ for small $\tilde{e}_i$'s for $0 \leq i \leq \mu - 2$ and $a_{\mu-1} = [a]_{\bar{Q}_{\mu-1}} + \bar{Q}_{\mu-1} \cdot \tilde{e}_{\mu-1}$ for small $\tilde{e}_{\mu-1}$.
$(\bar{b}, \bar{a}) \leftarrow (0, 0) \in R_{P\bar{Q}}^2$
for $i \leftarrow 0$ to $\mu - 1$ do
    $(\bar{b}, \bar{a}) \leftarrow (\bar{b}, \bar{a}) + a_i \cdot ([b^{(i)}]_{P\bar{Q}}, [a^{(i)}]_{P\bar{Q}})$
$(b', a') \leftarrow (\lfloor P^{-1} \cdot \bar{b} \rceil, \lfloor P^{-1} \cdot \bar{a} \rceil) \in R_{\bar{Q}}^2$
$b' \leftarrow b' + b$
return $(b', a')$
```

i) Modulus raising

ii) Homomorphic linear transformation; CoeffToSlot

iii) Homomorphic modular reduction

iv) Homomorphic linear transformation; SlotToCoeff

Modulus Raising:

The starting point of bootstrapping is modulus raising, where the ciphertext of level 0 is simply considered as an element of $\mathcal{R}_Q^2$, instead of $\mathcal{R}_{q_0}^2$. Since the ciphertext of level 0 is supposed to be $\langle \mathsf{ct}, \mathsf{sk} \rangle \approx m \mod q_0$, it is obtained $\langle \mathsf{ct}, \mathsf{sk} \rangle \approx m + q_0 I \mod Q$ for some $I \in \mathcal{R}$ when it is decrypted. It is assured that the absolute values of coefficients of I are rather small, for example, usually smaller than 12, because coefficients of sk consist of small numbers [20]. The crucial part of the bootstrapping of the CKKS scheme is to make ct' such that $\langle \mathsf{ct}', \mathsf{sk} \rangle \approx m \mod q_L$. This is divided into two parts: homomorphic linear transform and homomorphic evaluation of modular reduction function.

Homomorphic Linear Transformation:

The ciphertext ct after modulus raising can be considered as the ciphertext encrypting $m + q_0 I$, and thus the modular reduction should be performed to coefficients of message polynomial homomorphically. However, the operations are all for slots, not coefficients of the message polynomial. Thus, to perform some meaningful operations on coefficients, ct should be converted into a ciphertext that encrypts coefficients of $m+q_0 I$ as its slots. After evaluation of homomorphic modular reduction function, this ciphertext should be reversely converted into the other ciphertext ct' that encrypts the slots of the previous ciphertext as the coefficients of its message. These two operations are called CoeffToSlot and SlotToCoeff operations. These operations are regarded as homomorphic evaluation of encoding and decoding of messages, which are a linear transformation by some variants of Vandermonde matrix for roots of $\Phi_M(x)$. This can be performed by general homomorphic matrix multiplication [20], or FFT-like operation [16].

Homomorphic Modular Reduction Function:

After CoeffToSlot is performed, it is required to perform modular reduction homomorphically on each slot in modulus q_0. This procedure is called EvalMod. This modular reduction function is not an arithmetic function and even not a continuous function. Fortunately, by restricting the range of the messages such that m/q_0 is small enough, the approximation region can be given only near multiples of q_0. This allows me to approximate the modular reduction function more effectively. Since the operations that the CKKS supports are arithmetic operations, most of the works [16, 20, 44] dealing with CKKS bootstrapping approximate the modular reduction function with some polynomials, which are sub-optimal approximate polynomials.

The scaling factor is increased when the bootstrapping is performed because m/q_0 needs to be very small in the homomorphic modular reduction function. In this dissertation, the default scaling factor means the scaling factor used in the intended applications, and the bootstrapping scaling factor means the scaling factor used in the bootstrapping. The bit-length difference between these two scaling factors is usually 10.

2.4 Comparison Operation for FHE

Lee et al. [57] showed that the ReLU function have to be approximated with sufficiently high precision if the pre-trained model parameters with the original ResNet-20 model is used. 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 [54]. It reduces the running time of the homomorphic evaluation of the ReLU function, and this approximation method makes the homomorphic evaluation of non-arithmetic functions, such as the ReLU function, more practical.

Lee et al. [54] 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.

2.5 Approximation Theory

Approximation theory is needed to prove the convergence of the minimax polynomial obtained by the proposed improved multi-interval Remez algorithm. Assume that functions are defined on a union of the finite number of closed and bounded intervals in the real line. From the following well-known theorem [66] in real analysis, it is convinced that this domain of functions is a compact set.

Theorem 2.5.1 ([66] Bolzano-Weierstrass Theorem). *A subset of $\mathbb{R}^n$ is a compact set if and only if it is closed and bounded.*

A union of the finite number of closed and bounded intervals in the real line is trivially closed and bounded, and thus this domain is a compact set by Bolzano-Weierstrass theorem. This theorem will be used in the convergence proof of the improved multi-interval Remez algorithm in Section 3.1.2.

The next theorem [66] states that any continuous function on a compact set in the real line can be approximated with an arbitrarily small error by polynomial approximation. In fact, the theorem includes the case of continuous functions on more general domains, but the special case on compact sets in the real line is only used in this dissertation.

Theorem 2.5.2 ([66] Stone-Weierstrass Theorem). *Assume that f is a continuous function on the compact subset D of the real line. For every $\epsilon > 0$, there is a polynomial p such that $\|f - p\|_\infty < \epsilon$.*

There are many theorems for the minimax approximate polynomials of a function defined on a compact set in approximation theory. Before the introduction of these theorems, the definition of the Haar condition of a set of functions is refered to, which deals with the generalized version of power bases used in polynomial approximation and its equivalent statement. It is a well-known fact that the power basis $\{1, x, x^2, \cdots, x^d\}$ satisfies the Haar condition. Thus, if an argument deals with the polynomials with regard to a set of basis functions satisfying Haar condition, it naturally includes the case of polynomials.

Definition 2.5.1 ([18] Haar's Condition). *A set of functions $\{g_1, g_2, \cdots, g_n\}$ satisfies the Haar condition if each g_i is continuous and if each determinant*

$$D[x_1, \cdots, x_n] = \begin{vmatrix} g_1(x_1) & \cdots & g_n(x_1) \\ \vdots & \ddots & \vdots \\ g_1(x_n) & \cdots & g_n(x_n) \end{vmatrix}$$

for any n distinct points $x_1, \cdots, x_n$ is not zero.

Lemma 2.5.3 ([18]). *A set of functions $\{g_1, \cdots, g_n\}$ satisfies the Haar condition if and only if zero polynomial is the only polynomial $\sum_i c_i g_i$ that has more than $n-1$ roots.*

Firstly, there is the unique minimax approximate polynomial in the union of the finite number of closed and bounded intervals as in the following two theorems.

Theorem 2.5.4 ([18] Existence of Best Approximations). *Let $\mathcal{F}$ be a normed linear space, and f is any fixed element in $\mathcal{F}$. If $\mathcal{S}$ is a linear subspace of $\mathcal{F}$ with finite dimension, $\mathcal{S}$ contains at least one element of minimum distance from f.*

Theorem 2.5.5 ([18] Haar's Unicity Theorem). *Let f be any continuous function on a compact set K. Then the minimax polynomial $\sum_i c_i g_i$ of f is unique if and only if $\{g_1, g_2, \cdots, g_n\}$ satisfies the Haar condition.*

In Theorem 2.5.4 for the existence of the best approximation, consider a set $\mathcal{F}$ of continuous functions on a union D of the finite number of closed and bounded intervals. It can be easily known that $\mathcal{F}$ is a linear space with a max-norm $\|f\|_\infty = \max_{x \in D} |f(x)|$. The set $\mathcal{P}_d$ of polynomials with regard to the finite number of basis functions on D is a finite-dimensional linear subspace. Then, from Theorem 2.5.4, there is at least one minimax approximate polynomial for any $f \in \mathcal{F}$.

The core property of the minimax approximate polynomial for a function on D is introduced.

Theorem 2.5.6 ([18] Chebyshev Alternation Theorem). *Let $\{g_1, \cdots, g_n\}$ be a set of continuous functions defined on $[a, b]$ satisfying the Haar condition, and let D be a closed subset of $[a, b]$. A polynomial $p = \sum_i c_i g_i$ is the minimax approximate polynomial on D to any given continuous function f defined on D if and only if there are $n + 1$ distinct elements $x_0 < \cdots < x_n$ in D such that for the error function $r = f - p$ restricted on D,*

$$r(x_i) = -r(x_{i-1}) = \pm \sup_{x \in D} |r(x)|.$$

This condition is also called the equioscillation condition. This means that if we find a polynomial satisfying the equioscillation condition, then this is the unique minimax approximate polynomial, needless to compare with the maximum approximation error of any polynomials.

The following three theorems are used to prove the convergence of the improved multi-interval Remez algorithm in Section 3.1.2.

Theorem 2.5.7 ([18] de La Vallee Poussin Theorem). *Let $\{g_1, \cdots, g_n\}$ be a set of continuous functions on $[a, b]$ satisfying the Haar condition. Let f be a continuous on $[a, b]$, and p be a polynomial such that $p - f$ has alternately positive and negative values at $n + 1$ consecutive points x_i in $[a, b]$. Let p^* be a minimax approximate polynomial for f, and $e(f)$ be the minimax approximation error of p^*. Then, it is obtained*

$$e(f) \geq \min_i |p(x_i) - f(x_i)|.$$

Lemma 2.5.8 ([18]). *Let $\{g_1, \cdots, g_n\}$ be a set of continuous functions satisfying the Haar condition. Assume that $x_1 < \cdots < x_n$ and $y_1 < \cdots < y_n$. Then the determinants $D[x_1, \cdots, x_n]$ and $D[y_1, \cdots, y_n]$, defined by Definition 2.5.1, have the same sign.*

Theorem 2.5.9 ([18] Strong Unicity Theorem). *Let $\{g_1, \cdots, g_n\}$ be a set of functions satisfying the Haar condition, and let p^* be the minimax polynomial of a given continuous function u. Then, there is a constant $\gamma > 0$ determined by f such that for any polynomial p, it is obtained*

$$\|p - f\|_\infty \geq \|p^* - f\|_\infty + \gamma \|p - p^*\|_\infty.$$

Remez algorithm [18,62,65] is an iterative algorithm that always returns the minimax approximate polynomial for any continuous function on an interval of $[a, b]$. This algorithm strongly uses the Chebyshev alternation theorem [18] in that its purpose is finding the polynomial satisfying the equioscillation condition. In fact, the Remez algorithm can be applied to obtain the minimax approximate polynomial, whose basis function $\{g_1, \cdots, g_n\}$ satisfies the Haar condition. The following explanation includes the generalization of the Remez algorithm, and if we want to obtain the minimax approximate polynomial of degree d, we can choose the basis function $\{g_1, \cdots, g_n\}$ by the power basis $\{1, x, \cdots, x^d\}$, where $n = d + 1$.

Remez algorithm firstly initializes the set of reference points $\{x_1, \cdots, x_{n+1}\}$, which will be converged to the extreme points of the minimax approximate polynomial. Then, it obtains the minimax approximate polynomial in regard to only the set of reference points. Since the set of reference points is the set of finite points in $[a, b]$, it is a closed subset of $[a, b]$, and thus Chebyshev alternation theorem holds on the set of reference points. Let $f(x)$ be a continuous function on $[a, b]$. The minimax approximate polynomial on the set of reference points is exactly the polynomial $p(x)$ with the basis $\{g_1, \cdots, g_n\}$ satisfying

$$p(x_i) - f(x_i) = (-1)^i E \quad i = 1, \cdots, d + 2$$

for some real number E. This forms a system of linear equations having $n+1$ equations and $n+1$ variables of n coefficients of $p(x)$ and E, which is ensured to be not singular by Haar's condition, and thus the polynomial $p(x)$ is obtained. Then, n zeros of $p(x) - f(x)$ can be found, z_i between x_i and x_{i+1}, $i = 1, 2, \cdots, n$, and $n + 1$ extreme points $y_1, \cdots, y_{n+1}$ of $p(x) - f(x)$ can be found in each $[z_{i-1}, z_i]$, where $z_0 = a$ and $z_{n+1} = b$. That is, the minimum point of $p(x) - f(x)$ in $[z_{i-1}, z_i]$ is chosen if $p(x_i) - f(x_i) < 0$, and the maximum point of $p(x) - f(x)$ in $[z_{i-1}, z_i]$ is chosen if $p(x_i) - f(x_i) > 0$. Thus, a new set of extreme points $y_1, \cdots, y_{n+1}$ is found. If this satisfies equioscillation condition, the Remez algorithm returns $p(x)$ as the minimax approximate polynomial from the Chebyshev alternation theorem. Otherwise, it replaces the set of reference points with these extreme points $y_1, \cdots, y_{n+1}$ and processes above steps again. This is the Remez algorithm in Algorithm 5. The Remez algorithm is proved to be always converged to the minimax approximate polynomial by the following theorem.

Theorem 2.5.10 ([18] Convergence of Remez Algorithm). *Let $\{g_1, \cdots, g_n\}$ be a set of functions satisfying the Haar condition, p_k be a polynomial generated in the k-th iteration of Remez algorithm, and p^* be the minimax polynomial of a given f. Then, p_k converges uniformly to p^* by the following inequality,*

$$\|p_k - p^*\|_\infty \leq A\theta^k,$$

where A is a non-negative constant, and $0 < \theta < 1$.

2.6 Graph-Theoretic Algorithms

An arborescence in a given directed graph is a directed subgraph in which a single path exists on any node from a specific root node, and a spanning arborescence is an arborescence having paths from the root node to all nodes in the graph. The minimum spanning tree problem is the problem of finding a spanning tree whose sum of edge weights is minimum. This problem is also known to be solved within polynomial time, and Edmonds' algorithm is known to solve this problem [29], shown in Algorithm 6.

A spanning tree in a given undirected graph is a subgraph in a given graph such that all edges and all nodes are connected and there is no cycle in the subgraph. The

Algorithm 5 Remez Algorithm [18, 62, 65]

```
Input  : An input domain $[a,b]$, a continuous function $f$ on $[a,b]$, an approximation
         parameter $\delta$, and a basis $\{g_1,\cdots,g_n\}$.
Output: The minimax approximate polynomial $p$ for $f$

Select $x_1, x_2, \cdots, x_{d+2} \in [a,b]$ in strictly increasing order.
Find the polynomial $p(x) = \sum_{i=1}^{n} c_i g_i(x)$ with $p(x_i) - f(x_i) = (-1)^i E$ for $i =$
  $1, \cdots, d+2$ and some $E$ by solving the system of linear equations with variables $c_i$'s
  and $E$.
Divide the interval into $n+1$ sections $[z_{i-1}, z_i]$, $i = 1, \cdots, n+1$, from zeros
  $z_1, \cdots, z_n$ of $p(x) - f(x)$, where $x_i < z_i < x_{i+1}$, and boundary points $z_0 =$
  $a, z_{n+1} = b$.
Find the maximum (resp. minimum) points for each section when $p(x_i) - f(x_i)$ has
  positive (resp. negative) value. Denote these extreme points $y_1, \cdots, y_{n+1}$.
$\epsilon_{\mathsf{max}} \leftarrow \max_i |p(y_i) - f(y_i)|$
$\epsilon_{\mathsf{min}} \leftarrow \min_i |p(y_i) - f(y_i)|$
if $(\epsilon_{\mathsf{max}} - \epsilon_{\mathsf{min}})/\epsilon_{\mathsf{min}} < \delta$ then
    return $p(x)$
else
    Replace $x_i$'s with $y_i$'s and go to line 2.
end
```

minimum spanning tree problem is the problem of finding a spanning tree whose sum of edge weights is minimum. There are many algorithms for this, but Prim's algorithm [63] will be used, which is appropriate for the dense graph in this dissertation because a complete graph is used in this dissertation, shown in Algorithm 7.

Algorithm 6 Edmonds' Algorithm[29]

```
Input: A directed graph $G = (V, E)$ with an edge weight $w(e)$ for all $e \in E$, the root
       node $v_r \in V$
Output: A minimum spanning arborescence $G' = (V, E')$ from the root node $v_r$
Remove all edges to the root node $v_r$ from $E$
$E' \leftarrow \varnothing$
for $v \in V \backslash \{v_r\}$ do
    $\pi(v) \leftarrow$ the node such that an edge $(\pi(v), v) \in E$ has the minimum weight among
     edges to $v$
    $E' \leftarrow E' \cup \{(\pi(v), v)\}$
if $G' = (V, E')$ has no cycle then
    return $G' = (V, E')$
else
    $C = (V_c, E_c) \leftarrow$ a cycle in $G'$
    $\bar{V} \leftarrow (V \backslash V_c) \cup \{v_c\}$ for new node $v_c$
    $\bar{E} \leftarrow E \backslash E_c$
    for $(v_1, v_2) \in E$ such that $v_1 \in V \backslash V_c, v_2 \in V_c$ do
        Generate an edge $(v_1, v_c)$ with a weight $w(v_1, v_c) = w(v_1, v_2) - w(\pi(v_2), v_2)$
        $\bar{E} \leftarrow (\bar{E} \backslash \{(v_1, v_2\}) \cup \{(v_1, v_c)\}$
    for $(v_1, v_2) \in E$ such that $v_1 \in V_c, v_2 \in V \backslash V_c$ do
        if $(v_c, v_2) \notin \bar{E}$ or $w(v_c, v_2) > w(v_1, v_2)$ then
            Generate (or update) an edge $(v_c, v_2)$ with a weight $w(v_c, v_2) = w(v_1, v_2)$
            $\bar{E} \leftarrow \bar{E} \backslash \{(v_1, v_2)\} \cup \{(v_c, v_2)\}$
    $\bar{G'} = (\bar{V}, \bar{E'}) \leftarrow$Edmonds' algorithm for $(\bar{V}, \bar{E})$ with $v_r$
    $E' \leftarrow$ all edges in $E$ that correspond to edges in $\bar{E}'$
    $v_t \leftarrow$ the node such that $(u, v_t) \in E'$ corresponds to $(u, v_c) \in \bar{E}'$
    $E' \leftarrow (E' \cup E_c) \backslash \{(\pi(v_t), v_t)\}$
    return $G' \leftarrow (V, E')$
```

Algorithm 7 Prim's Algorithm[63]

```
Input: An undirected graph $G = (V, E)$ with an edge weight $w(e)$ for all $e \in E$
Output: A minimum spanning tree $G' = (V, E')$
Initialize $G' = (V', E')$, where $V' \leftarrow \{v\}, E' \leftarrow \varnothing$ for randomly selected $v \in V$
while $V' \neq V$ do
    $\bar{E} \leftarrow \{(v_1, v_2) | v_1 \in V', v_2 \in V \backslash V'\}$
    Find an edge $\bar{e} = (\bar{v}_1, \bar{v}_2) \in \bar{E}$ having the minimum edge weight among $\bar{E}$
    $V' \leftarrow V' \cup \{\bar{v}_2\}$
    $E' \leftarrow E' \cup \{\bar{e}\}$
return $G' = (V, E')$
```

Chapter 3

HIGH PRECISION BOOTSTRAPPING FOR RNS-CKKS SCHEME

Since the CKKS scheme includes noises used to ensure security as the approximate error in the message, the use of the RNS-CKKS scheme requires more sensitivity to the precision of the message than other homomorphic encryption schemes that support accurate decryption and homomorphic evaluation. This can be more sensitive for large-depth homomorphic operations because errors are likely to be amplified by the operations and distort the data significantly. Fortunately, the basic homomorphic operations in the RNS-CKKS scheme can ensure sufficiently high precision for practical use, but this is not the case for the bootstrapping operation. Ironically, while the bootstrapping operation in other homomorphic encryption schemes reduces the effect of the errors on messages so that they do not distort messages, the bootstrapping operation in the CKKS scheme amplifies the errors, which makes it the most major cause of data distortion among any other homomorphic operations in the RNS-CKKS scheme. Since advanced operations with large depth may require bootstrapping operation many times, the message precision problem in the bootstrapping operation is a crucial obstacle to applying the RNS-CKKS scheme to advanced applications.

Although the RNS-CKKS scheme is currently one of the most potential solutions to implement privacy-preserving machine learning (PPML) system [6,7,28], the methods for the PPML studied so far have mainly been applied to simple models such as MNIST, which has such a low depth that bootstrapping is not required. Thus, the message precision problem in the bootstrapping operation in the RNS-CKKS scheme did not need to be considered in the PPML model until now. However, the advanced machine learning model currently presented requires a large depth, and thus the bootstrapping operation is required and cannot avoid the message precision problem in the bootstrapping operation. Of course, the fact that bootstrapping requires longer running time and larger depth than other homomorphic operations is also pointed out as a major limitation of bootstrapping. While these points may be improved by simple parameter adjustments and using hardware optimization, the message precision problem in bootstrapping is difficult to solve with these simple methods.

Most of the works about PPML with FHE focused on the inference process rather than the training process because of the large running time. However, training neural networks with encrypted data is actually more important from a long-term perspective for solving the real security problem in machine learning, in that the companies cannot gather sufficiently many important but sensitive data, such as genetic or financial information so that they cannot construct the deep learning model for them because of the privacy of the data owners. While the inference process does not need a high precision number system, the training process is affected sensitively by the precision of the number system. Chen et al. [17] showed that convolutional neural networks (CNN) learning MNIST could not converge when the model is trained using a 16-bit fixed-point number system. When the 32-bit fixed-point number system is used to train the CNN with MNIST, the training performance was slightly lower than the case of using the single-precision floating-point number system, although all bits except one bit representing the sign are used to represent the data in 32-bit fixed-point number system, which is much better precision than the single-precision floating-point number system, which is 23-bit precision. Although many works proposed to use low-precision fixed-point numbers in the training procedure, they used additional special techniques, such as stochastic rounding [40] or the dynamic fixed-point number system [41], which cannot be supported by the RNS-CKKS scheme until now.

While most of the deep learning systems use single-precision floating-point numbers, the maximum precision achieved with the bootstrapping of the CKKS scheme in the previous papers was about only 20 bits. Considering that the CKKS scheme only supports fixed-point arithmetic, the 20-bit precision is not large enough to be applied wholly to the deep learning system. Thus, to apply the RNS-CKKS scheme to deep learning systems, it is necessary to achieve a precision sufficiently better than the 32-bit fixed-point precision, which requires a breakthrough for the bootstrapping in the RNS-CKKS scheme concerning its precision.

In this dissertation, I propose two methods to improve the bootstrapping operation of the RNS-CKKS scheme. Firstly, a fast algorithm is obtained, called an improved multi-interval Remez algorithm, obtaining the optimal minimax approximate polynomial of any continuous functions over any union of the finite number of intervals, which include the modular reduction function and the scaled sine/cosine function over the union of the approximation regions. Although the previous works have suggested methods to obtain polynomials that approximate the scaled sine/cosine function well from the minimax perspective, which are used to approximate the modular reduction function, these methods cannot obtain the optimal minimax approximate polynomial.

The original multi-interval Remez algorithm is not theoretically proven to obtain the minimax approximate polynomial, and it is only practically used for two or three approximation regions in the finite impulse response filter design, while it is needed to approximate functions over the union of tens of intervals. Furthermore, it takes impractically much time if this algorithm is used without further improvement to obtain a polynomial that can be used for the bootstrapping. To make the multi-interval Remez algorithm practical, the multi-interval Remez algorithm is modified as the improved multi-interval Remez algorithm. Then the correctness of the improved multi-interval

Remez algorithm is proved, including the original multi-interval Remez algorithm, for the union of any finite number of intervals. Since it can obtain the optimal minimax approximate polynomial in seconds, the polynomial can be even adaptively obtained when some parameters are abruptly changed on processing the ciphertexts so that the approximate polynomial should be updated. All polynomial approximation methods proposed in previous works for bootstrapping in the CKKS scheme can be replaced with the improved multi-interval Remez algorithm, which ensures the best quality of the approximation. It ensures to use the least degree of the approximate polynomial for a given amount of error.

Next, the composite function method is proposed to enlarge the approximation region in the homomorphic modular reduction process using the inverse sine function. The crucial point in the bootstrapping precision is that the difference between the modular reduction function and the sine/cosine function gives a significant precision loss. All previous works have used methods that approximate the modular reduction function as a part of the sine/cosine functions. This approximation has an inherent approximation error so that the limitation of the precision occurs. Besides, to ensure that these two functions are significantly close to each other, the approximation region has to be reduced significantly. They set the half-width of one interval in the approximation region as 2^{-10}, which is equal to the ratio of default scaling factor to the scaling factor used in the bootstrapping. The message has to be scaled by multiplying 2^{-10} to make the message into the approximation region, and it is scaled by multiplying 2^{10} at the end of the bootstrapping. Thus, the precision error in the computation for bootstrapping is amplified by 2^{10}, and the 10-bit precision loss occurs. If one tries to reduce this precision loss by enlarging the approximation region, the approximation error by the sine/cosine function becomes large, and thus the overall precision becomes lower than before.

Therefore, it is proposed to compose the optimal approximate polynomial of the inverse sine function to the sine/cosine function, since composing the inverse sine function to the sine/cosine function extends the approximation region of the modular reduction function, which makes it possible to improve the precision of the bootstrapping. Note that the inverse sine function in this situation has only one interval in the approximation region, and thus the small approximate error can be reached with relatively low degree polynomials. The minimax approximate polynomials for the scaled cosine function and the inverse sine function with sufficiently small minimax error are obtained by the improved multi-interval Remez algorithm. These polynomials are applied in the homomorphic modular reduction process by homomorphically evaluating the approximate polynomial for the scaled cosine function, several double-angle formulas, and the approximate polynomial for the inverse sine function. This enables me to minimize the inevitable precision loss by approximating the modular reduction function to the sine/cosine function.

Since the previous works do not focus on the maximum precision of the bootstrapping of the RNS-CKKS scheme, the maximum precision of the bootstrapping is checked with the previous techniques. The detailed relation with the precision of the bootstrapping and various parameters is analyzed with `SEAL` library. With the proposed methods, the approximation error in the bootstrapping of the RNS-CKKS scheme is reduced by 1/1176$\sim$1/42 (5.4$\sim$10.2-bit precision improvement) for each parameter setting. While the bootstrapping without the composite function method has 27.2$\sim$30.3-bit precision at maximum, the bootstrapping with the proposed composite function method has 32.6$\sim$40.5-bit precision, which are better precision than 32-bit fixed-point precision.

3.1 Improved Multi-Interval Remez Algorithm

Since the Remez algorithm works only when the approximation region is one interval, the multi-interval Remez algorithm is needed for the union of several intervals as an approximation region. The above Remez algorithm can be extended to the multiple sub-intervals of an interval [34, 61, 65]. The multi-interval Remez algorithm is the same as Algorithm 5, except Steps 3 and 4. For each iteration, firstly, it is required to find all of the local extreme points of the error function $p - f$ whose absolute error values are larger than the absolute error values at the current reference points. Then, $n + 1$ new extreme points are chosen among these points satisfying the following two criteria:

i) The error values alternate in sign.

ii) A new set of extreme points includes the global extreme point.

These two criteria are known to ensure the convergence to the minimax polynomial, even though there is no exact proof of its convergence to the best of my knowledge. However, it is noted that there are many choices of sets of extreme points satisfying these criteria. In the next subsection, the multi-interval Remez algorithm is modified, where one of the two criteria is changed.

In this dissertation, an improved multi-interval Remez algorithm is obtained for obtaining the optimal minimax approximate polynomial. With this proposed algorithm, the optimal minimax approximate polynomial can be obtained for continuous function on the union of finitely many closed intervals to apply the Remez algorithm to the bootstrapping of the CKKS scheme. The function to be approximated is the normalized modular reduction function defined in only near finitely many integers given as

$$\mathsf{normod}(x) = x - \mathsf{round}(x), \quad x \in \bigcup_{i=-(K-1)}^{K-1} [i - \epsilon, i + \epsilon],$$

where K determines the number of intervals in the domain, and normod function corresponds to the modular reduction function scaled for both its domain and range.

In addition, Han et al. [44] uses the cosine function to approximate $\mathsf{normod}(x)$ to use double-angle formula for efficient homomorphic evaluation. If the double-angle formula is used ℓ times, it is required to approximate the following cosine function

$$\cos\left(\frac{2\pi}{2^\ell}\left(x-\frac{1}{4}\right)\right), \quad x \in \bigcup_{i=-(K-1)}^{K-1} [i-\epsilon, i+\epsilon].$$

To design an approximation algorithm that deals with the above two functions, the general continuous function defined on an union of finitely many closed intervals is assumed, which is given as

$$D = \bigcup_{i=1}^{t} [a_i, b_i] \subset [a, b] \subset \mathbb{R},$$

where $a_i < b_i < a_{i+1} < b_{i+1}$ for all $i = 1, \cdots, t-1$.

When the multi-interval Remez algorithm is considered, which approximates a given continuous function on D with a polynomial having a degree less than or equal to d, there are two crucial points to be considered. One is to establish an efficient criterion for choosing new $d+2$ reference points among several extreme points. The other is to make efficient some steps in the improved multi-interval Remez algorithm. These two issues for the improved multi-interval Remez algorithm are dealt with in Subsections 3.1.1 and 3.1.3, respectively.

3.1.1 Improved Multi-Interval Remez Algorithm with Criteria for Choosing Extreme Points

It is assumed that the multi-interval Remez algorithm is applied on D and $\{g_1, \cdots, g_n\}$ satisfying Haar condition on $[a, b]$ is used as the basis of polynomials. After obtaining the minimax approximate polynomial regarding the set of reference points for each iteration, a new set of reference points is chosen for the next iteration. However, there are many boundary points in D, and all these boundary points have to be considered as extreme points of the error function. For this reason, there are many cases of selecting $n+1$ points among these extreme points. For bootstrapping in the CKKS scheme, there are many intervals to be considered, and thus there are lots of candidate extreme points. Since the criterion of the original multi-interval Remez algorithm cannot determine the unique new set of reference points for each iteration, it is necessary to make how to choose $n + 1$ points for each iteration to reduce the number of iterations as small as possible. Otherwise, it requires a large number of iteration for convergence to the minimax approximate polynomial. On the other hand, if the criterion is not designed properly, the improved multi-interval Remez algorithm may not converge into a single polynomial in some cases.

In order to set the criterion for selecting $n + 1$ reference points, it is needed to define a simple function for extreme points, $\mu_{p,f} : D \to \{-1, 0, 1\}$ as follows,

$$\mu_{p,f}(z) = \begin{cases} 1 & p(x) - f(x) \text{ is concave at } z \text{ on } D \\ -1 & p(x) - f(x) \text{ is convex at } z \text{ on } D \\ 0 & \text{otherwise,} \end{cases}$$

where $p(x)$ is a polynomial obtained in that iteration and $f(x)$ is a continuous function on D to be approximated. The notation $\mu_{p,f}$ is abused as μ.

Assume that the number of extreme points of $p(x) - f(x)$ on D is finite, and the set of extreme points is denoted by $B = \{w_1, w_2, \cdots, w_m\}$. Assume that B is ordered in increasing order, $w_1 < w_2 < \cdots < w_m$, and then the values of μ at these points are always 1 or -1. Let $\mathcal{S}$ be a set of functions defined as

$$\mathcal{S} = \{\sigma : [n+1] \to [m] \mid \sigma(i) < \sigma(i+1) \text{ for all } i = 1, \cdots, n\},$$

which means all the ways of choosing $n+1$ points of the m points. Clearly, $\mathcal{S}$ has only the identity function if $n + 1 = m$.

Then, three criteria are set for selecting $n + 1$ extreme points as follows:

i) *Local extreme value condition.* If E is the absolute value of error at points in the set of reference points, then it is obtained

$$\min_i \mu(x_{\sigma(i)})(p(x_{\sigma(i)}) - f(x_{\sigma(i)})) \geq E.$$

ii) *Alternating condition.* $\mu(x_{\sigma(i)}) \cdot \mu(x_{\sigma(i+1)}) = -1$ for $i = 1, \cdots, n$.

iii) *Maximum absolute sum condition*. Among σ's satisfying the above two conditions, choose σ maximizing the following value

$$\sum_{i=1}^{n+1} |p(x_{\sigma(i)}) - f(x_{\sigma(i)})|.$$

It is noted that the local extreme value condition in i) means in particular that the extreme points are discarded if the local maximum value of $p(x) - f(x)$ is negative or the local minimum of $p(x) - f(x)$ is positive.

Note that the first two conditions are also included in the original multi-interval Remez algorithm. The third condition, the maximum absolute sum condition, is the replacement of the condition that the new set of reference points includes the global extreme point. The numerical analysis will show that the third condition makes the proposed improved multi-interval Remez algorithm converge to the optimal minimax approximate polynomial fast. Although there are some cases in which the global maximum point is not included in the new set of reference points chosen by the maximum absolute sum condition, it is proved that the maximum absolute sum condition is enough for the improved multi-interval Remez algorithm to converge to the minimax approximate polynomial in the next subsection.

I propose the improved multi-interval Remez algorithm for the continuous function on the union of finitely many closed intervals as in Algorithm 8. The local extreme value condition is reflected in Step 3, and the alternating condition and the maximum absolute sum condition are reflected in Step 4.

3.1.2 Correctness of Improved Multi-Interval Remez Algorithm

It is required to be proved that the proposed algorithm always converges to the minimax approximate polynomial for a given piecewise continuous function on D. This proof is similar to the convergence proof of the original Remez algorithm on one closed interval [18, 62], but there are a few more general arguments than the original proof.

Algorithm 8 Improved Multi-Interval Remez algorithm

```
Input  : An input domain $D = \bigcup_{i=1}^{t}[a_i, b_i] \subset \mathbb{R}$, a continuous function $f$ on $D$, an
         approximation parameter $\delta$, and a basis $\{g_1, \cdots, g_n\}$
Output: The minimax approximate polynomial $p$ for $f$

Select $x_1, x_2, \cdots, x_{n+1} \in D$ in strictly increasing order.
Find the polynomial $p(x)$ with $p(x_i) - f(x_i) = (-1)^i E$ for some $E$.
Gather all extreme and boundary points such that $\mu_{p,f}(x)(p(x) - f(x)) \geq |E|$ into a
  set $B$.
Find $n + 1$ extreme points $y_1 < y_2 < \cdots < y_{n+1}$ with alternating condition and
  maximum absolute sum condition in $B$.
$\epsilon_{\mathsf{max}} \leftarrow \max_i |p(y_i) - f(y_i)|$
$\epsilon_{\mathsf{min}} \leftarrow \min_i |p(y_i) - f(y_i)|$
if $(\epsilon_{\mathsf{max}} - \epsilon_{\mathsf{min}})/\epsilon_{\mathsf{min}} < \delta$ then
    return $p(x)$
else
    Replace $x_i$'s with $y_i$'s and go to line 2.
end
```

This convergence proof includes the proof for both the original multi-interval Remez algorithm and the improved multi-interval Remez algorithm.

It is needed to be checked that $\mathcal{S}$ always contains at least one element σ_0 that satisfies the local extreme value condition and the alternating condition, and has $\sigma_0(i_0)$ for some i_0 such that $|p(x_{\sigma_0(i_0)}) - f(x_{\sigma_0(i_0)})| = \|p - f\|_\infty$. This existence is in fact the basic assumption of the original multi-interval Remez algorithm, but this existence is proved for mathematical clarification in the following theorems.

Theorem 3.1.1. *Let B and $\mathcal{S}$ be the sets defined above. Then, there is at least one element in $\mathcal{S}$ which satisfies the local extreme value condition and the alternating condition and has $\sigma_0(i_0)$ for some i_0 such that $|p(x_{\sigma_0(i_0)}) - f(x_{\sigma_0(i_0)})| = \|p - f\|_\infty$.*

Proof. Let a_i and b_i be the boundary points in D defined above and let $t_1, t_2, \cdots, t_{n+1} \in D$ be the reference points used to construct $p_k(x)$ at the k-th iteration. Without loss of generality, $t_i < t_{i+1}$ for all $i = 1, \cdots, n$, and the following equation for some proper positive value E is satisfied as

$$p(t_i) - f(t_i) = (-1)^{i-1}E.$$

Let u_{2i-1} be the largest point among all a_j and t_{2j}'s which are less than or equal to t_{2i-1}, and let v_{2i-1} be the smallest point among all b_j and t_{2j}'s which are larger than or equal to t_{2i-1}. Then, firstly, it is proved that there exists at least one local maximum point c_{2i-1} of $p_k(x) - f(x)$ in $[u_{2i-1}, v_{2i-1}]$, and $c_{2i-1} < t_{2i} < c_{2i+1}$ for all possible i. From the extreme value theorem for continuous function on interval [66], there exists at least one maximum point of $p_k(x) - f(x)$ in $[u_{2i-1}, v_{2i-1}]$, since $p_k(x) - f(x)$ is continuous on D. This value at the maximum point is denoted as c_{2i-1}. Since t_{2i-1} is in $[u_{2i-1}, v_{2i-1}]$, $p_k(c_{2i-1}) - f(c_{2i-1}) \geq E > -E = p_k(t_{2j}) - f(t_{2j})$ for all possible j, and thus c_{2i-1} cannot be equal to any t_{2i}'s. Since elements appeared more than once in intervals $[u_{2i-1}, v_{2i-1}]$, $i = 1, 2, \cdots, \lfloor \frac{n+2}{2} \rfloor$, are only t_{2i}'s and $v_{2i-1} \leq t_{2i} \leq u_{2i+1}$ for all possible i, it is proved that $c_{2i-1} < t_{2i}$ and $t_{2i} < c_{2i+1}$.

Let u_{2i} be the largest point among all a_j and c_{2j-1}'s which are less than or equal to t_{2i}, and let v_{2i} be the smallest point among all b_j and c_{2j-1}'s which are larger than or equal to t_{2i}. Then, it is proved that there exists at least one local minimum point c_{2i} of $p_k(x) - f(x)$ in $[u_{2i}, v_{2i}]$, and c_i's are sorted in strictly increasing order. Again, from the extreme value theorem for continuous function on interval, there exists at least one minimum point of $p_k(x) - f(x)$ in $[u_{2i}, v_{2i}]$. This value at the minimum point is denoted as c_{2i}. Since t_{2i} is in $[u_{2i}, v_{2i}]$, $p_k(c_{2i}) - f(t_{2j}) \leq -E < E \leq p_k(c_{2j-1}) - f(c_{2j-1})$ for all possible j, and thus c_{2i} cannot be equal to any c_{2j-1}. Since elements appeared more than once in intervals $[u_{2i}, v_{2i}]$, $i = 1, 2, \cdots, \lfloor \frac{n+2}{2} \rfloor$, are only c_{2i-1}'s and $v_{2i} \leq c_{2i+1} \leq u_{2i+2}$ for all possible i, Now, it is proved that c_i's are sorted in strictly increasing order.

Since c_i's are all local extreme points, $c_i \in B$ for all i. Then, $\sigma' \in \mathcal{S}$ is set such that $x_{\sigma'(i)} = c_i$. Since c_{2i-1}'s are local maximum points and c_{2i}'s are local minimum points, σ' satisfies alternating condition. Since $\mu(c_i)(p(c_i) - f(c_i)) \geq E$, σ' also satisfies the local extreme value condition. If one of c_i has the maximum absolute value of $p - f$, it is done.

Assume that all of c_i's do not have the maximum absolute value of $p_k - f$. Let x_m be the global extreme point of $p_k - f$. If there is some k such that $c_k < x_m < c_{k+1}$, either c_k or c_{k+1} has the same value of μ at x_m. Then, it is replaced with x_m and define this function as σ_0. Since σ_0 satisfies all of conditions in Theorem 3.1.1, it is done.

If $x_m < c_1$ or $x_m > c_{n+1}$, it is separated into two cases again. If $\mu(x_m) = \mu(c_1)$ (resp. $\mu(x_m) = \mu(c_{n+1})$), c_1 (resp. c_{n+1}) is just replaced with x_m and define this function as σ_0, and σ_0 satisfies all these conditions. If $\mu(x_m) \neq \mu(c_1)$ (resp. $\mu(x_m) \neq \mu(c_{n+1})$), c_{n+1} (resp. c_1) is replaced with x_m, and relabel the points to define the new function σ_0. This also satisfies all three conditions. Thus, it is proved.

$\square$

Remark. This theorem also ensures that $m \geq n + 1$. If $m < n + 1$, $\mathcal{S}$ has to be empty. This theorem ensures that there is at least one element in $\mathcal{S}$, it can be convinced that

$m \geq n+1$.

Before proving the convergence, it is required to generalize the de La Vallee Poussin theorem, which was used to prove the convergence of the original Remez algorithm on an interval. Since the original de La Vallee Poussin theorem [18] only deals with a single interval, it is generalized in the following theorem that deals with the closed subset of an interval, whose proof is almost the same as that of the original theorem.

Lemma 3.1.2 (Generalized de La Vallee Poussin Theorem). *Let $\{g_1, \cdots, g_n\}$ be a set of continuous functions on $[a,b]$ satisfying the Haar condition, and let D be a closed subset of $[a,b]$. Let f be a continuous on D, and p be a polynomial such that $p-f$ has alternately positive and negative values at $n+1$ consecutive points x_i in D. Let p^* be a minimax approximate polynomial for f on D, and $e(f)$ be the minimax approximation error of p^*. Then,*

$$e_D(f) \geq \min_i |p(x_i) - f(x_i)|.$$

Proof. Assume that the above statement is false. Then, there is a polynomial p_0 such that $p_0 - f$ has alternately positive and negative values at $n+1$ consecutive points in D, and

$$\|p^* - f\|_\infty < |p_0(x_i) - f(x_i)| \tag{3.1}$$

for all i. Then, $p_0 - p^* = (p_0 - f) - (p^* - f)$ has alternately positive and negative values at the consecutive x_i, which leads to the fact that there is n roots in $[a,b]$. From Lemma 2.5.3, $p_0 - p^*$ has to be zero, which is contradiction. □

Now, the convergence of Algorithm 8 is proved.

Theorem 3.1.3. *Let $\{g_1, \cdots, g_n\}$ be a set of functions satisfying the Haar condition on $[a,b]$, D be the multiple sub-intervals of $[a,b]$, and f be a continuous function on D. Let p_k be an approximate polynomial generated in the k-th iteration of the* *improved multi-interval Remez algorithm, and p^* be the optimal minimax approximate polynomial of f. Then, as k increases, p_k converges uniformly to p^* as in the following inequality*

$$\|p_k - p^*\|_\infty \le A\theta^k,$$

where A is a non-negative constant and $0 < \theta < 1$.

Proof. Let $\{x_1^{(0)}, \cdots, x_{n+1}^{(0)}\}$ be the initial set of reference points and $\{x_1^{(k)}, \cdots, x_{n+1}^{(k)}\}$ be the new set of reference points chosen at the end of iteration k. Let $r_k = p_k - f$ be the error function of p_k and $r^* = p^* - f$ be the error function of p^*. Since p_k is generated such that the absolute values of the error function r_k at the reference points $x_i^{(k-1)}$, $i = 1, 2, \cdots, n+1$ are the same. For $k \ge 1$,

$$\alpha_k = \min_i |r_k(x_i^{(k-1)})| = \max_i |r_k(x_i^{(k-1)})|,$$

$$\beta_k = \|r_k\|_\infty,$$

$$\gamma_k = \min_i |r_k(x_i^{(k)})|. \tag{3.2}$$

Define $\beta^* = \|r^*\|_\infty$. $\beta^* \ge \gamma_k$ is satisfied from Lemma 3.1.2, $\beta_k \ge \beta^*$ by definition of p^*, and $\gamma_k \ge \alpha_k$ by the local extreme value condition for new set of reference points. Then,

$$\alpha_k \le \gamma_k \le \beta^* \le \beta_k.$$

Let $c^{(k)} = [c_1^{(k)}, \cdots, c_n^{(k)}]^T$ be the coefficient vector of p_k. Then, $c^{(k)}$ is the solution vector of the following system of linear equations

$$(-1)^{i+1} h^{(k)} + \sum_{j=1}^{n} c_j^{(k)} g_j(x_i^{(k-1)}) = f(x_i^{(k-1)}), \quad i = 1, \cdots, n+1 \tag{3.3}$$

for the $n+1$ unknowns $h^{(k)}$ and $c_j^{(k)}$'s, and $|h^{(k)}| = \alpha_k$. From Theorem 2.5.5, it is assured that the system of linear equations in (3.3) is nonsingular, which can be rewritten as in the matrix equation for $k+1$, instead of k,

$$\begin{bmatrix} 1 & g_1(x_1^{(k)}) & \cdots & g_n(x_1^{(k)}) \\ -1 & g_1(x_2^{(k)}) & \cdots & g_n(x_2^{(k)}) \\ \vdots & \vdots & \ddots & \vdots \\ (-1)^n & g_1(x_{n+1}^{(k)}) & \cdots & g_n(x_{n+1}^{(k)}) \end{bmatrix} \begin{bmatrix} h^{(k+1)} \\ c_1^{(k+1)} \\ \vdots \\ c_n^{(k+1)} \end{bmatrix} = \begin{bmatrix} f(x_1^{(k)}) \\ f(x_2^{(k)}) \\ \vdots \\ f(x_{n+1}^{(k)}) \end{bmatrix}. \tag{3.4}$$

From Cramer's rule, $h^{(k+1)}$ is as follows,

$$h^{(k+1)} = \begin{vmatrix} f(x_1^{(k)}) & g_1(x_1^{(k)}) & \cdots & g_n(x_1^{(k)}) \\ f(x_2^{(k)}) & g_1(x_2^{(k)}) & \cdots & g_n(x_2^{(k)}) \\ \vdots & \vdots & \ddots & \vdots \\ f(x_{n+1}^{(k)}) & g_1(x_{n+1}^{(k)}) & \cdots & g_n(x_{n+1}^{(k)}) \end{vmatrix} \Bigg/ \begin{vmatrix} 1 & g_1(x_1^{(k)}) & \cdots & g_n(x_1^{(k)}) \\ -1 & g_1(x_2^{(k)}) & \cdots & g_n(x_2^{(k)}) \\ \vdots & \vdots & \ddots & \vdots \\ (-1)^n & g_1(x_{n+1}^{(k)}) & \cdots & g_n(x_{n+1}^{(k)}) \end{vmatrix}. \tag{3.5}$$

Let $M_i^{(k)}$ be the minor of the matrix in (3.4) removing the first column and the i-th row. Then, (3.5) can be written as

$$h^{(k+1)} = \frac{\sum_{i=1}^{n+1} f(x_i^{(k)}) M_i^{(k)} (-1)^{i+1}}{\sum_{j=1}^{n+1} M_j^{(k)}}. \tag{3.6}$$

If f is replaced by any polynomial $p = \sum_{j=1}^{n} c'_j g_j$ in (3.4), the minimax approximation on $\{x_1^{(k)}, \cdots, x_{n+1}^{(k)}\}$ is p itself. This leads to

$$\frac{\sum_{i=1}^{n+1} p_k(x_i^{(k)}) M_i^{(k)} (-1)^{i+1}}{\sum_{j=1}^{n+1} M_j^{(k)}} = 0. \tag{3.7}$$

By substracting (3.6) from (3.7), and $r_k = p_k - f$,

$$-h^{(k+1)} = \frac{\sum_{i=1}^{n+1} r_k(x_i^{(k)}) M_i^{(k)} (-1)^{i+1}}{\sum_{j=1}^{n+1} M_j^{(k)}}.$$

By the fact that $r_k(x_i^{(k)})$'s alternate in sign by the alternating condition for new set of reference points, and all minors M_i have the same sign by Lemma 2.5.8,

$$\left| \sum_{i=1}^{n+1} r_k(x_i^{(k)}) (-1)^i M_i^{(k)} \right| = \sum_{i=1}^{n+1} |r_k(x_i^{(k)})| |M_i^{(k)}|$$

$$\alpha_{k+1} = |h^{(k+1)}| = \frac{\sum_{i=1}^{n+1} |M_i^{(k)}||r_k(x_i^{(k)})|}{\sum_{j=1}^{n+1} |M_j^{(k)}|}. \tag{3.8}$$

Now let

$$\theta_i^{(k)} = \frac{|M_i^{(k)}|}{\sum_{j=1}^{n+1} |M_j^{(k)}|}.$$

Since α_{k+1} is weighted average of $|r_k(x_i^{(k)})|$ by $\theta_i^{(k)}$'s as weights, $\alpha_{k+1} \geq \gamma_k$ is satisfied from (3.2). Note that $\alpha_k \leq \gamma_k \leq \alpha_{k+1}$, and thus α_k is a non-decreasing sequence. This fact is used in the last part of the proof.

Note that there are some $n+1$ alternating points, where the approximate error values alternate, including the global extreme point by Theorem 3.1.1, and the approximate error values at $x_1^{(k)}, \cdots, x_{n+1}^{(k)}$ have the maximum absolute sum by the maximum absolute sum condition for new set of reference points. That is, $\sum_{i=1}^{n+1} |r_k(x_i^{(k)})|$ is larger than or equal to sum of any $n+1$ absolute error values including β_k and thus,

$$\sum_{i=1}^{n+1} |r_k(x_i^{(k)})| \geq \beta_k. \tag{3.9}$$

First, it will be proved that $\theta_i^{(k)}$ is larger than a constant $1 - \theta > 0$ throughout the iterations. It is known that $M_i \neq 0$ for all i from the Haar condition. The following inequality is firstly proved,

$$x_{i+1}^{(k)} - x_i^{(k)} \geq \epsilon > 0, \quad i = 0, \cdots, n, \tag{3.10}$$

where ϵ does not depend on k. Assume that (3.10) is not true. Let $x^{(k)} = (x_1^{(k)}, \cdots, x_{n+1}^{(k)})$ be a sequence defined on D^{n+1}. Then $x^{(k)}$ has its subsequence such that $\min_i |x_{i+1}^{(k)} - x_i^{(k)}|$ converges to zero. Since D^{n+1} is a closed and bounded subset of $\mathbb{R}^{n+1}$, it is a compact set, and thus this subsequence also has its subsequence converging to a point $(x_1^*, \cdots, x_{n+1}^*)$. Since $\min_i |x_{i+1}^* - x_i^*| = 0$, there is some i such that $x_i^* = x_{i+1}^*$. Let p be the minimax polynomial of f on $(x_1^*, \cdots, x_{n+1}^*)$. Since there is actually less than or equal to n points, p is the approximate polynomial generated by the Lagrange interpolation, and thus

$$p(x_i^*) = f(x_i^*), \quad i = 1, \cdots, n+1.$$

It is known that $\alpha_1 > 0$ by the fact that α_k is weighted average of absolute approximation errors at the previous set of reference points. Then, there exists a number $\delta > 0$ such that whenever $|y_1 - y_2| < \epsilon$ and $y_1, y_2 \in D$,

$$|(p-f)(y_1) - (p-f)(y_2)| < \alpha_1$$

because $p-f$ is a continuous function on the compact set, and thus it is also uniformly continuous. Since there is a subsequence of $(x_1^{(k)}, \cdots, x_{n+1}^{(k)})$ converging to $x^{(k)} = (x_1^*, \cdots, x_{n+1}^*)$, there is some k_0 such that

$$|x_i^{(k_0)} - x_i^*| < \delta, \quad i = 1, \cdots, n+1.$$

Then,

$$|(p-f)(x_i^{(k_0)}) - (p-f)(x_i^*)| = |p(x_i^{(k_0)}) - f(x_i^{(k_0)})| < \alpha_1$$

because $(p-f)(x_i^*) = 0$. In fact, p is not the minimax approximate polynomial in regard to the k-th set of reference points $\{x_1^{(k_0)}, \cdots, x_{n+1}^{(k_0)}\}$. Since α_{k+1} is the error value of the minimax approximate polynomial on $\{x_1^{(k_0)}, \cdots, x_{n+1}^{(k_0)}\}$,

$$\alpha_{k+1} \le \max_i |p(x_i^{(k_0)}) - f(x_i^{(k_0)})| < \alpha_1.$$

This contradicts the fact that α_k is non-decreasing sequence, and thus (3.10) is satisfied.

Now, it will be proved that $\theta_i^{(k)}$ is larger than a constant $1-\theta$. Consider the subset D' of D^{n+1} such that for $(y_1, \cdots, y_{n+1}) \in D'$, $y_{i+1} - y_i \ge \epsilon$. it can be easily seen that D' is a closed and bounded subset in $\mathbb{R}^{n+1}$, and thus D' is a compact set. Then, $|M_i|$, which is the same function as $|M_i^{(k)}|$ except that the inputs are y_i's instead of $x_i^{(k)}$'s, is a continuous function on D^{n+1}, so is on D', and thus there is an element at which $|M_i|$ has the mininum value on D' from the extreme value theorem. From the Haar condition in Definition 2.5.1, $|M_i|$ cannot be zero because y_i's are distinct and the minimum value of $|M_i|$ is not zero. Since the finite number of functions $|M_i|$'s are considered, the lower bound of all $|M_i|$'s is bigger than zero. In addition, since $\sum_{j=1}^{n+1} |M_j|$ is also a continuous function on D', there is an upper bound of $\sum_{j=1}^{n+1} |M_j|$ on D' from the extreme value theorem. This leads to the fact that θ_i's are lower-bounded beyond zero.

From $\gamma_{k+1} \geq \alpha_{k+1}$, (3.8), and (3.9),

$$\begin{aligned}
\gamma_{k+1} - \gamma_k &\geq \alpha_{k+1} - \gamma_k \\
&= \sum_{i=1}^{n+1} \theta_i^{(k)} (|r_k(x_i^{(k)})| - \gamma_k) \\
&\geq (1-\theta)(\beta_k - \gamma_k) \qquad (3.11) \\
&\geq (1-\theta)(\beta^* - \gamma_k). \qquad (3.12)
\end{aligned}$$

From (3.12),

$$\begin{aligned}
\beta^* - \gamma_{k+1} &= (\beta^* - \gamma_k) - (\gamma_{k+1} - \gamma_k) \\
&\leq (\beta^* - \gamma_k) - (1-\theta)(\beta^* - \gamma_k) \\
&= \theta(\beta^* - \gamma_k).
\end{aligned}$$

Then, the following inequality is obtained for some nonnegative B as

$$\beta^* - \gamma_k \leq B\theta^k. \qquad (3.13)$$

From (3.11) and (3.13),

$$\begin{aligned}
\beta_k - \beta^* &\leq \beta_k - \gamma_k \\
&\leq \frac{1}{1-\theta}(\gamma_{k+1} - \gamma_k) \\
&\leq \frac{1}{1-\theta}(\beta^* - \gamma_k) \\
&\leq \frac{1}{1-\theta} B\theta^k \\
&\leq C\theta^k. \qquad (3.14)
\end{aligned}$$

From Theorem 2.5.9, there is a constant $\gamma > 0$ such that

$$\|p^* - f\|_\infty + \gamma \|p_k - p^*\|_\infty \leq \|p_k - f\|_\infty.$$

Since $\beta_k = \|p_k - f\|_\infty$, $\beta^* = \|p^* - f\|_\infty$, and (3.14), the proof is completed by the following inequality

$$\begin{aligned}\|p_k - p^*\|_\infty &= \frac{\beta_k - \beta^*}{\gamma} \\ &\leq A\theta^k\end{aligned}$$

for nonnegative constant A.

□

Remark. The maximum sum condition is used in the inequality (3.11). Note that the inequality (3.11) can be satisfied if the global extreme point is included to the new set of reference points as in the original multi-interval Remez algorithm, instead of the maximum absolute sum condition in the improved multi-interval Remez algorithm. Thus, this proof naturally includes the convergence proof of the original multi-interval Remez algorithm.

From the proof, it is known that the convergence rate of α_k determines the convergence rate of the algorithm. Since α_k is always lower than β^* and non-decreasing sequence, it is desirable to obtain α_k as large as possible for each iteration. The maximum sum condition is more effective than the global extreme point inclusion condition; The global extreme point inclusion condition cannot care about the reference points other than the global extreme point, but the maximum sum condition cares for all the reference points to be large. This can give some intuition for the effectiveness of the maximum sum condition.

3.1.3 Efficient Implementation of Improved Multi-Interval Remez Algorithm

In this subsection, it is required to consider the issues in each step of Algorithm 8 and suggest how to implement Steps 1, 2, 3, and 4 of Algorithm 8 as follows.

Initialization:

Depending on the initialization method, there can be a large difference in the number of iterations required. Therefore, the closer the polynomial produced by initializing the initial reference points to the optimal minimax approximation polynomial, the fewer iterations are required. The node setting method of Han et al. [44] is used to effectively set the initial reference points in the improved multi-interval Remez algorithm. Since Han et al.'s node setting method was for polynomial interpolation, it chooses the $d+1$ number of nodes when the approximate polynomial of degree d is needed. Instead, if it is required to obtain the optimal minimax approximate polynomial of degree d, the $d+2$ number of nodes are chosen with Han et al.'s method as if the approximate polynomial of degree $d+1$ is needed, and uses them for the initial reference points.

Finding Approximate Polynomial:

A naive approach is finding coefficients of the approximate polynomial with power basis at the current reference points for the continuous function $f(x)$, i.e., c_j's can be obtained in the following equation

$$\sum_{j=0}^{d} c_j x_i^j - f(x_i) = (-1)^i E,$$

where E is also an unknown variable in this system of linear equations. However, this method suffers from the precision problem for the coefficients. It is known that as the degree of the basis of approximate polynomial increases, the coefficients usually decrease, and it is required to set higher precision for the coefficients of the higher degree basis. Han et al. [44] use the Chebyshev basis for this coefficient precision problem since the coefficients of a polynomial with the Chebyshev basis usually have the almost same order. Thus, the Chebyshev basis is used instead of the power basis.

Obtaining Extreme Points:

Since a tiny minimax approximation error is dealt with, it is required to obtain the extreme points as precisely as possible. Otherwise, the extreme point for the minimax approximate polynomial cannot be reached precisely, and then the minimax approximation error obtained with this algorithm becomes large. Basically, to obtain the extreme points, $p(x)-f(x)$ can be scanned with a small scan step and obtain the extreme points where the increase and decrease are exchanged. A small scan step increases the accuracy of the extreme point but causes a long scan time accordingly. To be more specific, it takes approximately 2^{ℓ} proportional time to find the extreme points with the accuracy of ℓ-bit. Therefore, it is necessary to devise a method to obtain high accuracy extreme points more quickly.

In order to obtain the exact point of the extreme value, a method of finding the points is used where the increase and decrease are exchanged and then finding the exact extreme point using a kind of binary search. Let $r(x) = p(x) - f(x)$ and sc be the scan step. If $x_{i,0}$ can be found where $\mu(x_{i,0})r(x_{i,0}) \geq |E|$, and $(r(x_{i,0}) - r(x_{i,0} - \mathsf{sc}))(r(x_{i,0} + \mathsf{sc}) - r(x_{i,0})) \leq 0$, the i-th extreme points are obtained using the following process successively ℓ times,

$$x_{i,k} = \underset{x \in \{x_{i,k-1} - \mathsf{sc}/2^k, x_{i,k-1}, x_{i,k-1} + \mathsf{sc}/2^k\}}{\arg\max} |r(x)|, k = 1, 2, \cdots, \ell,$$

where the i-th extreme point x_i is set to be $x_{i,\ell}$. Then, the extreme point is obtained with $O(\log(\mathsf{sc}) + \ell)$-bit precision. Since sc needs not to be a too small value, the extreme point can be found with arbitrary precision with linear time to precision ℓ. In summary, it is proposed that the ℓ-bit precision of the extreme points can be obtained by the linear time of ℓ instead of 2^{ℓ}.

This procedure for each interval in the approximation region can be performed independently with each other, and thus it can be performed effectively with several threads. Since this step is the slowest step among any other steps in the improved multi-interval Remez algorithm, the parallel processing for this procedure is desirable to make the whole algorithm much fast.

One can say that the Newton method is more efficient than the binary search method in finding the extreme points because one may just find the roots of the derivative of $p(x) - f(x)$. However, the extreme points are very densely distributed in this situation, and thus the Newton method may not be stably performed. Even if only one extreme point is missed, the algorithm can act in an undefined manner. The binary search method is fast enough and finds all of the extreme points very robustly, and thus the binary search is used instead of the Newton method.

Obtaining New Reference Points:

When the new reference points satisfying the local extreme value condition are found, the alternating condition, and maximum absolute sum condition, there is a naive approach: among local extreme points which satisfy the local extreme value condition, find all $d + 2$ points satisfying the alternating condition and choose the $n + 1$ points which have the maximum absolute sum value. If there are m local extreme points, it is required to investigate $\binom{m}{d+2}$ points, and this value is too large, making this algorithm impractical. Thus, it is required to find a more efficient method than this naive approach.

I propose a very efficient and provable algorithm for finding the new reference points. The proposed algorithm always gives the $d+2$ points satisfying the three criteria. It can be considered as an elimination method in that some elements are eliminated for each iteration in the proposed algorithm until $n + 1$ points are obtained. It is clear that as long as $m > d + 2$, at least one element which may not be included in the new reference points can be found. This proposed algorithm is given in Algorithm 9. Algorithm 9 takes $O(m \log m)$ running time, which is a quasi-linear time algorithm.

It is noted that there are always some points in all situations such that it can be ensured that if a set of $d + 2$ points is chosen including these points satisfying the alternating condition, there exists the other set of $d + 2$ points without these points which satisfies the alternating condition and whose absolute sum is larger. Algorithm 3 finds these points until the number of the remaining points is $d + 2$.

To understand the last part of Algorithm 9, the example can be given that if the extreme point x_2 is removed, $T = \{|r(x_1)| + |r(x_2)|, |r(x_2)| + |r(x_3)|, |r(x_3)| + |r(x_4)|, \cdots\}$ is changed to $T = \{|r(x_1)| + |r(x_3)|, |r(x_3)| + |r(x_4)|, \cdots\}$. It is assumed that whenever an element is removed in the ordered set B in Algorithm 9, the remaining points remain sorted and indices are relabeled in increasing order. When the values to remove some extreme points are compared, there are the cases that the compared values are equal or the smallest element is more than one. In such cases, one of these elements is randomly removed. The correctness of Algorithm 9 is proven in the following theorem.

Theorem 3.1.4. *Algorithm 9 always returns the $n+1$ points satisfying the alternating condition and the maximum absolute sum condition.*

Proof. Let $B_{\mathsf{init}} = \{t'_1, t'_2, \cdots, t'_m\}$ be the initial elements in B, and let t'_ℓ be an element removed in the first while statement. It is first shown that each element removed in the first while statement in Algorithm 9 is not included in the new reference; that is, if the subset A of B_{init} having $n + 1$ elements satisfies alternating condition and contain this removed element, there is another subset A' of B having $n + 1$ elements satisfying alternating condition, not containing the removed element, and having absolute sum larger than or equal to A. Since it is removed in the first while statement, there is an element $t'_{\ell'}$ such that $|r(t'_{\ell'})| \geq |r(t'_\ell)|$, $\mu(t'_{\ell'}) = \mu(t'_\ell)$, and $|\ell' - \ell| = 1$. Clearly, A cannot contain $t'_{\ell'}$, because A satisfies the alternating condition. Let A' be the same set as A except that t'_ℓ is replaced with $t'_{\ell'}$. Then A' also satisfies alternating condition, does not contain t'_ℓ, and has absolute sum larger than or equal to that of A.

It is observed that at the end of the first while statement, B itself satisfies the alternating condition. Then it is required to prove that elements removed in the second while statement in Algorithm 9 are not included in the new reference points. In other words, if the subset A of B_{init} having $n+1$ elements satisfies the alternating condition

Algorithm 9 New Reference

Input : An increasing ordered set of extreme points $B = \{t_1, t_2, \cdots, t_m\}$ with $m \geq d+2$, and the degree of the approximate polynomial d.

Output: $d+2$ points in B satisfying alternating condition and maximum absolute sum condition.

```
$i \leftarrow 1$
while $t_i$ is not the last element of $B$ do
    if $\mu(t_i)\mu(t_{i+1}) = 1$ then Remove from $B$ one of two points $t_i, t_{i+1}$ having the
      smaller value among $\{|r(t_i)|, |r(t_{i+1})|\}$.
    else $i \leftarrow i + 1$
end
if $|B| > d + 3$ then Calculate all $|r(t_i)| + |r(t_{i+1})|$ for $i = 1, \cdots, |B| - 1$ and sort
  and store these values into the array $T$.
while $|B| > d + 2$ do
    if $|B| = d + 3$ then Remove from $B$ one of two points $t_1, t_{|B|}$ having less value
      among $\{|r(t_1)|, |r(t_{|B|})|\}$.
    else if $|B| = d + 4$ then Insert $|r(t_1)| + |r(t_{|B|})|$ into $T$ and sort $T$. Remove from
      $B$ the two elements having the smallest value in $T$.
    else
        if $t_1$ or $t_{|B|}$ is included in the smallest element in $T$ then Remove from $B$ only
          $t_1$ or $t_{|B|}$.
        else Remove from $B$ the two elements having the smallest element in $T$.
        Remove from $T$ all elements related to the removed extreme points, and insert
          into $T$ the sum of absolute error values of the two newly adjacent extreme
          points.
    end
end
```

and contains these removed elements, there is another subset A' of B having $n+1$ elements satisfying alternating condition, not containing these removed elements, and having absolute sum larger than or equal to A. Let t'_ℓ be an element removed in the second while statement.

Then, there are three cases: at the time of removal of t'_ℓ, the remaining set B can have $n+2$, $n+3$, or larger than $n+3$ elements as in Algorithm 9. Each case is considered separately. By the induction argument, it is assumed that the remaining set B in each iteration has $n+1$ points that satisfy the alternating condition and have the maximum absolute sum among all possible $n+1$ points in B_{init}. In other words, it can be assumed that if there are $n+1$ points in B_{init} that satisfy the alternating condition and contain at least one of the removed elements before that time, there are $n+1$ points in the remaining set B at that time such that they satisfy the alternating condition and have absolute sum larger than or equal to the previous ones. This inductive assumption makes me consider only the remaining set B at that iteration instead of all B_{init} in the proof.

i) Case of $n+2$: If the remaining set B has $n+2$ elements at the time of removal of t'_ℓ, elements in B at that time are labeled as $t_1, t_2, \cdots, t_{n+2}$, and t'_ℓ is labeled as t_1 or t_{n+2}. Then, $|r(t)|$ at one of the two points t_1 and t_{n+2}, which is not t'_ℓ has the value of $|r(x)|$ larger than or equal to the value at t'_ℓ. This element is denoted as $t'_{\ell'}$. If there is a subset A of $n+1$ points in the remaining set B that satisfy alternating condition and contain t'_ℓ, the element $t'_{\ell'}$ must not be in these $n+1$ points due to alternating condition. Let A' be the same set as A except that t'_ℓ is replaced with $t'_{\ell'}$. Then, A' also satisfies the alternating condition, does not contain t'_ℓ, and has absolute sum larger than or equal to that of A.

ii) Case of $n+3$: If the remaining set B has $n+3$ elements at the time of removal of t'_ℓ, the elements in B at that time are labeled as $t_1, t_2, \cdots, t_{n+3}$, and it is required to remove two elements. Then, there must be a different element t'_p

which is also removed at the time of removal of t'_ℓ. $\{t'_\ell, t'_p\}$ can be $\{t_i, t_{i+1}\}$ for some i or $\{t_1, t_{n+3}\}$ as in Algorithm 9. Since all of the subsets of B having $n+1$ elements that satisfy the alternating condition are the cases of $B\backslash\{t_i, t_{i+1}\}$ for some i or $B\backslash\{t_1, t_{n+3}\}$, one subset of B with the alternating condition that has the maximum absolute sum has to be $B\backslash t'_\ell, t'_p$. Therefore, the resulting subset can be obtained by removing these two elements.

iii) Case of larger than $n+3$: If the remaining set B has elements larger than $n+3$ elements at the time of removal of t'_ℓ, the elements in B at that time are labeled as $t_1, t_2, \cdots, t_j$, where $j > n+3$. Then, there are two cases: One is that t'_ℓ is labeled as t_1 or t_j, and the other is not the first case.

   (a) If t'_ℓ is labeled as t_1 or t_j, let t'_p be the adjacent element in B. If the subset A in B that satisfies the alternating condition and contains t'_ℓ also contains t'_p, there is at least one pair of adjacent elements $t'_{\ell'}$ and $t'_{p'}$ in B that is not contained in A, since A satisfies the alternating condition and more than three elements are removed from B. Note that $|r(t'_{\ell'})| + |r(t'_{p'})| \geq |r(t'_\ell)| + |r(t'_p)|$. Let A' be the same set as A except that t'_ℓ and t'_p are replaced with $t'_{\ell'}$ and $t'_{p'}$. A' also satisfies alternating condition, does not contain t'_ℓ, and has absolute sum larger than or equal to that of A.

   If A does not contain t'_p, the adjacent element of t'_p which is not t'_ℓ cannot be contained in A, since A satisfies the alternating condition. Let $t'_{\ell'}$ be the adjacent element of t'_p. Note that $|r(t'_{\ell'})| + |r(t'_p)| \geq |r(t'_\ell)| + |r(t'_p)|$. Let A' be the same set with A except that t'_ℓ is replaced with $t'_{\ell'}$. A' also satisfies the alternating condition, does not contain t'_ℓ, and has absolute sum larger than or equal to that of A.

   (b) If t'_ℓ is not labeled as t_1 or t_j, the adjacent element t'_p of t'_ℓ in B where $|r(t'_\ell)| + |r(t'_p)|$ is the smallest value in T cannot be t_1 or t_j. If this is the case, t'_ℓ cannot be removed but t'_p is removed in that iteration. If the alternated subset A in B that contains t'_ℓ also contain t'_p, there is at least one pair of adjacent elements $t'_{\ell'}$ and $t'_{p'}$ in B that is not contained in A. Note also that $|r(t'_{\ell'})| + |r(t'_{p'})| \geq |r(t'_\ell)| + |r(t'_p)|$. Let A' be the same set as A except that t'_ℓ and t'_p are replaced with $t'_{\ell'}$ and $t'_{p'}$. A' also satisfies the alternating condition, does not contain t'_ℓ, and has absolute sum larger than or equal to that of A.

If A does not contain t'_p, there is the adjacent element of t'_p which is not t'_ℓ, since t'_p is not t_1 or t_j. Let $t'_{\ell'}$ be adjacent element of t'_p. Then, $t'_{\ell'}$ cannot be contained in A, since A satisfies the alternating condition. Note that $|r(t'_{\ell'})| + |r(t'_p)| \geq |r(t'_\ell)| + |r(t'_p)|$. Let A' be the same set as A except that t'_ℓ is replaced with $t'_{\ell'}$. A' also satisfies alternating condition, does not contain t'_ℓ, and has absolute sum larger than or equal to that of A.

Thus, the theorem is proved. $\square$

3.1.4 Numerical Analysis with Improved Multi-Interval Remez Algorithm

This subsection shows the numerical analysis of the improved multi-interval Remez algorithm for its efficiency and the optimal minimax approximation error.

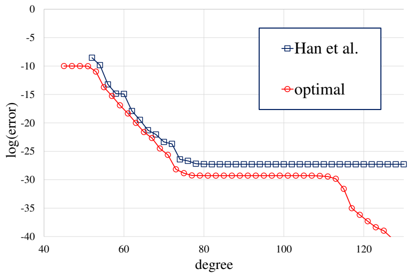
Maximum Sum Condition:

Table 3.1 shows the number of iterations required to converge to the optimal minimax approximate polynomial in the multi-interval Remez algorithm and the improved multi-interval Remez algorithm. The initial set of reference points is selected uniformly in each interval since it is desirable to observe their performances in the worst case. While selecting new reference points is not unique for each iteration in the multi-interval Remez algorithm, the improved multi-interval Remez algorithm selects the new reference points uniquely for each iteration. Thus, when the multi-interval Remez algorithm is analyzed, the new reference points are randomly selected for each iteration among the possible sets of reference points that satisfy the local extreme value condition and the alternating condition and have the global extreme point. The approximation parameter δ is set in Algorithm 8 as 2^{-40} and repeat this simulation 100 times. It shows that the improved multi-interval Remez algorithm is much better to reduce the iteration number of the Remez algorithm.

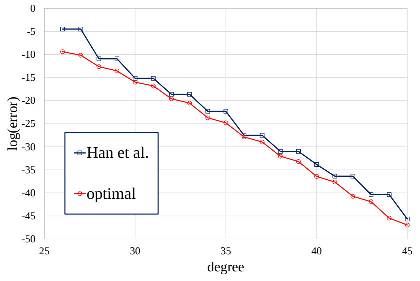
Note that the number of iterations depends on the initial set of reference points. In fact, the uniformly distributed reference points are not desirable as an initial set of reference points because these reference points are far from the converged reference points. In fact, the improved multi-interval Remez algorithm with the initialization method explained in the previous subsection only needs 4$\sim$14 iterations. The overall running time of the improved multi-interval Remez algorithm with the method in the previous subsection is 1$\sim$3 seconds by PC with AMD Ryzen Threadripper 1950X 16-core CPU @ 3.40GHz.

Table 3.1: Comparison of iteration numbers between the improved multi-interval Remez algorithm and the multi-interval Remez algorithm for $\delta = 2^{-40}$

degree of approx. poly.	modified Remez algorithm	multi-interval Remez algorithm			
		average	standard deviation	max	min
79	**28**	60.0	9.38	82	41
99	**8**	17.1	3.34	28	11
119	**26**	53.4	8.10	79	37
139	**39**	60.3	4.71	79	48
159	**39**	72.1	9.71	98	42
179	**48**	72.3	9.72	105	53
199	**56**	80.4	7.28	94	60



(a) Modular reduction function



(b) Cosine function with scaling number two

Figure 3.1: Comparison of minimax approximatio error between the previous approximation method and the improved multi-interval Remez algorithm.

Minimax Error:

The optimal minimax approximate polynomials for the modular reduction function and the scaled cosine function with the scaling number two are obtained. Fig. 6.2(a) shows the minimax approximation error of the approximate polynomial of the modular reduction function derived by the improved multi-interval Remez algorithm and the minimax approximation error of the previous homomorphic modular reduction method with scaling number zero in [44], compared to the modular reduction function. That is, let $p_1(x)$ be the optimal minimax approximate polynomial of the normod function and let $q_1(x)$ be the approximate polynomial obtained by Han et al.'s method with scaling number zero when the half-width of approximation region is 2^{-10}. Then, $\max_{x\in D}|p_1(x) - \mathsf{normod}(x)|$ and $\max_{x\in D}|q_1(x) - \mathsf{normod}(x)|$ are compared in Fig. 6.2(a). Note that while the minimax approximation error of the approximate polynomial of the modular reduction function decreases steadily as the degree of the approximate polynomial increases, the minimax approximation error of the previous method does not decrease when the degree is larger than 76 because of the approximation error between the modular reduction function and the sine/cosine function.

Fig. 6.2(b) shows the minimax approximation error of the composition of the approximate polynomial of the scaled cosine function with scaling number two derived by the improved multi-interval Remez algorithm and two double angle formulas and the minimax approximation error of method in [44], compared to the cosine function. That is, let $p_2(x)$ be the optimal minimax approximate polynomial of $\cos\left(\frac{\pi}{2}(x-1/4)\right)$ and let $q_2(x)$ be the approximate polynomial obtained by Han et al.'s method with scaling number two when the half-width of approximation region is 2^{-3}. If $r(x) = 2x^2 - 1$, then $\max_{x\in D} |r \circ r \circ p_2(x) - \sin(2\pi x)|$ and $\max_{x\in D} |r \circ r \circ q_2(x) - \sin(2\pi x)|$ are compared in Fig. 6.2(b). The proposed method improves the minimax approximation error by 2.3 bits on average, and by 5 bits at maximum for the same degree of the approximate polynomial. This improvement leads to a reduction of 1$\sim$2 degrees for the given minimax approximation error.

In fact, the approximate polynomial for the modular reduction function cannot yet be used in the bootstrapping of the RNS-CKKS scheme because of the huge coefficients. It is a very unstable polynomial to evaluate in the RNS-CKKS scheme in that these large coefficients amplify the approximation error in the message. It is an interesting open problem to stably use the minimax approximate polynomial of the modular reduction function in the RNS-CKKS scheme. Instead of using the unstable minimax approximate polynomial of the modular reduction function, the modular reduction function is approximated with a composition of several stable polynomials in Section 3.2.

3.2 Composite Function Approximation for CKKS Bootstrapping

3.2.1 Numerical Analysis of Message Precision in Bootstrapping with Improved Multi-Interval Remez Algorithm in `SEAL` Library

Since the previous researches for the bootstrapping of the RNS-CKKS scheme did not deal deeply with its message precision, the message precision for the bootstrapping is numerically analyzed with improved multi-interval Remez algorithm of the RNS-CKKS scheme by changing several parameters: the degree d of the approximate polynomial of the scaled cosine function, the bit-length difference $\delta_{\mathsf{diff}} = \log \Delta_{\mathsf{boot}} - \log \Delta$ between the default scaling factor and the bootstrapping scaling factor, the bootstrapping scaling factor Δ_{boot}, and the number of the slots. it is assumed that the range of the real part and the imaginary part of the messages to be bootstrapped is $[-1, 1]$. The bootstrapping precision is measured as $-\log_2(e_r + e_i)/2$, where e_r and e_i are the average error of the real part and the imaginary part of all slots, respectively.

Numerical analysis in this subsection is conducted in PC with Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz single-threaded, and the SEAL library version 3.5.9 [67] is used. The double angle formula for the cosine function is assumed to be used twice. The improved multi-interval Remez algorithm is used to obtain the optimal minimax approximate polynomial in all simulations, rather than polynomial approximation methods in the previous papers, [16, 20, 44]. The polynomial modulus degree is set to be 2^{16}, the secret key Hamming weight is set to be 192, the value of K is set to be 25, and the maximum modulus for the ciphertext is set to be 2^{1553}, which satisfies the 128-bit security as in [8]. The CoeffToSlot and SlotToCoeff procedures in [16] with two level consumption are used in all simulations. The scaling factor management method and the delayed rescaling method in [50] is applied, and the depth consumption of the polynomial evaluation is optimized by Bossuat et al.'s evaluation method [8]. The input messages are sampled by the uniform distribution over the bootstrapping range.

Degree of Approximate Polynomial:

Table 3.2 shows the message precision of the bootstrapping with the improved multi-interval Remez algorithm when the degree of the approximate polynomial for the scaled cosine function is changed. The value of $\log \delta_{\mathsf{diff}}$ is 12, $\log \Delta_{\mathsf{boot}}$ is 60, and the number of the slots is 2^8 in this simulation. The approximation error means the minimax error of the approximate polynomial for the scaled cosine function, and the bootstrapping error means the average error for each slot when the bootstrapping is performed with the library. When the scaling factor is changed from the bootstrapping scaling factor to the default scaling factor, the message and its error are multiplied by δ_{diff}. Both the bootstrapping error before changing the scaling factor and that after changing the scaling factor are shown. Although the approximation error continues to decrease as the degree of the approximate polynomial increases, the bootstrapping error does not decrease below a certain value. This bound is caused by either the difference between the modular reduction function and the cosine function or the rescaling error, depending on the situation. Thus, the message precision cannot be raised infinitely by using a high degree approximate polynomial. The actual lower bound of the bootstrapping error before changing the scaling factor is denoted by e_{min} in this subsection, and then the lower bound of the bootstrapping error after changing the scaling factor is $e_{\mathsf{min}}\delta_{\mathsf{diff}}$.

Table 3.2: Message precision of the bootstrapping with the improved multi-interval Remez algorithm for various degrees of the approximate polynomials

degree of approx. poly.	approximation error by the optimal minimax polynomial	bootstrapping error		message precision (bits)
		before changing scaling factor e_{min}	after changing scaling factor $e_{\mathsf{min}}\delta_{\mathsf{diff}}$	
60	1.77×10^{-11}	2.83×10^{-10}	1.16×10^{-6}	19.7
62	5.26×10^{-13}	8.50×10^{-12}	3.48×10^{-8}	24.8
64	3.07×10^{-14}	6.17×10^{-13}	2.53×10^{-9}	28.6
66	1.56×10^{-15}	3.76×10^{-13}	1.54×10^{-9}	29.2
68	6.59×10^{-17}	3.76×10^{-13}	1.54×10^{-9}	29.2

Value of δ_{diff}:

The bit length difference between the default scaling factor Δ and the bootstrapping scaling factor Δ_{boot}, which will be denoted as $\delta_{\mathsf{diff}} = \log \Delta_{\mathsf{boot}} - \log \Delta$, is closely related to the message precision. The value of δ_{diff} is usually chosen as 10 bits to lower the difference between the modular reduction function and the sine/cosine function since the half-width of each interval in the approximation region is $2^{-\delta_{\mathsf{diff}}}$.

It causes loss to the message precision of the bootstrapping. In the bootstrapping procedure, it is required to divide the message by $2^{\delta_{\mathsf{diff}}}$ so that it can be included in the approximation region and multiply $2^{\delta_{\mathsf{diff}}}$ at the end of the bootstrapping. If the precision error until multiplying $2^{\delta_{\mathsf{diff}}}$ is e, the final error becomes $2^{\delta_{\mathsf{diff}}}e$. e cannot be reduced below a certain error value because of the rescaling error dealt with in the previous subsection. If this lower bound is denoted as $e_b = 2^{-\delta_b}$, the message precision will be $\delta_b - \delta_{\mathsf{diff}}$.

Because δ_{diff} has a significant effect on both the message precision of the bootstrapping and the message precision of the intended operation in the application, it is desirable to reduce the δ_{diff} to prevent this precision loss. However, the difference between the sine/cosine function and the modular reduction function is somewhat dominant, and this difference becomes more dominant as δ_{diff} increases.

Table 3.3 shows the maximum message precision of the bootstrapping with improved multi-interval Remez algorithm for various δ_{diff}. The degree of the approximate polynomials for each case is set to be large enough to reach the minimum approximate error e_{min}, and the scaling factor and the number of slots are fixed to be 60 and 2^8, respectively.

The bootstrapping error after changing the scaling factor is $e_{\mathsf{min}}\delta_{\mathsf{diff}}$. As δ_{diff} decreases, e_{min} increases rapidly so that the $e_{\mathsf{min}}\delta_{\mathsf{diff}}$ grows. This is because the difference between the modular reduction function and the cosine function becomes larger when the approximation region is enlarged. It can be naively expected that the bootstrapping error can be decreased infinitely when $\log \delta_{\mathsf{diff}}$ is increased, because

$|\epsilon - \sin\epsilon| = O(\epsilon^3)$. However, if the δ_{diff} is larger than 16, the value of e_{min} does not decrease, and thus $e_{\mathsf{min}}\delta_{\mathsf{diff}}$ increases. This lower bound of e_{min} is caused by the rescaling error and the homomorphic linear transform in the bootstrapping. This bound of e_{min} is related to the bootstrapping scaling factor Δ_{boot} and the number of slots, which will be dealt with in the following paragraphs.

Note that it is not required to use the scaling factor $\Delta_{\mathsf{boot}}/\delta_{\mathsf{diff}}$ after bootstrapping. If a scaling factor $2^{-\ell}\Delta_{\mathsf{boot}}/\delta_{\mathsf{diff}}$ is used, the bootstrapping error is amplified by ℓ-bit and the range of the message becomes $[-2^{\ell}, 2^{\ell}]$. Indeed, there are many cases that large range of the message is more important than low bootstrapping error, and thus users can control the scaling factor after bootstrapping concerning the range and the bootstrapping error they want to use.

Table 3.3: Message precision of the bootstrapping with the improved multi-interval Remez algorithm for various values of $\log\delta_{\mathsf{diff}}$

$\log\delta_{\mathsf{diff}}$	bootstrapping error		message precision (bits)
	before changing scaling factor e_{min}	after changing scaling factor $e_{\mathsf{min}}\delta_{\mathsf{diff}}$	
3	2.55×10^{-5}	2.04×10^{-4}	12.3
7	7.45×10^{-9}	9.53×10^{-7}	20.0
10	1.32×10^{-11}	1.35×10^{-8}	26.1
11	1.64×10^{-12}	3.36×10^{-9}	28.1
12	3.76×10^{-13}	1.54×10^{-9}	29.2
13	2.88×10^{-13}	2.36×10^{-9}	28.7
14	2.77×10^{-13}	4.54×10^{-9}	27.7

Bootstrapping Scaling Factor:

Table 3.4 shows the maximum message precision for various bootstrapping scaling factors when the number of slots is 2^8. The degree of the approximate polynomial and the value of δ_{diff} are set to reach the lower bound of e_{min} for each bootstrapping scaling factor and to minimize the value of $e_{\mathsf{min}}\delta_{\mathsf{diff}}$, which determines the actual message precision of the bootstrapping. The second column in Table 3.4 shows the lower bound of e_{min}, the third column shows the value of δ_{diff} which minimizes $e_{\mathsf{min}}\delta_{\mathsf{diff}}$, and the last column shows the maximum message precision with the corresponding bootstrapping scaling factor.

The maximum message precision of the bootstrapping in the RNS-CKKS scheme decreases as the bootstrapping scaling factor decreases. This means that it is required to use as large a bootstrapping scaling factor as possible when precise bootstrapping is needed. Since the bootstrapping scaling factor is related to the multiplicative depth, this gives the trade-off between the depth and the precision.

Note that the bit-length of scaling factors can be different for each level, and thus it is required to use the same scaling factor throughout the bootstrapping. This fact is used in Bossuat et al.'s work [8].

Number of Slots:

Table 3.5 shows the maximum message precision for various numbers of slots when the bootstrapping scaling factor is 60, the maximum scaling factor. The degree of the approximate polynomials and δ_{diff} are set to the same as Table 3.4. The error analysis in [20] shows that the approximation error in SLOTTOCOEFF step is amplified more as more slots are used. The result of Table 3.5 corresponds to this error analysis. This gives the trade-off between the number of slots and the message precision. Note that all precision is less than 32-bit precision. These precision results will be improved in the next subsection by using the composite function method with the inverse sine function.

Table 3.4: Maximum message precision of the bootstrapping with the improved multi-interval Remez algorithm for various bootstrapping scaling factors

$\log \Delta_{\mathsf{boot}}$	$\log \delta_{\mathsf{diff}}$	bootstrapping error		message precision (bits)
		before changing scaling factor e_{min}	after changing scaling factor $e_{\mathsf{min}}\delta_{\mathsf{diff}}$	
50	9	3.30×10^{-10}	1.69×10^{-7}	22.5
54	10	2.21×10^{-11}	2.27×10^{-8}	25.4
57	11	2.88×10^{-12}	5.90×10^{-9}	27.3
60	12	3.71×10^{-13}	1.52×10^{-9}	29.3

Note that $\log \delta_{\mathsf{diff}}$ is not high when the number of slots is large, although this $\log \delta_{\mathsf{diff}}$ value does not ensure the high precision as the difference between the modular reduction function and the sine/cosine function is rather high. This phenomenon is because the coefficients of the message polynomial is very small when the number of slots is large as discussed in [8]. Thus, the approximation region for the cosine function can be generally much less than $1/\delta_{\mathsf{diff}}$.

Table 3.5: Maximum message precision of the bootstrapping with improved multi-interval Remez algorithm for various numbers of slots

$\log n$	degree of approx. poly.	$\log \delta_{\mathsf{diff}}$	bootstrapping error		message precision (bits)	remaining modulus	running time (s)
			before changing scaling factor e_{min}	after changing scaling factor $e_{\mathsf{min}}\delta_{\mathsf{diff}}$			
5	67	14	4.52×10^{-14}	7.42×10^{-10}	30.3	653	91.9
8	66	12	3.71×10^{-13}	1.52×10^{-9}	29.3	653	133.6
10	66	11	1.34×10^{-12}	2.75×10^{-9}	28.4	653	189.3
12	66	9	8.46×10^{-12}	4.33×10^{-9}	27.8	653	287.0
14	66	8	2.46×10^{-11}	6.31×10^{-9}	27.2	653	461.0

At first glance, it seems to be the best method to use the optimal minimax approximate polynomials for the modular reduction function. However, it can be seen that some of the coefficients of the optimal minimax approximate polynomials with regard to the Chebyshev basis are so large that the amplified approximate errors by these coefficients totally distort the messages in the ciphertext. On the other hand, the optimal minimax approximate polynomial coefficients of the scaled sine/cosine functions with more than one scale number are small enough not to distort the messages. Thus, the approximation of the modular reduction function by the sine/cosine function is essential for the correctness of the RNS-CKKS scheme.

When one adhere to the approximation by the scaled sine/cosine function, the difference of the modular reduction function and the sine/cosine function is a crucial obstacle, which is mentioned as an important open problem in Han et al.'s paper [44]. This difference is sharply increased as the approximation region of the modular reduction function becomes longer, and this prevents me from reducing δ_{diff}.

3.2.2 Composite Function Approximation of Modular Reduction Function by Inverse Sine Function

I propose a simple and novel method for solving this problem, which is called the composite function approximation method. In short, the optimal minimax approximate polynomial of the sine/cosine function is composed with the approximate polynomial of the inverse sine function. It is easy to check that if there are two functions f and g for $0 < \epsilon < \frac{1}{4}$ as

$$f : \bigcup_{k=-\infty}^{\infty} [2\pi(k-\epsilon), 2\pi(k+\epsilon)] \to [-\sin 2\pi\epsilon, \sin 2\pi\epsilon], \quad f(x) = \sin x$$

$$g : [-\sin 2\pi\epsilon, \sin 2\pi\epsilon] \to [-2\pi\epsilon, 2\pi\epsilon], \quad g(x) = \arcsin x,$$

then the following equation holds as

$$x - 2\pi \cdot \mathsf{round}\left(\frac{x}{2\pi}\right) = (g \circ f)(x), \quad x \in \bigcup_{k=-\infty}^{\infty} [2\pi(k-\epsilon), 2\pi(k+\epsilon)].$$

If $t = \frac{x}{2\pi}$ is substituted, then

$$\mathsf{normod}(t) = \frac{1}{2\pi}(g \circ f)(2\pi t), \quad t \in \bigcup_{k=-\infty}^{\infty} [k - \epsilon, k + \epsilon].$$

If both f and g are approximated with the optimal minimax approximate polynomials derived by the improved multi-interval Remez algorithm, the modular reduction function can be approximated with any small approximate error by the composition of f and g. Note that $g(x)$ can be approximated very well with some approximate polynomials of a small degree since the domain of $g(x)$ is only one interval. Indeed, the cosine approximation with double-angle formula in [44] can be regarded as the special case of the proposed composite function approximation, in that they approximate $g(x)$ with x, that is, the identity function. Note that the cosine function in [44] is merely a parallel shift of the sine function. Thus, it is said that they approximate the sine function instead of the cosine function.

The sine function f was evaluated by composing the scaled cosine function and several double-angle formulas in [44]. If the number of the used double-angle formula is ℓ, then the functions h_1, h_2, and h_3 are defined as

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

Then, the normod function, which is equivalent to the modular reduction function, can be represented as

$$\mathsf{normod}(x) = h_3 \circ h_2^\ell \circ h_1(x).$$

Thus, if $\tilde{h}_1$ is the optimal minimax approximate polynomial of h_1 and $\tilde{h}_3$ is that of h_3, normod function can be approximated by the composition of several polynoimals as

$$\mathsf{normod}(x) \approx \tilde{h}_3 \circ h_2^\ell \circ \tilde{h}_1(x).$$

With this method, the modular reduction function can be approximated by the composition of several polynomials at arbitrary precision. This enables me to reduce δ_{diff} to 3 and reach the message precision of $\delta_b - \delta_{\mathsf{diff}}$, which is the best precision mentioned in the previous subsection. The next subsection shows that this high precision can be indeed reached in the `SEAL` library.

3.2.3 Simulation Result with **SEAL** Library

This subsection demonstrates that the composite function method can improve the message precision in the RNS-CKKS scheme. The simulation environment is the same as the simulation in Subsection 3.2.1.

Table 3.6 shows that the value of e_{min} with the composite function method does not change. The degrees of approximate polynomials of the scaled cosine function and inverse sine function are set to minimize e_{min}, and these degrees are shown in Table 3.6. In contrast to the result in Table 3.3, all of the values of e_{min} in Table 3.6 are almost the same as the minimum value of e_{min} in Table 3.3 regardless of δ_{diff}. Since e_{min} is fixed with the minimum value, the bootstrapping precision, which is determined by $e_{\mathsf{min}}\delta_{\mathsf{diff}}$, is increased as δ_{diff} decreases.

Table 3.6: Maximum message precision of the bootstrapping with improved multi-interval Remez algorithm and composite function method for various δ_{diff}

$\log \delta_{\mathsf{diff}}$	bootstrapping error		message precision (bits)
	before changing scaling factor e_{min}	after changing scaling factor $e_{\mathsf{min}}\delta_{\mathsf{diff}}$	
3	2.93×10^{-13}	2.34×10^{-12}	38.6
7	2.90×10^{-13}	3.71×10^{-11}	34.6
10	2.85×10^{-13}	2.92×10^{-10}	31.7
11	3.21×10^{-13}	6.58×10^{-10}	30.5
12	2.88×10^{-13}	1.18×10^{-9}	29.7

Table 3.7 shows the maximum precision of the bootstrapping with the improved

multi-interval Remez algorithm and composite function method for various slots. The δ_{diff} value and $\log \Delta_{\mathsf{diff}}$ value are set to be 3 and 60, respectively. The maximum message precision is increased by 5.4-10.2 bits and becomes 32.6-40.5 bit precision. All of the message precision is larger than the 32-bit precision. Thus, the bootstrapping of the RNS-CKKS scheme is made more reliable enough to be used in practical applications.

The half-width of the approximation region has to be $2^{-\delta_{\mathsf{diff}}}$ when the range of real and imaginary part of the messages is assumed to be the same as that of the coefficients of the message polynomial. However, when the messages are sampled from the uniform distribution over the range, the coefficients are significantly reduced so that the approximation region may be reduced as discussed in [8]. ϵ denotes the half-width of each approximation region for each number of slots, and this value is numerically set to have no effect on the bootstrapping. If one wants to be conservative on the distribution of the message and the range of the coefficients, they may set ϵ to be $2^{-\delta_{\mathsf{diff}}}$.

Although the inverse sine approximation procedure is added, the overall running time of the bootstrapping is similar or reduced. Note that the more depth level left in a ciphertext, the more time homomorphic evaluation takes. Since more depth level is consumed in the inverse sine approximation procedure, the ciphertexts in the SlotToCoeff procedure have less remaining depth level. Thus, the running time of the SlotToCoeff procedure in the new bootstrapping is less than that in the original one. The remaining modulus bit length is reduced because of the additional depth consumption of the inverse sine approximation procedure. This additional depth consumption can be seen as a trade-off for the high precision.

Table 3.7: Maximum message precision of the bootstrapping with composite function method for various number of slots

$\log n$	$\log 1/\epsilon$	deg. of app. poly. of cos.	deg. of app. poly. of inv. sine	bootstrapping error		message preci-sion (bits)	rema-ining modu-lus	run-ning time (s)
				before changing scaling factor e_{min}	after changing scaling factor $e_{\mathsf{min}}\delta_{\mathsf{diff}}$			
5	4	71	15	7.93×10^{-14}	6.34×10^{-13}	40.5	473	94.7
8	6	70	9	2.93×10^{-13}	2.34×10^{-12}	38.6	473	133.2
10	9	69	7	1.14×10^{-12}	9.13×10^{-12}	36.7	533	188.9
12	10	69	5	4.84×10^{-12}	3.87×10^{-11}	34.5	533	273.9
14	10	68	5	1.97×10^{-11}	1.53×10^{-10}	32.6	533	451.5

Chapter 4

PRIVACY-PRESERVING DEEP NEURAL NETWORK

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 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 [60], 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 it is desirable 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 it is desirable 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 [33] or Cheon-Kim-Kim-Song (CKKS) scheme [19, 21]. 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 [20, 35]. However, the bootstrapping technique has been actively improved with regard to algorithmic time complexity [8, 16, 44], precision [56], and implementation [48], 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.

For the first time, the ResNet-20 model for the CIFAR-10 dataset [53] is implemented using the residue number system CKKS (RNS-CKKS) [21] FHE scheme, which is a variant of the CKKS scheme using the SEAL library 3.6.1 version [68], one of the most reliable libraries implementing the RNS-CKKS scheme. In addition, bootstrapping of the RNS-CKKS scheme is implemented in the SEAL library according to [8, 16, 20, 44, 56] 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. The ReLU function is first applied based on the composition of minimax approximate polynomials [57] to the encrypted data. Using the results, the possibility of applying FHE with bootstrapping to the standard deep machine learning model is shown 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. [5] 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. The

Softmax function is first implemented in the machine learning model using the method in [19], and this is the first implementation of a privacy-preserving machine learning model based on FHE mitigating the model extraction attack.

The pretrained model parameters is prepared 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. It is found 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, it is verified that the proposed implemented PPML model successfully performs ResNet-20 on encrypted data, even with the model parameters trained for the plaintext model.

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 [2, 26, 30, 37, 45], 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 [45], 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.

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 [5–7, 49, 64]. 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.

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 [60], 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, parameters should be set capable of depth consumption for the desired circuit. Thus, to homomorphically evaluate deeper circuits, large parameters must be set. 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.

4.1 Building Blocks for Privacy-Preserving ResNet Models

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.

4.1.1 Binary Tree Based Implementation of Polynomial Evaluation

For a more intuitive and systematic implementation, the baby-step giant-step polynomial evaluation algorithm is modified 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 10. The output of `DividePoly` is a binary tree useful in the homomorphic polynomial evaluation process.

Algorithm 11 shows the binary tree-based baby-step giant-step polynomial evaluation algorithm. For optimal depth consumption, the leftmost leaf node may be further divided as by Bossuat et al.[8] in lines 3–13. The giant step degree is generalized as an arbitrary integer rather than a power-of-two integer, as in [55].

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 the Chebyshev polynomials are homomorphically evaluated in lines 15 and 17, the formula $m = n$ is used, where $T_0(x)$ is 1. When other Chebyshev polynomials are homomorphically evaluated in line 16, m is set 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 [8]; however, it is easier to design the implementation for the algorithm.

Lee et al. [55] 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 `OddPolyEval` is denoted rather than `PolyEval` in the following subsection, these polynomial evaluation processes are omitted.

Algorithm 10 `DividePoly(p; k)`

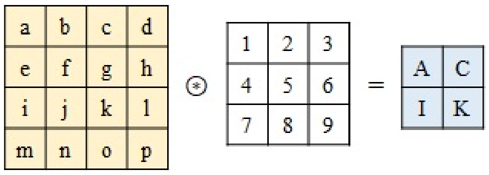
```
Input  : A degree-$d$ polynomial $p$, a giant step parameter $k$
Output: A binary tree $P$ with leaf having polynomials
if $d < k$ then
    return a binary tree $P$ with a single root node having $p$
else
    Find $m$ such that $k \cdot 2^{m-1} < d \leq k \cdot 2^m$.
    Generate a binary tree $P$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

4.1.2 Strided Convolution

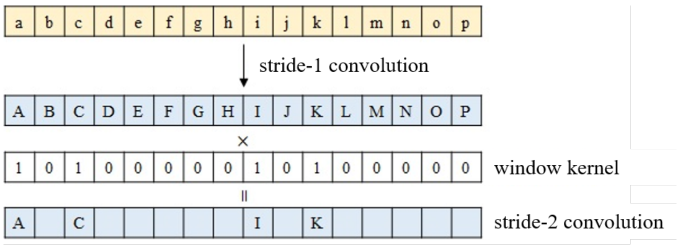
Juvekar et al. [49] proposed an efficient convolution operation for a packing structure in an FHE scheme. They also proposed a strided convolution operation on the homomorphic encryption scheme by decomposing the strided convolution into a sum of nonstrided convolutions. However, their proposed strided convolution operation is not natural for the packing structure in the RNS-CKKS scheme. Furthermore, the fol-

Algorithm 11 `PolyEval`$(\mathsf{ct}, p; k)$

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



(a) Plaintext



(b) Ciphertext

Figure 4.1: Stride-2 convolution.

lowing operations after their strided convolution are difficult to perform on the RNS-CKKS scheme.

I propose an efficient and natural method for strided convolution in the RNS-CKKS scheme. Instead of decomposing the strided convolution, the output of the strided convolution is regared 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 non-strided convolution is performed, 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, homomorphic scalar multiplication is performed 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. 4.1.

It is also found 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 component-wise, this slot structure does not consider the ReLU function. The non-strided convolution to the slot structure after the proposed strided convolution can be performed with Gazelle's convolution method [49], 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, non-strided and strided convolution operations can be performed, even after several strided convolutions.

In ResNet-20, convolution with a stride of one or two is only be used, and thus it is assumed that the strided convolution is convolution with stride two. Each convolution operation should be given an additional parameter $\mathsf{slotstr}$, which represents the slot structure for meaningful data in the input ciphertext of each convolution. The parameter $\mathsf{slotstr}$ 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, Gazelle's convolution method is applied with the steps multiplied by 2^{slotstr}. If the strided convolution is performed, the same procedure is performed as the non-strided convolution, except for the following filtering. A specific algorithm for the strided convolution is presented in Subsection 4.2.3.

4.1.3 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, the Softmax function is implemented in the proposed 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.

Different approximation techniques is used 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, the following approximation methods are chosen, and the entire algorithm is suggested in Subsection 4.2.7.

Exponential Function

If the exponential functions are simply approximated 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 it is required to approximate the exponential function e^x in $[-B, B]$. Then, the function is regarded as $(e^{x/B})^B$. Note that e^y can be approximated in $[-1, 1]$ when $y = x/B$ is set, and the exponential function in this interval is easy to approximate. Thus, the exponential function in $[-1, 1]$ is approximated using the least-squares method, and it is found that the approximate polynomial with degree 12 can approximate sufficiently precisely. Then, when the exponential function is homomorphically evaluated in $[-B, B]$, the input is divided by B, evaluate the approximate polynomial for the exponential function, and exponentiate it with B. If B is set as a power-of-two integer, the exponentiation with B can be implemented by repeated squaring.

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 [23, 38]. 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 $y = 1-x$ is substituted, 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 the Softmax function is approximated.

When the range of the input is $(0, 2R]$, the inverse function is considered 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.

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, the Gumbel Softmax technique can be used, 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 the Gumbel Softmax function is used, 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 $(0, 2R]$.

4.1.4 Position of Bootstrapping

Because the bootstrapping operation is first used in a machine learning model, it is required to consider performing the bootstrapping operation in the middle of the ResNet-20 model. In this subsection, several factors affecting the efficiency of the bootstrapping operation are analyzed.

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. 4.2. Fig. 4.2 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.

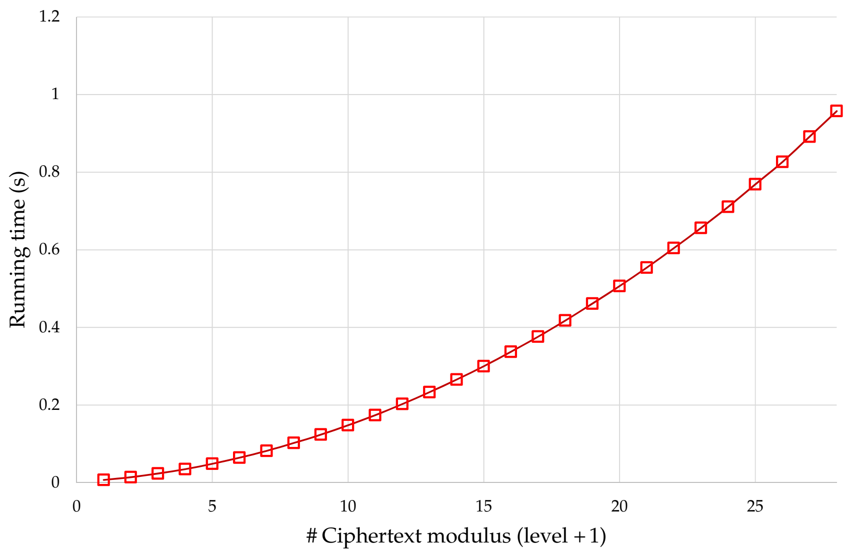
Although the most time-consuming operation in ResNet-20 is the bootstrapping operation, the level of the ciphertexts for each key-switching operation in the bootstrapping is fixed, regardless of the structure of ResNet-20. Thus, it is desirable to compare the number of key-switching operations of the convolution and ReLU functions. it is noted 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 5.4.

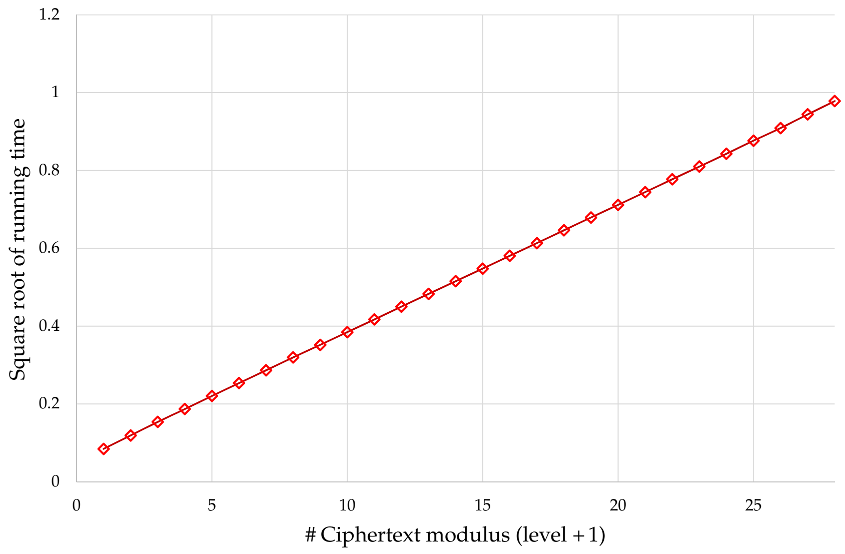
4.2 Implementation Details of ResNet-20 on RNS-CKKS

4.2.1 Structure

Fig. 4.3 shows the structure of the ResNet-20 model and Table 4.1 shows the specification of the ResNet-20. With this structure, the proposed implemented structure is designed for ResNet-20 using the RNS-CKKS scheme, as shown in Fig. 4.4, 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.



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



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

Figure 4.2: 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.

Table 4.1: 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

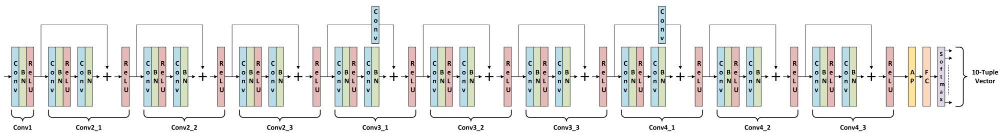


Figure 4.3: Structure of ResNet-20.

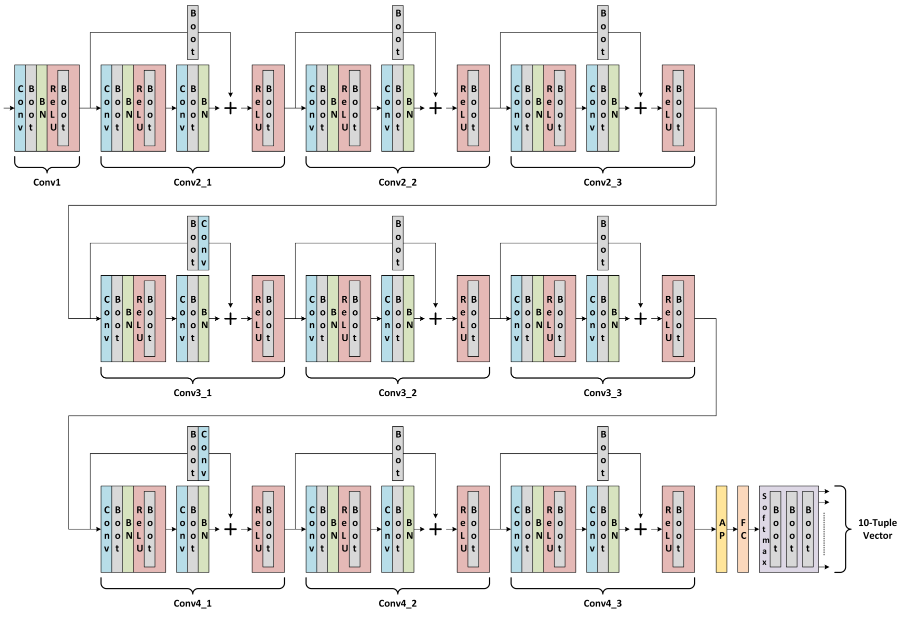


Figure 4.4: Proposed structure of ResNet-20 over RNS-CKKS scheme.

4.2.2 General Setting for RNS-CKKS Scheme

Parameters

The ciphertext polynomial degree is set to 2^{16}, and the secret key Hamming weight is set 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 [22], which is the fastest attack on the LWE with a sparse key. Table 4.2 lists the parameters.

Table 4.2: 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

Data Packing

The message is a $32 \times 32 \times 3$ CIFAR-10 RGB image, and one single image is processed at a time. 2^{15} message slots are used in one ciphertext with the proposed parameters, which is a half of polynomial degree. Rather than using the full slots of the ciphertext, the sparse packing method [20] is employed 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.

In the proposed implementation, it is not sufficient to have ciphertexts composing the encrypted data, but it is required to store the slot structure parameter generated by the strided convolution. For ease of understanding, the dimensions of the tensor are also stored 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 12 shows the detailed algorithm for encrypting image tensors.

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

Algorithm 12 $\texttt{EncTensor}(A \in \mathbb{R}^{L\times L\times H})$

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

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. the absolute input values for ReLU are analyzed, bootstrapping, and Softmax when performing ResNet-20 with several images. Because the observed maximum absolute input value for these procedures is 37.1, it is conjectured 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. it is empirically found that the precision of the approximate polynomial or the function should be at least 16bits below the decimal point; thus, each non-arithmetic function is approximated with 16-bit average precision.

Optimization for Precision of Homomorphic Operations

Several methods is applied to reduce the rescaling and relinearization errors and ensure the precision of the resultant message, such as scaling factor management in [50], lazy rescaling, and lazy relinearization [4, 58]. 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.

4.2.3 Convolution and Batch Normalization

Most of the operations in ResNet-20 are convolutions with zero-padded inputs to maintain their size. I use the packed single-input single-output (SISO) convolution with stride 1 used in Gazelle [49]. Strided convolution with stride 2 is also required to perform downsampling, and it is performed by the method proposed in Subsection 4.1.2. Algorithm 13 shows the detailed algorithm for convolution, which includes both non-strided convolution and strided 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 $\mathsf{slotstr}$, as discussed in Subsection 4.1.2.

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

4.2.4 ReLU

For the first time, the ReLU function in ResNet-20 is implemented with the RNS-CKKS scheme using the composition of the minimax polynomial approximation proposed by Lee et al.[57]. To find an appropriate precision value, a ResNet-20 simulation over the RNS-CKKS scheme is repeatedly performed while changing the precision, and it is found that the minimum 16-bit precision shows good performance on average.

To synthesize the sign function for the ReLU approximation, the composition of the small minimax approximate polynomials is generated with precision parameter $\alpha = 12$ using three minimax approximate polynomials with degrees 7, 15, and 27. Algorithm 14 generates composite polynomials approximating the sign function [54].

Algorithm 13 $\texttt{Conv}(\mathsf{Tensorct}, W, \mathsf{stride})$

```
Input  : An encrypted tensor $\mathsf{Tensorct} = (\{\mathsf{ct}_k\}_{k=0,\cdots,t-1}, \ell, \mathsf{slotstr}, t)$, 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 $\mathsf{str}$
Output: An output encrypted tensor $\mathsf{Tensorct}'$
$L \leftarrow \ell \cdot \mathsf{slotstr}$
for $h = 0$ to $t' - 1$ do
    $\mathsf{ct}'_h \leftarrow \mathbf{0}$
    for $k = 0$ to $t - 1$ do
        for $(i, j) = (0, 0)$ to $(c - 1, c - 1)$ do
            $w \leftarrow \mathbf{0} \in \mathbb{R}^{L^2}$
            for $(i', j') = (0, 0)$ to $(\ell - 1, \ell - 1)$ do
                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
                    $w[(i' \cdot L + j') \cdot \mathsf{slotstr} \cdot \mathsf{str}] \leftarrow W[i, j, k, h]$
                end
            end
            $r \leftarrow (i - \lfloor c/2 \rfloor) \cdot L + (j - \lfloor c/2 \rfloor)$
            $\mathsf{ct}'_h \leftarrow \mathsf{ct}'_h \oplus (w \odot \texttt{rot}(\mathsf{ct}_k, r \cdot \mathsf{slotstr}))$
        end
    end
end
return $(\{\mathsf{ct}'_h\}_{h=0,\cdots,t'-1}, \ell/\mathsf{str}, \mathsf{slotstr} \cdot \mathsf{str}, t')$
```

GenMinimax(f, d, D) in Algorithm 14 is an algorithm that generates the minimax approximate polynomial with degree d for function f on domain D, and this algorithm is implemented using the multi-interval Remez algorithm [56]. Range(f, D) denotes the range of f in the domain D.

Algorithm 15 shows the homomorphic evaluation method for the ReLU function using the composite polynomials generated by Algorithm 14 as the input. After homomorphically evaluating the p_i's in order, $x(1 + \mathrm{sign}(x))/2$ is homomorphically evaluated.

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 [55] and the optimal level consumption method in [8]. Because the homomorphic evaluation of polynomial compositions consumes many depths, it is impossible to complete it without bootstrapping. Thus, bootstrapping is used twice in a layer, once in the middle, and once at the end of evaluating the ReLU function.

Algorithm 14 GenSignPoly$(\alpha, \{d_i\}_i)$

```
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
        $p_{s-1} \circ \cdots \circ p_0(x) \approx \mathrm{sign}(x)$
for $i = 0$ to $s - 1$ do
    if $i = 0$ then
        $D_0 \leftarrow [-1, -2^{-\alpha}] \cup [2^{-\alpha}, 1]$
    else
        $D_i \leftarrow \mathsf{Range}(p_{i-1}, D_{i-1})$
    end
    $p_i \leftarrow \mathsf{GenMinimax}(\mathrm{sign}, d_i, D_i)$
end
```

Algorithm 15 $\texttt{ReLU}(\mathsf{Tensorct}, \{p_i\}_i)$

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

4.2.5 Bootstrapping

Because it is required 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, the bootstrapping technique is applied 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, the most advanced bootstrapping with the SEAL library [8, 48, 56] is implemented. CoeffToSlot and SlotToCoeff are implemented using a collapsed FFT structure [16] 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 [44, 56], where the cosine function and the inverse sine function are approximated using the multi-interval Remez algorithm as in [56].

The most crucial issue when using bootstrapping in the RNS-CKKS scheme is bootstrapping failure. More than a thousand bootstrapping procedures are required in the proposed 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 $\cup_{i=-(K-1)}^{K-1}[i-\epsilon, i+\epsilon]$ [20]. 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, it is required to investigate the relationship between the bootstrapping failure probability and the parameter K.

Table 4.3: 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} [8]	12	17	21
2^{-30}	14	20	24
2^{-40}	16	23	28

It is described 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 $a \cdot s + b = m + e \pmod{q}$ for secret key s; hence, $a \cdot s + b \approx m + q \cdot I \pmod{Q}$, 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 $I < K = O(\sqrt{h})$ [58]. 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.

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 the ciphertext and thousands of bootstrapping are performed, the failure probability is very high when using previous approximate polynomials.

In Table 4.3, several bounds are presented 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 4.3, a trade-off can be offered between the evaluation time and failure probability of the entire network. Following [8, 58], the approximated modular reduction in the CKKS bootstrapping thus far has a failure probability $\approx 2^{-23}$, but it is not sufficiently small because it is required to perform many bootstrapping procedures for ResNet-20. Thus, the bootstrapping failure probability is set to less than 2^{-40} in the proposed 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 [56]. The number of double-angle formulas ℓ is set to two.

4.2.6 Average Pooling and Fully Connected Layer

The size of the tensor after all convolutional layers are performed is $8 \times 8 \times 64$. The average pooling is performed 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 the fully connected layer is performed. Algorithm 16 shows the detailed procedures for average pooling and the fully connected layers.

Algorithm 16 `AvgPoolFullCon`($\mathsf{Tensorct}, W$)

```
Input  : An encrypted tensor $\mathsf{Tensorct} = (\{\mathsf{ct}_k\}_{k=0,\cdots,t-1}, \ell, \mathsf{slotstr}, t)$, weight parameters for fully connected layer $W \in \mathbb{R}^{T \times t}$
Output: An array of ciphertexts $\{\mathsf{ct}'_k\}_{k=0,\cdots,T-1}$
for $k = 0$ to $t-1$ do
    $\bar{\mathsf{ct}}_k \leftarrow \mathsf{ct}_k$
    for $i = 0$ to $\log \ell - 1$ do
        $\mathsf{tmpct} \leftarrow \mathtt{rot}(\bar{\mathsf{ct}}_k, \mathsf{slotstr} \cdot 2^i)$
        $\bar{\mathsf{ct}}_k \leftarrow \bar{\mathsf{ct}}_k \oplus \mathsf{tmpct}$
    end
    for $j = 0$ to $\log \ell - 1$ do
        $\mathsf{tmpct} \leftarrow \mathtt{rot}(\bar{\mathsf{ct}}_k, \mathsf{slotstr} \cdot \ell \cdot 2^i)$
        $\bar{\mathsf{ct}}_k \leftarrow \bar{\mathsf{ct}}_k \oplus \mathsf{tmpct}$
    end
end
for $u = 0$ to $T-1$ do
    $\mathsf{ct}'_u \leftarrow \mathbf{0}$
    for $k = 0$ to $t-1$ do
        $\mathsf{ct}'_u \leftarrow \mathsf{ct}'_u \oplus (W[u,k] \odot \bar{\mathsf{ct}}_k)$
    end
end
return $\{\mathsf{ct}'_k\}_{k=0,\cdots,T-1}$
```

4.2.7 Softmax

The Softmax method proposed in Subsection 4.1.3 is used. 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 Subsection 4.1.3, The value 64 is used 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 17 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.

4.3 Multiplexed Convolution

4.3.1 Comparison of Bootstrapping Runtime for Several Data Packing Methods

Since the most time-consuming component in the implementation of standard ResNet on the RNS-CKKS scheme is bootstrapping, it is desirable to reduce the bootstrapping runtime. The required number of KSOs and the runtime for bootstrapping according to the number of slots are presented in Table 4.4, where the runtime is obtained using the same parameters and simulation environments of Section 4.7. To reduce the total bootstrapping runtime, intermediate data should be packed into ciphertexts as compact as possible during the inference stage. In addition, the gap between valid data is increased by a factor of s after each strided convolution, leading to a reduction of packing density by a factor of s^2, but this low packing density should be resolved to effectively reduce the bootstrapping runtime.

In this section, it is attempted to remove this gap by packing these sparsely packed

Algorithm 17 $\texttt{Softmax}(\mathsf{Tensorct}, B, R, \lambda)$

```
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 approxima-
         tion parameter $d$
Output: An encrypted one-hot vector $\{\mathsf{ct}'_k\}_{k=0,\cdots,T-1}$
$p_{\exp} \leftarrow \texttt{GenApproxPoly}(e^x, [-1,1])$
$\tilde{\mathsf{ct}} \leftarrow \mathbf{0}$
for $k = 0$ to $T-1$ do
    $\mathsf{ct}'_k \leftarrow 1/B \odot \mathsf{ct}_k$
    $\mathsf{ct}'_k \leftarrow \texttt{polyeval}(\mathsf{ct}'_k, p_{\exp})$
    for $i = 0$ to $\log B - \log \lambda$ do
        $\mathsf{ct}'_k \leftarrow \mathsf{ct}'_k \otimes \mathsf{ct}'_k$
    end
    $\tilde{\mathsf{ct}} \leftarrow \tilde{\mathsf{ct}} \oplus \mathsf{ct}'_k$
end
$\tilde{\mathsf{ct}} \leftarrow 2 \ominus 1/R \odot \tilde{\mathsf{ct}}$
$\mathsf{tmpct} \leftarrow \tilde{\mathsf{ct}} \ominus 1$
for $j = 0$ to $d-1$ do
    $\mathsf{tmpct} \leftarrow \mathsf{tmpct} \otimes \mathsf{tmpct}$
    $\tilde{\mathsf{ct}} \leftarrow \tilde{\mathsf{ct}} \otimes (1 \oplus \mathsf{tmpct})$
end
for $k = 0$ to $T-1$ do
    $\mathsf{ct}'_k \leftarrow \mathsf{ct}_k \otimes \tilde{\mathsf{ct}}$
end
return $\{\mathsf{ct}'_k\}_{k=0,\cdots,T-1}$
```

boot $\log_2$(#slots)	10	11	12	13	14	15
#KSOs	63	70	77	84	91	94
runtime	72s	80s	86s	96s	112s	140s

Table 4.4: The number of KSOs and bootstrapping runtime according to various number of slots for bootstrapping

data in a compact manner. Several data packing methods are compared. It is assumed that the data of size less than n_t is packed in a ciphertext using RS packing so that RS bootstrapping can be used.

Gap packing Since gap packing packs only one channel data into one ciphertext, the required number of bootstrapping operations will be the same as the number of channels. Thus, an unnecessarily large number of KSOs are required.

Gap packing with multiple channels The gap packing can be improved by packing data of multiple channels into one ciphertext as much as possible. Although this packing can reduce the number of bootstrappings a lot, there are still many dummy slots as shown in Figure 4.5(c). For CNNs with many strided convolutions, the total bootstrapping runtime will increase exponentially with the number of strided convolutions.

Multiplexed packing Recently, HEAR [51] used a new data packing method, referred to *multiplexed packing* herein. In this packing method, plaintext tensors of $h_i \times w_i$ size for k_i^2 channels are first mapped to one larger *multiplexed tensor* of size $k_ih_i \times k_iw_i$. Then, several multiplexed tensors are encrypted in one ciphertext. Although multiplexed packing was proposed to deal with the pooling of HE-friendly CNNs and speed up convolution in [51], it is repurposed to reduce the bootstrapping runtime of CNNs with strided convolutions. Figure 4.6 describes multiplexed packing with $h_i = w_i = 4$ and $k_i = 2$. The formal description of multiplexed packing can be seen in Section 4.4.4.

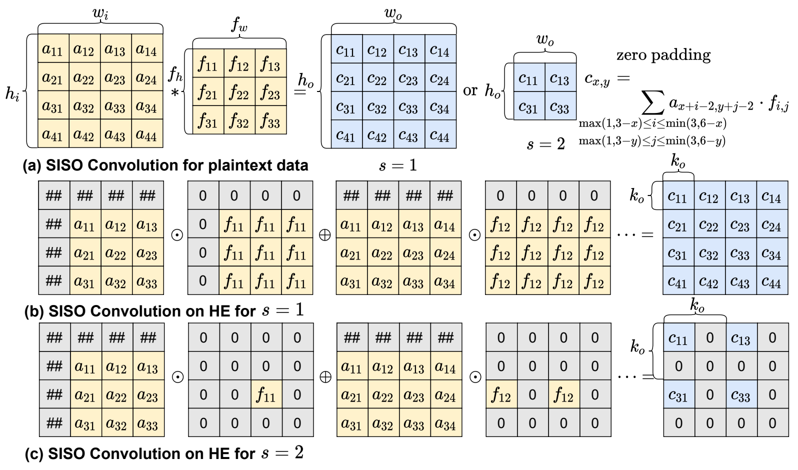


Figure 4.5: SISO convolution on HE [49].

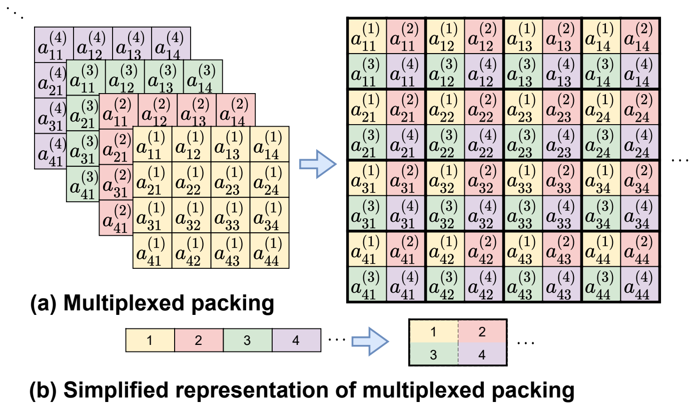


Figure 4.6: Multiplexed packing MultPack when $h_i = w_i = 4$ and $k_i = 2$.

Figure 4.7 illustrates several packing methods for $k_i = 2$, where $c_n = \frac{n_t}{k_i^2 h_i w_i}$. Table 4.5 shows the required number of bootstrappings for implementation of ResNet-20 when each data packing method is used. The number of KSOs for total bootstrappings in ResNet-20 inference is also presented, and it is substantially reduced by multiplexed packing.

Thus, it is required that the corresponding plaintext data be packed in the ciphertext using multiplexed packing during ResNet inference. Then, it is required to design a homomorphic convolution that takes an input ciphertext of multiplexed input tensor and outputs a ciphertext of multiplexed output tensor.

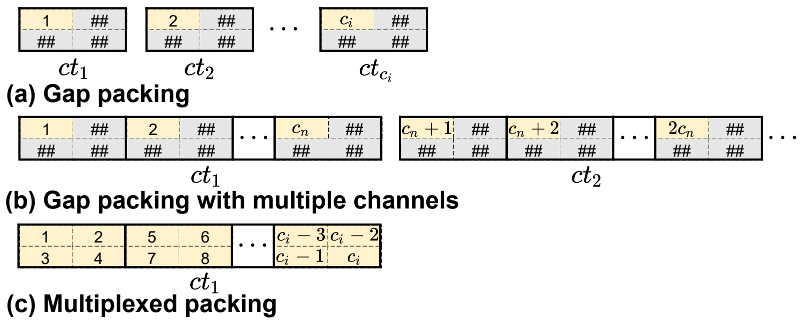


Figure 4.7: Comparison of several data packing methods.

4.3.2 Multiplexed Parallel Convolution on Fully Homomorphic Encryption

In this subsection, the homomorphic convolution algorithms are proposed so that take a ciphertext having multiplexed input tensor and output a ciphertext having multiplexed output tensor. HEAR [51] proposed such a convolution algorithm that supports stride one (i.e., $s = 1$). HEAR performs homomorphic convolution on multiple input channels simultaneously, adds SISO convolution results for all input channels, and collects only valid values by multiplying dummy slots by zero. This convolution algorithm is generalized to support the strided convolutions (i.e., $s \geq 2$). it is proposed to select and collect the valid values for the output gap $k_o = sk_i$ instead of the input gap k_i. Then, the output ciphertext has the plaintext output of strided convolution in the form of multiplexed tensor for $k_o = sk_i$. This convolution algorithm is denoted as $MultConv$ and Figure 4.8 describes the procedure of $MultConv$ when $s = 2$.

Unlike previous works for HE-friendly networks not relying on bootstrapping, a large number of full slots are required, which is usually larger than the data size, to support bootstrapping and precise APRs. First, packing data is considered into ciphertext using RS packing so that RS bootstrapping can be used. Then, it is noted that one

$\log_2$(#slots)	before Str_conv1			before Str_conv2			after Str_conv2			total ResNet-20		
	(a)	(b)	(c)	(a)	(b)	(c)	(a)	(b)	(c)	(a)	(b)	(c)
10	16	0	0	32	0	0	64	0	0	672	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	1	0	0	6
13	0	0	0	0	0	1	0	0	0	0	0	6
14	0	1	1	0	0	0	0	0	0	0	6	6
15	0	0	0	0	1	0	0	2	0	0	18	0
total #KSOs										42,336	2,238	1,512

Table 4.5: The number of bootstrappings for implementation of ResNet-20 according to various data packing methods. (a), (b), and (c) imply gap packing, gap packing with multiple channels, and multiplexed packing, respectively. Str_conv1 and Str_conv2 denote the first and the second strided convolutions ($s = 2$) in ResNet-20, respectively.

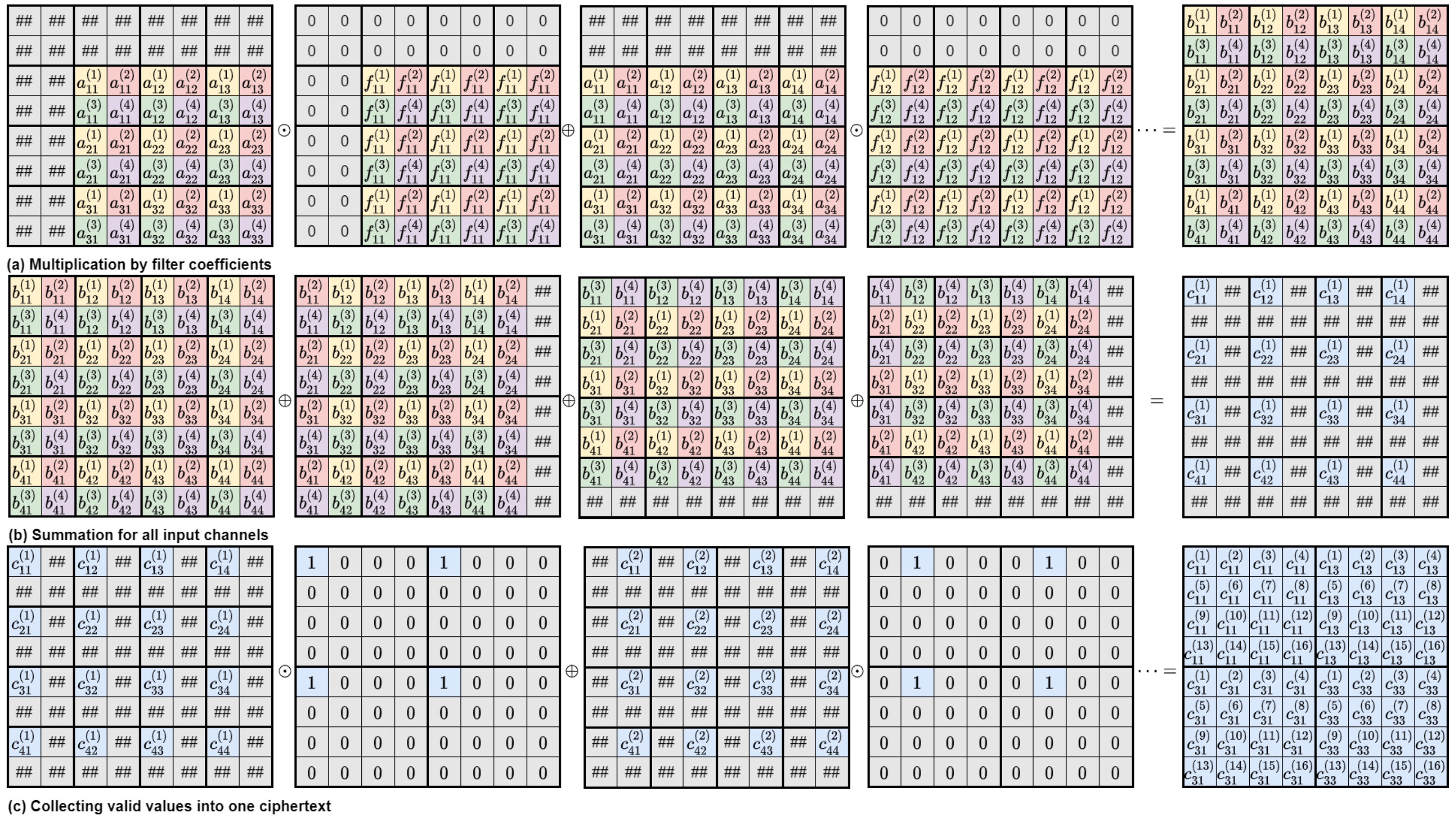

(a) Multiplication by filter coefficients
(b) Summation for all input channels
(c) Collecting valid values into one ciphertext

Figure 4.8: Multiplexed convolution algorithm for multiplexed input tensor for $s = 2$, $k_i = 2$, and $h_i = w_i = 4$.

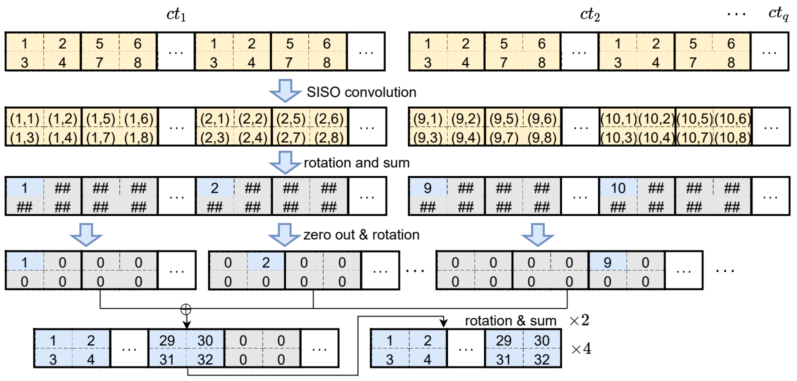


Figure 4.9: Multiplexed parallel convolution algorithm when $k_i = 2$ and $c_o = c_n = 32$.

input channel is repeatedly used for SISO convolutions for multiple output channels. A multiplexed parallel convolution algorithm is proposed, denoted as $MultParConv$, that simultaneously performs SISO convolutions for multiple output channels, which consider the input packed by RS packing as just several independent inputs. This algorithm reduces the convolution runtime of $MultConv$ while still compatible with RS bootstrapping. Figure 4.9 shows the procedure of $MultParConv$ using simplified representation of multiplexed packing.

The detailed algorithms of $MultConv$ and $MultParConv$ are presented in Section 4.4.5. Each execution of $MultConv$ and $MultParConv$ requires $f_h f_w - 1 + c_o(2\lceil \log_2 k_i \rceil + \lceil \log_2 t_i \rceil + 1)$ and $f_h f_w - 1 + q(2\lceil \log_2 k_i \rceil + \lceil \log_2 t_i \rceil) + c_o + \log_2 p_o$ rotations, respectively, where $t_i = \lceil \frac{c_i}{k_i^2} \rceil$, $t_o = \lceil \frac{c_o}{k_o^2} \rceil$, $p_i = 2^{\lfloor \log_2 (\frac{n_t}{k_i^2 h_i w_i t_i}) \rfloor}$, $p_o = 2^{\lfloor \log_2 (\frac{n_t}{k_o^2 h_o w_o t_o}) \rfloor}$, and $q = \lceil \frac{c_o}{p_i} \rceil$. Then, the total required rotations for $MultConv$ and $MultParConv$ in ResNet-20 inference are 4,360 and 1,657, respectively, which implies that $MultParConv$ requires 62% fewer rotations (i.e., number of KSOs) than $MultConv$.

4.4 Details of Multiplexed Convolution

4.4.1 Notations and Description of Parameters

In this subsection, specific notations and description of parameters are provided. x is used to denote a vector in R^n for some integer n. For $x = (x_0, x_1, \cdots, x_{n-1})$, $\langle x \rangle_r$ denotes the cyclically shifted vector of x by r to the left, that is, $(x_r, x_{r+1}, \cdots, x_{n-1}, x_0, \cdots, x_{r-1})$. $x \cdot y$ denotes the component-wise multiplication $(x_0 y_0, \cdots, x_{n-1} y_{n-1})$. For an integer $a \in Z$, the remainder of a divided by q is denoted by a mod q. For a real number $x \in R$, $\lceil x \rceil$ denotes the least integer greater than or equal to x, and $\lfloor x \rfloor$ denotes the greatest integer less than or equal to x.

In this dissertation, various parameters such as $h_i, h_o, w_i, w_o, c_i, c_o, f_h, f_w, s, k_i, k_o, t_i, t_o, p_i, p_o,$ and q are used, and the values of these parameters are determined differently for each component such as convolution, batch normalization (or convolution/batch normalization integration in Subsection 4.6.1), downsampling, and average pooling. The specific values of parameters that are used in the simulation can be seen in Table 4.6.

4.4.2 Mapping of Three-Dimensional Tensor to One-Dimensional Vector

It is often necessary to map three-dimensional tensor $A \in R^{h_i \times w_i \times c_i}$ to one-dimensional vector in R^{n_t} to perform convolutions on the HE scheme, and A can be the original tensor or *(parallelly) multiplexed tensor* defined in Subsection 4.4.5. The following is the definition of Vec function that is used to map tensor A to a vector in R^{n_t},

$Vec(A) = y = (y_0, \cdots, y_{n_t-1}) \in R^{n_t}$ such that

$$y_i = \begin{cases} A_{\lfloor (i \bmod h_i w_i)/w_i \rfloor, i \bmod w_i, \lfloor i/h_i w_i \rfloor}, & 0 \le i < h_i w_i c_i, \\ 0, & \text{otherwise.} \end{cases}$$

Figure 4.10 describes this Vec function.

component		f_h	f_w	s	h_i	h_o	w_i	w_o	c_i	c_o	k_i	k_o	t_i	t_o	p_i	p_o	q
$ConvBN1$		3	3	1	32	32	32	32	3	16	1	1	3	16	8	2	2
$ConvBN2\_xa$		3	3	1	32	32	32	32	16	16	1	1	16	16	2	2	8
$ConvBN2\_xb$		3	3	1	32	32	32	32	16	16	1	1	16	16	2	2	8
$ConvBN3\_xa$	$x = 1$	3	3	2	32	16	32	16	16	32	1	2	16	8	2	4	16
	otherwise	3	3	1	16	16	16	16	32	32	2	2	8	8	4	4	8
$ConvBN3\_xb$		3	3	1	16	16	16	16	32	32	2	2	8	8	4	4	8
$ConvBN4\_xa$	$x = 1$	3	3	2	16	8	16	8	32	64	2	4	8	4	4	8	16
	otherwise	3	3	1	8	8	8	8	64	64	4	4	4	4	8	8	8
$ConvBN4\_xb$		3	3	1	8	8	8	8	64	64	4	4	4	4	8	8	8
$ConvBN\_s1$		1	1	2	32	16	32	16	16	32	1	2	16	8	2	4	16
$ConvBN\_s2$		1	1	2	16	8	16	8	32	64	2	4	8	4	4	8	16
$Downsamp1$		-	-	-	32	16	32	16	16	32	1	2	16	8	2	4	-
$Downsamp2$		-	-	-	16	8	16	8	32	64	2	4	8	4	4	8	-

Table 4.6: Parameters that are used in each ConvBN or Downsamp process

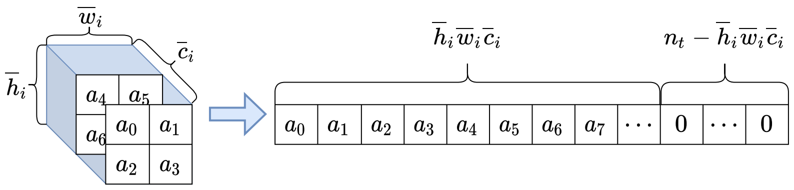

$\overline{w}_i$
$\overline{c}_i$
$\overline{h}_i$
a_4 a_5
a_6 a_0 a_1
a_2 a_3
$\overline{h}_i\overline{w}_i\overline{c}_i$
$n_t - \overline{h}_i\overline{w}_i\overline{c}_i$
a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7 $\cdots$ 0 $\cdots$ 0

Figure 4.10: Vec function that maps a given tensor in $R^{h_i \times w_i \times c_i}$ to a vector in R^{n_t}.

In this dissertation, $n_t = 2^{15}$ is used, and this allows that all tensors to be encrypted can be packed into one ciphertext, that is, $h_i w_i c_i \leq n_t$ for each tensor $A \in R^{h_i \times w_i \times c_i}$. In several figures in this dissertation, a three-dimensional tensor A is often identified as $Vec(A)$ or the corresponding ciphertext $Enc(Vec(A))$. In addition, for a three-dimensional tensor A, the rotation of ciphertext of $Vec(A)$, that is, $Rot(Enc(Vec(A)); r)$ for some nonnegative integer r is refered to as rotation of tensor A. When a tensor is rotated, each element moves to the left, but it goes up when it reaches the leftmost point, and it moves to the front page when it reaches the top leftmost point. Furthermore, for two tensors A and B, homomorphic addition, subtraction, and multiplication of $Enc(Vec(A))$ and $Enc(Vec(B))$ are referred to as those of A and B, respectively.

4.4.3 Batch Normalization on Homomorphic Encryption

Batch normalization [46] should be performed for the output tensor of convolution. As in convolution, h_i, w_i, and c_i are parameters representing the size of the input tensor, and h_o, w_o, and c_o are parameters representing the size of the output tensor in batch normalization. That is, batch normalization outputs a tensor $A' \in R^{h_o \times w_o \times c_o}$ for some input tensor $A \in R^{h_i \times w_i \times c_i}$. $h_i = h_o, w_i = w_o$, and $c_i = c_o$ are satisfied for batch normalization.

The weight, running variance, running mean, and bias of batch normalization are denoted by $T, V, M, I \in R^{c_i}$. A constant vector $C = (C_0, C_1, \cdots, C_{c_i - 1}) \in R^{c_i}$ is considered such that $C_j = \frac{T_j}{\sqrt{V_j + \epsilon}}$ for $0 \leq j < c_i$, where ϵ is an added value for numerical stability. Then, batch normalization can be seen as evaluating the equation $C_j \cdot (A_{i_1, i_2, j} - M_j) + I_j$ for $0 \leq i_1 < h_i, 0 \leq i_2 < w_i$, and $0 \leq j < c_i$.

For the description of batch normalization on HE, it is required to define C, M, and $I \in R^{h_i \times w_i \times c_i}$ first. C, M, and I are defined as $C_{i_1, i_2, j} = C_j$, $M_{i_1, i_2, j} = M_j$, and $I_{i_1, i_2, j} = B_j$ for $0 \leq i_1 < h_i, 0 \leq i_2 < w_i$, and $0 \leq j < c_i$, respectively. Then, batch normalization can be performed using the equation $Vec(C) \cdot (Vec(A) - Vec(M)) +$

$Vec(I) = Vec(C)\cdot Vec(A) + (Vec(I) - Vec(C)\cdot Vec(M))$. This can be implemented on HE by using one homomorphic addition and scalar multiplication. That is, for the input tensor ciphertext ct_a, it is performed that $Vec(C) \odot ct_a \oplus (Vec(I) - Vec(C)\cdot Vec(M))$.

4.4.4 Multiplexed Packing

For $t_i = \lceil \frac{c_i}{k_i^2} \rceil$, $MultPack$ is the function that maps a tensor $A = (A_{i_1,i_2,i_3})_{0\le i_1<h_i, 0\le i_2<w_i, 0\le i_3<c_i} \in R^{h_i\times w_i\times c_i}$ to a ciphertext $Enc(Vec(A')) \in R^{n_t}$, where $A' = (A'_{i_3,i_4,i_5})_{0\le i_3<k_ih_i, 0\le i_4<k_iw_i, 0\le i_5<t_i} \in R^{k_ih_i\times k_iw_i\times t_i}$ is a *multiplexed tensor* such that

$$A'_{i_3,i_4,i_5} = \begin{cases} A_{\lfloor i_3/k_i \rfloor, \lfloor i_4/k_i \rfloor, k_i^2 i_5 + k_i(i_3 \bmod k_i) + i_4 \bmod k_i}, & \text{if } k_i^2 i_5 + k_i(i_3 \bmod k_i) + i_4 \bmod k_i < c_i, \\ 0, & \text{otherwise,} \end{cases}$$

for $0 \le i_3 < k_ih_i, 0 \le i_4 < k_iw_i$, and $0 \le i_5 < t_i$.

This multiplexed packing method is a generalized version of raster scan packing method, and it is the same as raster scan packing method using Vec when $k_i = 1$. Each corresponding plaintext tensor is required to be packed into the ciphertext slots using the multiplexed packing method throughout the entire CNN, where the value of gap k_i can be changed.

4.4.5 Convolution Algorithms for Multiplexed Tensor

Multiplexed Convolution

For description of $MultConv$ algorithm, some definitions and a subroutine algorithm are required.

The filter (weight tensor) of the convolution is $U \in R^{f_h\times f_w\times c_i\times c_o}$. First, $MultWgt(U; i_1, i_2, i)$ function that maps a weight tensor $U \in R^{f_h\times f_w\times c_i\times c_o}$ to an element of R^{n_t} is defined. Before the definition of $MultWgt$, three-dimensional *multiplexed shifted weight tensor* $U'^{(i_1,i_2,i)} = (U'^{(i_1,i_2,i)}_{i_3,i_4,i_5})_{0\le i_3<k_ih_i, 0\le i_4<k_iw_i, 0\le i_5<t_i} \in$

$R^{k_ih_i \times k_iw_i \times t_i}$ for given i_1, i_2, and i, where $0 \le i_1 < f_h, 0 \le i_2 < f_w$, and $0 \le i < c_o$ is defined as follows:

$$U'^{(i_1,i_2,i)}_{i_3,i_4,i_5} = \begin{cases} 0, & \text{if } k_i^2 i_5 + k_i(i_3 \bmod k_i) + i_4 \bmod k_i \ge c_i \\ & \text{or } \lfloor i_3/k_i \rfloor - (f_h - 1)/2 + i_1 \notin [0, h_i - 1] \\ & \text{or } \lfloor i_4/k_i \rfloor - (f_w - 1)/2 + i_2 \notin [0, w_i - 1], \\ U_{i_1, i_2, k_i^2 i_5 + k_i(i_3 \bmod k_i) + i_4 \bmod k_i, i}, & \text{otherwise,} \end{cases}$$

for $0 \le i_3 < k_ih_i$, $0 \le i_4 < k_iw_i$, and $0 \le i_5 < t_i$. Then, $MultWgt$ function is defined as $MultWgt(U; i_1, i_2, i) = Vec(U'^{(i_1,i_2,i)})$.

In addition to the weight tensor, it is also required to define *multiplexed selecting tensor* $S'^{(i)} = (S'^{(i)}_{i_3,i_4,i_5})_{0 \le i_3 < k_oh_o, 0 \le i_4 < k_ow_o, 0 \le i_5 < t_o} \in R^{k_oh_o \times k_ow_o \times t_o}$, which is used to select valid values in $MultConv$ algorithm, where $t_o = \lfloor \frac{c_o}{k_o^2} \rfloor$. Multiplexed selecting tensor $S'^{(i)}$ is defined as follows:

$$S'^{(i)}_{i_3,i_4,i_5} = \begin{cases} 1, & \text{if } k_o^2 i_5 + k_o(i_3 \bmod k_o) + i_4 \bmod k_o = i \\ 0, & \text{otherwise,} \end{cases}$$

for $0 \le i_3 < k_oh_o, 0 \le i_4 < k_ow_o$, and $0 \le i_5 < t_o$.

$SumSlots$ is a useful subroutine algorithm that adds m slot values spaced apart by p. Algorithm 18 shows the $SumSlots$ algorithm. Then, Algorithm 19 describes the proposed multiplexed convolution algorithm, $MultConv$ using $MultWgt$ function, multiplexed selecting tensor $S'^{(i)}$, and $SumSlots$ algorithm. Here, ct_{zero} is a ciphertext of all-zero vector $0 \in R^{n_t}$.

Multiplexed Parallel Convolution

I propose a *multiplexed parallel packing* method $MultParPack$ that packs p_i identical multiplexed tensors into one ciphertext for $p_i = 2^{\lfloor \log_2(\frac{n_t}{k_i^2 h_i w_i t_i}) \rfloor}$. Figure 4.11 describes how to perform multiplexed parallel packing of $3 \times 3 \times c_i$ input tensor for given

Algorithm 18 $SumSlots(ct_a; m, p)$

```
1:  Input: Tensor ciphertext $ct_a$, number of added slots $m$, and gap $p$
2:  Output: Tensor ciphertext $ct_c$
3:  $ct_b^{(0)} \leftarrow ct_a$
4:  for $j \leftarrow 1$ to $\lfloor \log_2 m \rfloor$ do
5:      $ct_b^{(j)} \leftarrow ct_b^{(j-1)} \oplus Rot(ct_b^{(j-1)}; 2^{j-1} \cdot p)$
6:  end for
7:  $ct_c \leftarrow ct_b^{(\lfloor \log_2 m \rfloor)}$
8:  for $j \leftarrow 0$ to $\lfloor \log_2 m \rfloor - 1$ do
9:      if $\lfloor m/2^j \rfloor \bmod 2 = 1$ then
10:         $ct_c \leftarrow ct_c \oplus Rot(ct_b^{(j)}; \lfloor m/2^{j+1} \rfloor \cdot 2^{j+1} p)$
11:     end if
12: end for
13: Return $ct_c$
```

gap $k_i = 2$ and number of copies p_i. For the input tensor $A \in R^{h_i \times w_i \times c_i}$, this function first obtains a multiplexed tensor $A' \in R^{k_i h_i \times k_i w_i \times t_i}$ such that $MultPack(A) = Enc(Vec(A'))$ and simply places p_i copies of A' in sequence. This extended tensor is mapped to a vector in R^{n_t} using Vec function and then encrypted into a ciphertext. If $k_i^2 h_i w_i t_i \nmid n_t$, some zeros are filled between p_i copies of A'. The definition of $MultParPack$ function is given as:

$$MultParPack(A) = \bigoplus_{j=0}^{p_i-1} Rot(MultPack(A); j(n_t/p_i)).$$

Each corresponding plaintext tensor is required to be packed into the ciphertext slots using the multiplexed parallel packing method during the entire CNN. A *multiplexed parallel convolution* algorithm is proposed, $MultParConv$, which is an improved algorithm of $MultConv$. $MultParConv$ takes a *parallelly multiplexed tensor* for gap k_i as an input and outputs a parallelly multiplexed tensor for output gap k_o. Let $q = \lceil \frac{c_o}{p_i} \rceil$. Then, while the previous multiplexed convolution algorithm $MultConv$

Algorithm 19 $MultConv(ct'_a, U)$

```
1:  Input: Multiplexed tensor ciphertext $ct'_a$ and weight tensor $U$
2:  Output: Multiplexed tensor ciphertext $ct'_d$
3:  $ct'_d \leftarrow ct_{zero}$
4:  for $i_1 \leftarrow 0$ to $f_h - 1$ do
5:      for $i_2 \leftarrow 0$ to $f_w - 1$ do
6:          $ct'^{(i_1,i_2)} \leftarrow Rot(ct'_a; k_i^2 w_i(i_1 - (f_h - 1)/2) + k_i(i_2 - (f_w - 1)/2))$
7:      end for
8:  end for
9:  for $i \leftarrow 0$ to $c_o - 1$ do
10:     $ct'_b \leftarrow ct_{zero}$
11:     for $i_1 \leftarrow 0$ to $f_h - 1$ do
12:         for $i_2 \leftarrow 0$ to $f_w - 1$ do
13:             $ct'_b \leftarrow ct'_b \oplus ct'^{(i_1,i_2)} \odot MultWgt(U; i_1, i_2, i)$
14:         end for
15:     end for
16:     $ct'_c \leftarrow SumSlots(ct'_b; k_i, 1)$
17:     $ct'_c \leftarrow SumSlots(ct'_c; k_i, kw_i)$
18:     $ct'_c \leftarrow SumSlots(ct'_c; t_i, k^2 h_i w_i)$
19:     $ct'_d \leftarrow ct'_d \oplus Rot(ct'_c; -\lfloor i/k_o^2 \rfloor k_o^2 h_o w_o - \lfloor (i \bmod k_o^2)/k_o \rfloor k_o w_o - (i \bmod k_o)) \odot$
        $Vec(S'^{(i)})$
20: end for
21: Return $ct'_d$
```

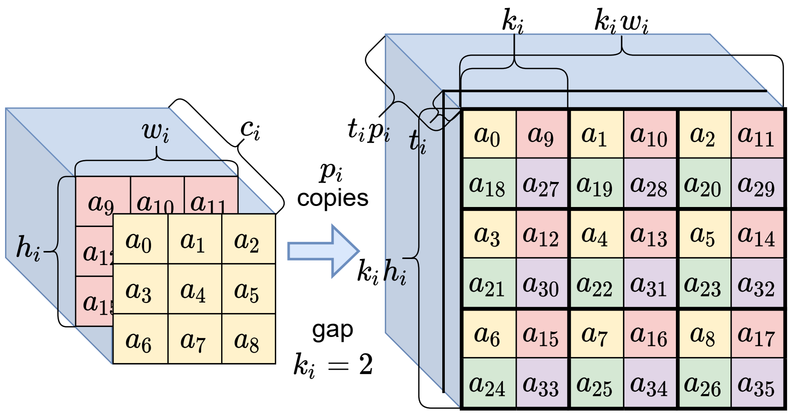

w_i
c_i
h_i
a_9 a_{10} a_{11}
a_{12}
a_{15}
a_0 a_1 a_2
a_3 a_4 a_5
a_6 a_7 a_8
p_i copies
gap
$k_i = 2$
$t_i p_i$
t_i
$k_i h_i$
k_i
$k_i w_i$
a_0 a_9 a_1 a_{10} a_2 a_{11}
a_{18} a_{27} a_{19} a_{28} a_{20} a_{29}
a_3 a_{12} a_4 a_{13} a_5 a_{14}
a_{21} a_{30} a_{22} a_{31} a_{23} a_{32}
a_6 a_{15} a_7 a_{16} a_8 a_{17}
a_{24} a_{33} a_{25} a_{34} a_{26} a_{35}

Figure 4.11: Multiplexed parallel packing method $MultParPack$ when $k_i^2 h_i w_i t_i \mid n_t$.

performs multiplication by weight and summing up c_o times, multiplexed parallel convolution algorithm $MultParConv$ performs only q times, reducing the required number of rotations to about $1/p_i$.

Before description of $MultParConv$ in detail, it is required to define $ParMultWgt(U; i_1, i_2, i_3)$ that maps weight tensor $U \in R^{h_i \times w_i \times c_i \times c_o}$ to an element of R^{n_t}. To define $ParMultWgt$, parallelly multiplexed shifted weight tensor $U''^{(i_1,i_2,i_3)} = (U''^{(i_1,i_2,i_3)}_{i_5,i_6,i_7})_{0 \le i_5 < k_i h_i, 0 \le i_6 < k_i w_i, 0 \le i_7 < t_i p_i} \in R^{k_i h_i \times k_i w_i \times t_i p_i}$ should be defined first for $0 \le i_1 < f_h, 0 \le i_2 < f_w$, and $0 \le i_3 < q$ as follows:

$$U''^{(i_1,i_2,i_3)}_{i_5,i_6,i_7} = \begin{cases} 0, & \text{if } k_i^2(i_7 \bmod t_i) + k_i(i_5 \bmod k_i) + i_6 \bmod k_i \ge c_i \\ & \text{or } \lfloor i_7/t_i \rfloor + p_i i_3 \ge c_o \\ & \text{or } \lfloor i_5/k_i \rfloor - (f_h - 1)/2 + i_1 \notin [0, h_i - 1] \\ & \text{or } \lfloor i_6/k_i \rfloor - (f_w - 1)/2 + i_2 \notin [0, w_i - 1], \\ U_{i_1, i_2, k_i^2(i_7 \bmod t_i) + k_i(i_5 \bmod k_i) + i_6 \bmod k_i, \lfloor i_7/t_i \rfloor + p_i i_3}, & \text{otherwise,} \end{cases}$$

for $0 \le i_5 < k_i h_i$, $0 \le i_6 < k_i w_i$, and $0 \le i_7 < t_i p_i$. Then, $ParMultWgt$ is defined as $ParMultWgt(U; i_1, i_2, i_3) = Vec(U''^{(i_1,i_2,i_3)})$. The multiplexed selecting tensor $S'^{(i)}$ defined in Subsection 4.3.1 is also used in MultParConv.

Then, Algorithm 20 shows the proposed multiplexed parallel convolution algorithm $MultParConv$, where $t_o = \lfloor \frac{c_o}{k_o^2} \rfloor$ and $p_o = 2^{\lfloor \log_2 (\frac{n_t}{k_o^2 h_o w_o t_o}) \rfloor}$.

Algorithm 20 $MultParConv(ct''_a, U)$

```
1: Input: Parallelly multiplexed tensor ciphertext $ct'_a$ and weight tensor $U$
2: Output: Parallelly multiplexed tensor ciphertext
3: $ct''_d \leftarrow ct_{zero}$
4: for $i_1 \leftarrow 0$ to $f_h - 1$ do
5:     for $i_2 \leftarrow 0$ to $f_w - 1$ do
6:         $ct''^{(i_1,i_2)} \leftarrow Rot(ct''_a; k_i^2 w_i (i_1 - (f_h - 1)/2) + k_i (i_2 - (f_w - 1)/2))$
7:     end for
8: end for
9: for $i_3 \leftarrow 0$ to $q - 1$ do
10:    $ct''_b \leftarrow ct_{zero}$
11:    for $i_1 \leftarrow 0$ to $f_h - 1$ do
12:        for $i_2 \leftarrow 0$ to $f_w - 1$ do
13:            $ct''_b \leftarrow ct''_b \oplus ct''^{(i_1,i_2)} \odot ParMultWgt(U; i_1, i_2, i_3)$
14:        end for
15:    end for
16:    $ct''_c \leftarrow SumSlots(ct''_b; k_i, 1)$
17:    $ct''_c \leftarrow SumSlots(ct''_c; k_i, k_i w_i)$
18:    $ct''_c \leftarrow SumSlots(ct''_c; t_i, k_i^2 h_i w_i)$
19:    for $i_4 \leftarrow 0$ to $\min(p_i - 1, c_o - 1 - p_i i_3)$ do
20:        $i \leftarrow p_i i_3 + i_4$
21:        $ct''_d \leftarrow ct''_d \oplus Rot(ct''_c; -\lfloor i/k_o^2 \rfloor k_o^2 h_o w_o + \lfloor n_t/p_i \rfloor (i \bmod p_i) - \lfloor (i \bmod k_o^2)/k_o \rfloor k_o w_o - i \bmod k_o) \odot Vec(S'^{(i)})$
22:    end for
23: end for
24: for $j \leftarrow 0$ to $\log_2 p_o - 1$ do
25:    $ct''_d \leftarrow ct''_d \oplus Rot(ct''_d; -2^j (n_t/p_o))$
26: end for
27: Return $ct''_d$
```

4.4.6 Multiplexed Parallel Batch Normalization, Downsampling, and Average Pooling

In Subsection 4.3.2, multiplexed parallel convolution algorithm is proposed, $MultParConv$ that works for an input parallelly multiplexed tensor. Besides convolution, the ResNet model has also batch normalization and average pooling. For the CIFAR-10 dataset, the ResNet model also has downsampling. Batch normalization, average pooling, and downsampling should be implemented to be also compatible with the multiplexed parallel packing method. Thus, new batch normalization, downsampling, and average pooling algorithms that work for an input ciphertext having plaintext tensor using $MultParPack$ are described in this subsection.

Multiplexed Parallel Batch Normalization

I propose an algorithm $ParMultBN$ that performs batch normalization for a given input parallelly multiplexed tensor. To this end, it is required to define new function $ParBNConst$ that maps batch normalization constant vectors $C, M, I \in R^{c_i}$ (explained in Subsection 4.4.3) to a vector in R^{n_t} properly. For a given input constant vector $H \in R^{c_i}$, $ParBNConst$ outputs a vector $h'' = (h''_0, h''_1, \cdots, h''_{n_t-1}) \in R^{n_t}$ satisfying

$$h''_j = \begin{cases} 0, & \text{if } j \bmod (n_t/p_i) \geq k_i^2 h_i w_i t_i \text{ or } k_i^2 i_3 + k_i(i_1 \bmod k_i) + i_2 \bmod k_i \geq c_i \\ H_{k_i^2 i_3 + k_i(i_1 \bmod k_i) + i_2 \bmod k_i}, & \text{otherwise,} \end{cases}$$

for $0 \leq j < n_t$, where $i_1 = \lfloor ((j \bmod (n_t/p_i)) \bmod k_i^2 h_i w_i)/k_i w_i \rfloor$, $i_2 = (j \bmod (n_t/p_i)) \bmod k_i w_i$, and $i_3 = \lfloor (j \bmod (n_t/p_i))/k_i^2 h_i w_i \rfloor$. $ParMultBN$ that performs batch normalization using this $ParBNConst$ function is proposed, and Algorithm 21 describes the proposed $ParMultBN$.

Algorithm 21 $ParMultBN(ct''_a, C, M, I)$

```
1: Input: Parallelly multiplexed tensor ciphertext $ct''_a$ and batch normalization constant vectors $C, M, I \in R_i^c$
2: Output: Parallelly multiplexed tensor ciphertext $ct''_b$
3: $c'' \leftarrow ParBNConst(C), m'' \leftarrow ParBNConst(M), i'' \leftarrow ParBNConst(I)$
4: $ct''_b \leftarrow c'' \odot ct''_a \oplus (i'' - c'' \cdot m'')$
5: Return $ct''_b$
```

Multiplexed Parallel Downsampling

ResNet models for the CIFAR-10 dataset require two downsampling processes. $DownSamp$ algorithm that performs downsampling for a given input parallelly multiplexed tensor is proposed. This prevents the density of valid values from decreasing after downsampling. To specifically describe the proposed downsampling algorithm, it is required to define downsampling selecting tensor $S''^{(i_1,i_2)} = (S''^{(i_1,i_2)}_{i_3,i_4,i_5})_{0 \le i_3 < k_i h_i, 0 \le i_4 < k_i w_i, 0 \le i_5 < t_i} \in R^{k_i h_i \times k_i w_i \times t_i}$, which is used to select $4k_i$ valid values. Downsampling selecting tensor $S''^{(i_1,i_2)} = (S''^{(i_1,i_2)}_{i_3,i_4,i_5})_{0 \le i_3 < k_i h_i, 0 \le i_4 < k_i w_i, 0 \le i_5 < t_i}$ for $0 \le i_1 < k_i$ and $0 \le i_2 < t_i$ is defined as follows:

$$S''^{(i_1,i_2)}_{i_3,i_4,i_5} = \begin{cases} 1, & \text{if } (\lfloor i_3/k_i \rfloor) \bmod 2 = 0 \\ & \text{ and } (\lfloor i_4/k_i \rfloor) \bmod 2 = 0 \\ & \text{ and } i_3 \bmod k_i = i_1 \\ & \text{ and } i_5 = i_2 \\ 0, & \text{otherwise,} \end{cases}$$

for $0 \le i_3 < k_i h_i, 0 \le i_4 < k_i w_i$, and $0 \le i_5 < t_i$. Algorithm 22 describes the proposed downsampling algorithm $DownSamp$.

Algorithm 22 $Downsamp(ct''_a)$

```
1: Input: Parallelly multiplexed tensor ciphertext $ct''_a$
2: Output: Parallelly multiplexed tensor ciphertext $ct''_c$
3: $ct''_c \leftarrow ct_{zero}$
4: for $i_1 \leftarrow 0$ to $k_i - 1$ do
5:     for $i_2 \leftarrow 0$ to $t_i - 1$ do
6:         $i_3 \leftarrow \lfloor((k_i i_2 + i_1) \bmod 2k_o)/2\rfloor$
7:         $i_4 \leftarrow (k_i i_2 + i_1) \bmod 2$
8:         $i_5 \leftarrow \lfloor(k_i i_2 + i_1)/2k_o\rfloor$
9:         $ct''_b \leftarrow ct''_a \odot Vec(S''^{(i_1,i_2)})$
10:        $ct''_c \leftarrow ct''_b \oplus Rot(ct''_b; k_i^2 h_i w_i(i_2 - i_5) + k_i w_i(i_1 - i_3) - k_i i_4)$
11:    end for
12: end for
13: $ct''_c \leftarrow Rot(ct''_c; -k_o^2 h_o w_o t_i/8)$
14: for $j \leftarrow 0$ to $\log_2 p_o - 1$ do
15:     $ct''_c \leftarrow ct''_c \oplus Rot(ct''_c; -2^j k_o^2 h_o w_o t_o)$
16: end for
17: Return $ct''_c$
```

Algorithm 23 $AvgPool(ct''_a)$

```
1: **Input:** Parallelly multiplexed tensor ciphertext $ct''_a$
2: **Output:** One-dimensional array ciphertext $ct_b$
3: $ct_b \leftarrow ct_{zero}$
4: **for** $j \leftarrow 0$ **to** $\log_2 w_i - 1$ **do**
5:     $ct''_a \leftarrow Rot(ct''_a; 2^j \cdot k_i)$
6: **end for**
7: **for** $j \leftarrow 0$ **to** $\log_2 h_i - 1$ **do**
8:     $ct''_a \leftarrow Rot(ct''_a; 2^j \cdot k_i^2 w_i)$
9: **end for**
10: **for** $i_1 \leftarrow 0$ **to** $k_i - 1$ **do**
11:     **for** $i_2 \leftarrow 0$ **to** $t_i - 1$ **do**
12:         $ct_b \leftarrow ct_b \oplus Rot(ct''_a; k_i^2 h_i w_i i_2 + k_i w_i i_1 - k_i(k_i i_2 + i_1)) \odot \bar{s}'^{(k_i i_2 + i_1)}$
13:     **end for**
14: **end for**
15: **Return** $ct_b$
```

Average Pooling

When the average pooling is reached after performing all convolutions, batch normalizations, and APRs in the ResNet model, there is a ciphertext that contains data packed using $MultParPack$. The data of ciphertext packed by this multiplexed packing method is arranged in a complex order in one dimension, which limits execution of fully connected layer. Thus, an average pooling algorithm $AvgPool$ that not only performs average pooling but also rearranges indices is proposed.

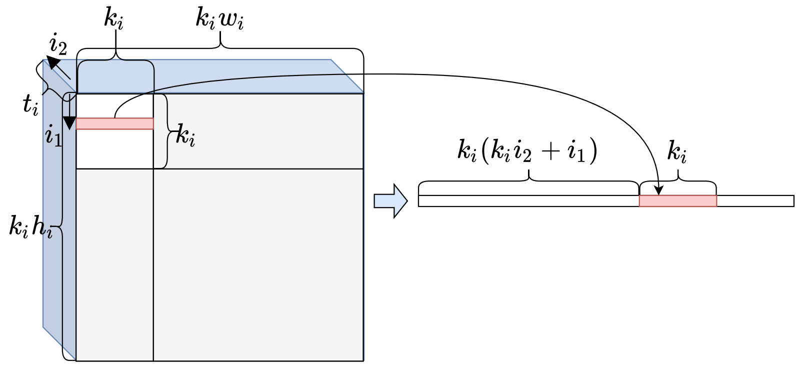


Figure 4.12: Rearranging process that selects and places $k_i^2 t_i$ valid values sequentially in AvgPool algorithm.

Average pooling is the process that obtains a vector of R^{c_i} by computing the average value for $h_i w_i$ values for an input tensor of $R^{h_i \times w_i \times c_i}$. To this end, $h_i w_i$ values can be added using rotations and additions of tensors. Dividing by $h_i w_i$ can be performed instead in the process of multiplying selecting vector. Then, in each page, only k_i^2 values are valid out of the $k_i^2 h_i w_i$ values, and the rest are the invalid garbage values. Only $k_i^2 t_i$ valid values are placed sequentially in one-dimensional vector. For this rearranging process, it is required to define selecting vector $\bar{s}'^{(i_3)} = (\bar{s}_j'^{(i_3)})_{0 \le j < n_t} \in R^n$, which is defined as follows:

$$\bar{s}_j'^{(i_3)} = \begin{cases} \frac{1}{h_i w_i}, & \text{if } j - k_i i_3 \in [0, k_i - 1] \\ 0, & \text{otherwise,} \end{cases}$$

for $0 \leq j < n_t$ and $0 \leq i_3 < k_i t_i$. Algorithm 23 shows the proposed average pooling algorithm that uses this selecting vector. Figure 23 describes the rearranging process that selects and places $k_i^2 t_i$ valid values sequentially in Algorithm 23.

4.4.7 Convolution/Batch Normalization Integration Algorithm

For a given input ciphertext ct_x, scaling processes, convolution, and batch normalization are processed by evaluating $ct_x \odot (B \cdot 1)$, $MultParConv(ct_x, U)$, $c'' \odot ct_x \oplus (i'' - c'' \cdot m'')$, and $ct_x \odot (\frac{1}{B} \cdot 1)$ functions sequentially, where 1 is all-one vector in R^n. Considering $MultParConv$ is a linear function, these operations are equivalent to evaluating

$$\begin{aligned} & (c'' \odot MultParConv(ct_x, BU) \oplus (i'' - c'' \cdot m'')) \odot (\frac{1}{B} \cdot 1) \\ = \ & c'' \odot MultParConv(ct_x, U) \oplus \frac{1}{B}(i'' - c'' \cdot m''). \end{aligned}$$

Here, if $MultParConv(ct_x, U)$ is performed while replacing the original selecting tensor $Vec(S'^{(i)})$ by $ParBNConst(C) \cdot Vec(S'^{(i)})$, $c'' \odot MultParConv(ct_x, U)$ can be performed without additional level consumption. In addition, computation of $\frac{1}{B}(i'' - c'' \cdot m'')$ requires no additional level consumption since it simply requires operations for plaintext vectors. Thus, scaling processes, convolution, and batch normalization are performed with only two level consumptions. Algorithm 24 describes the proposed convolution/batch normalization integration algorithm that uses level optimization technique.

Algorithm 24 $MultParConvBN(ct''_a, U, C, M, I)$

```
1: Input: Parallelly multiplexed tensor ciphertext $ct''_a$, weight tensor $U$, and batch normalization constant vectors $C, M, I$
2: Output: Parallelly multiplexed tensor ciphertext $ct''_d$
3: $ct''_d \leftarrow ct_{zero}$
4: for $i_1 \leftarrow 0$ to $f_h - 1$ do
5:     for $i_2 \leftarrow 0$ to $f_w - 1$ do
6:         $ct''^{(i_1,i_2)} \leftarrow Rot(ct''_a; k_i^2 w_i(i_1 - (f_h - 1)/2) + k_i(i_2 - (f_w - 1)/2))$
7:     end for
8: end for
9: for $i_3 \leftarrow 0$ to $q - 1$ do
10:    $ct''_b \leftarrow ct_{zero}$
11:    for $i_1 \leftarrow 0$ to $f_h - 1$ do
12:        for $i_2 \leftarrow 0$ to $f_w - 1$ do
13:            $ct''_b \leftarrow ct''_b \oplus ct''^{(i_1,i_2)} \odot ParMultWgt(U; i_1, i_2, i_3)$
14:        end for
15:    end for
16:    $ct''_c \leftarrow SumSlots(ct''_b; k_i, 1)$
17:    $ct''_c \leftarrow SumSlots(ct''_c; k_i, k_i w_i)$
18:    $ct''_c \leftarrow SumSlots(ct''_c; t_i, k_i^2 h_i w_i)$
19:    for $i_4 \leftarrow 0$ to $\min(p_i - 1, c_o - 1 - p_i i_3)$ do
20:        $i \leftarrow p_i i_3 + i_4$
21:        $ct''_d \leftarrow ct''_d \oplus Rot(ct''_c; -\lfloor i/k_o^2 \rfloor k_o^2 h_o w_o + \lfloor n_t/p_i \rfloor (i \bmod p_i) - \lfloor (i \bmod k_o^2)/k_o \rfloor k_o w_o - i \bmod k_o) \odot (ParBNConst(C) \cdot Vec(S'^{(i)}))$
22:    end for
23: end for
24: for $j \leftarrow 0$ to $\log_2 p_o - 1$ do
25:    $ct''_d \leftarrow ct''_d \oplus Rot(ct''_d; -2^j(n_t/p_o))$
26: end for
27: $ct''_d \leftarrow ct''_d \ominus \frac{1}{B}(c'' \cdot m'' - i'')$
28: Return $ct''_d$
```

4.5 Catastrophic Divergence from Imaginary Error and Imaginary-Removing Bootstrapping

In this section, an imaginary-removing bootstrapping is proposed, which makes it possible to implement VDSCNNs. The most sensitive component in ResNet implementation with many layers on the RNS-CKKS scheme is the APR. Since the RNS-CKKS scheme actually deals with complex numbers, precision noise during each homomorphic operation occurs not only in the real part of each data but also in the imaginary part. It can be found that the results of the APR in the real part can completely diverge if the accumulated noise in the imaginary part is not small enough.

I adopt the APR consisting of the composition of minimax approximate polynomials for piecewise sign functions [54]. Assume that p_1 and p_2 are sequential component minimax approximate polynomials in this order. If the range within the approximation domain of p_1 is $[-1-b, -1+b] \cup [1-b, 1+b]$, the approximation domain of p_2 is designed to be this range. Since the minimax approximate polynomial usually diverges when the input value is outside the approximation domain, the result value of p_2 will diverge greatly and lead to a failure of APR if the result value of p_1 is outside of $[-1-b, 1+b]$.

Consider the neighborhood of the local maximum point x_0 such that $p_1(x_0) = 1 + b$. $p_1(x)$ can be approximated by the second Taylor polynomial $T_{p_1,2}(x) = p_1(x_0) - a(x - x_0)^2$ for positive real number a near x_0, which is also valid in the complex domain. When the value of $x - x_0$ is a pure imaginary number, the value of $T_{p_1,2}(x)$ is always greater than $p_1(x_0) = 1 + b$. Thus, there exist some values of x such that $\mathrm{Re}(p_1(x))$ is outside of $[-1-b, 1+b]$ when allowing imaginary noise, which leads to a failure in the whole ResNet inference.

Hence, to stably perform ResNet with many layers, it is important to remove the imaginary part of the input of each APR. It is proposed to apply the *imaginary-removing bootstrapping* operation before the APR. The formula $\mathrm{Re}(x) = x/2 + \overline{x/2}$

is homomorphically evaluated by halving all coefficient values in SlotToCoeff operation in the bootstrapping and homomorphically computing $v + \bar{v}$. This additional operation costs only one KSO for homomorphic conjugation, and no additional level is consumed.

Figure 4.13 shows the mean of absolute values of imaginary parts after each layer using normal and imaginary-removing bootstrappings for one instance of ResNet-110 inference. It is observed that the diverging phenomenon occurs after the 69th layer due to the accumulated noise in the imaginary part. This catastrophic divergence occurs for 12 images out of 50 tested images (i.e., 24% of tested images). The proposed imaginary-removing bootstrapping makes the noise of imaginary parts remain much smaller during deeper ResNet inference, and it can be confirmed that imaginary-removing bootstrapping never causes this diverging phenomenon when conducting simulations for a various number of layers and test images as in Section 4.7. It is worth mentioning that this divergence problem of VDSCNNs on FHE and its solution are addressed for the first time.

4.6 Implementation of Privacy-Preserving ResNet Models

4.6.1 Optimization of Level Consumption

In the proposed implementation, convolution, batch normalization, bootstrapping, and APR are repeatedly performed in this order. Since the bootstrapping and APR work only for input values in $[-1, 1]$, it is required to do scaling by $1/B$ before bootstrapping and by B after the APR. Sufficiently large B is set to maintain all the computed values within $[-B, B]$. $B = 40$ and $B = 65$ are set for the CIFAR-10 and CIFAR-100 datasets, respectively, and each value of B is obtained by adding some margin to the maximum value of all used values.

I propose a method of reducing level consumption by integrating computations, as shown in Figure 4.14. The constant of batch normalization (i.e., a) is multiplied

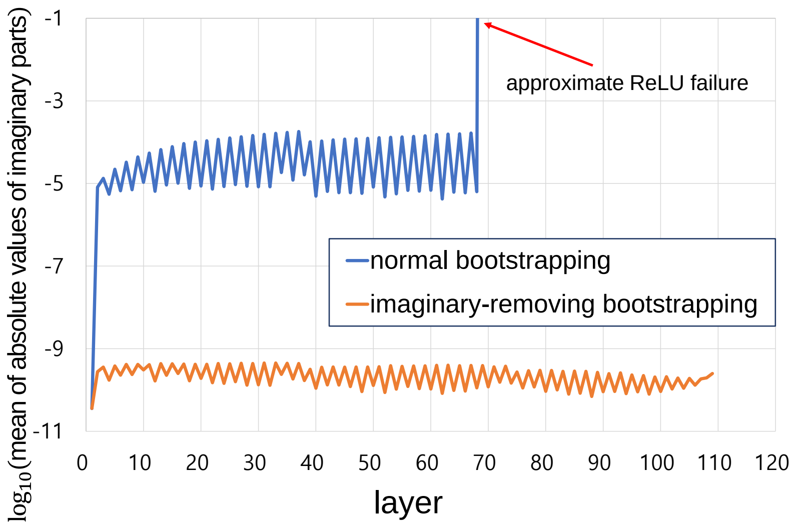


Figure 4.13: Mean of absolute values of imaginary parts after each layer when performing ResNet-110 inference using the normal bootstrapping and the proposed imaginary-removing bootstrapping.

during the selecting procedure in convolution instead of batch normalization, and then add a modified constant vector by taking into account the value of B during batch normalization. By these judicious integrations, three levels can be saved. Figure 4.14 describes this level optimization technique, and the proposed convolution/batch normalization integration algorithm, denoted as $MultParConvBN$, is presented in Subsection 4.4.7.

4.6.2 The Proposed Architecture for ResNet on the RNS-CKKS Scheme

Parameter Setting

The polynomial degree is set to $N = 2^{16}$ and the number of full slots is $n_t = 2^{15}$. Some parameters are optimized to achieve a higher security level. First, the Hamming weight of the secret key is set to 192, which is larger than 64 used in many previous works because larger Hamming weight of secret key increases available modulus bits. In addition, base modulus, special modulus, and bootstrapping modulus are set to 51-

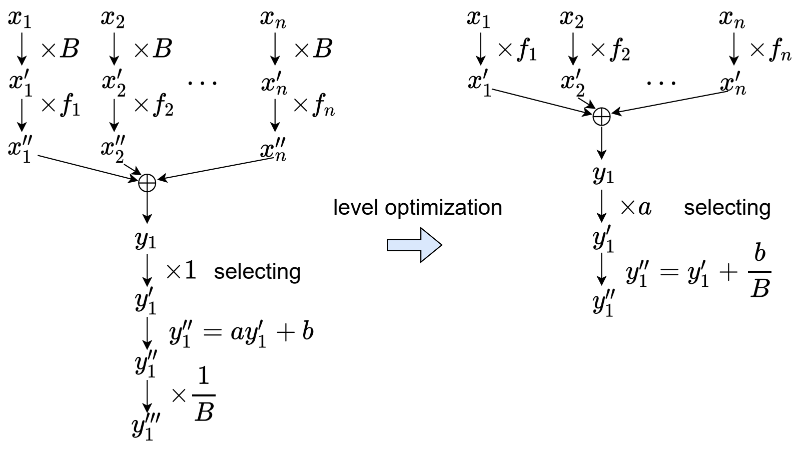


Figure 4.14: Level optimization by integrating computations.

bit prime instead of 60-bit prime, and default modulus are set to 46-bit prime instead of 50-bit prime. Even if the length of the modulus bits is reduced, high accuracy of bootstrapping or APR can be achieved. Based on the hybrid dual attack for the learning with errors (LWE) problem with the sparse secret key [22], the total modulus bit length for 128-bit security is 1,553 bits.

The RS bootstrappings with $n = 2^{14}, 2^{13}$, and 2^{12} are used since data in each input ciphertext for the bootstrapping is less than $n_t = 2^{15}$. CoeffToSlot and SlotToCoeff procedures are performed with level collapsing technique with three levels. The degrees of the approximate polynomials for the cosine function and the inverse sine function are 59 and 1, respectively, and the number of the double-angle formula is two. The total level consumption is 14 in the bootstrapping, and the total modulus consumption is 644. The imaginary-removing bootstrappings for $n = 2^{14}, 2^{13}$, and 2^{12} are refered to as $Boot14$, $Boot13$, $Boot12$, respectively.

I use the approximate homomorphic ReLU algorithm that uses APRs using a composition of minimax approximate polynomial as in [54, 57]. The precision parameter $\alpha = 13$ and set of degrees $\{15, 15, 27\}$ are used. The homomorphic ReLU algorithm for these parameters is refered to as $AppReLU(ct_x)$. The ℓ_1-norm approximation error of AppReLU is less than 2^{-13}, and this marginal error enables me to use the pre-trained parameters of standard ResNet models. That is, it is not needed to train/retrain contrary to a nonstandard HE-friendly network.

The Proposed Structure of ResNet on the RNS-CKKS Scheme

The 32×32 CIFAR-10 and CIFAR-100 images is used for the evaluation. Downsampling and average pooling algorithms that support multiplexed tensors are devised. These algorithms are refered to as $Downsamp$ and $AvgPool$, presented in Subsection 4.4.6. The fully connected layer using the diagonal method in [42] is used. ResNet-20/32/44/56/110 on the RNS-CKKS scheme are implemeneted using $MultParConvBN$, $AppReLU$, Boot, $AvgPool$, $Downsamp$, and fully connected layer. Figure 4.15 shows the proposed ResNet structure on the RNS-CKKS scheme, where $MultParConvBN$ is simply referred to as $ConvBN$. The parameters used in $ConvBN$ and $Downsamp$ are presented in Table 4.6.

While two sequential bootstrappings are required to perform APR, convolution, and batch normalization in one layer without this optimization, only single use of bootstrapping is necessary for the proposed implementation because the required level consumption for convolution, batch normalization, and bootstrapping are reduced a lot. In addition, the proposed architecture for ResNet uses a 1,501-bit modulus, and thus, it achieves the standard 128-bit security level.

4.7 Simulation Results

In this section, numerical results of the proposed architecture for ResNet are presented. The numerical analyses are conducted on the representative RNS-CKKS scheme library SEAL [68] on AMD Ryzen Threadripper PRO 3995WX at 2.096 GHz (64 cores) with 512 GB RAM, running the Ubuntu 20.04 operating system. The CIFAR-10 and CIFAR-100 datasets are employed for evaluation, which are both composed of 50,000 images for training and 10,000 images for testing [53]. Pre-trained parameters are used

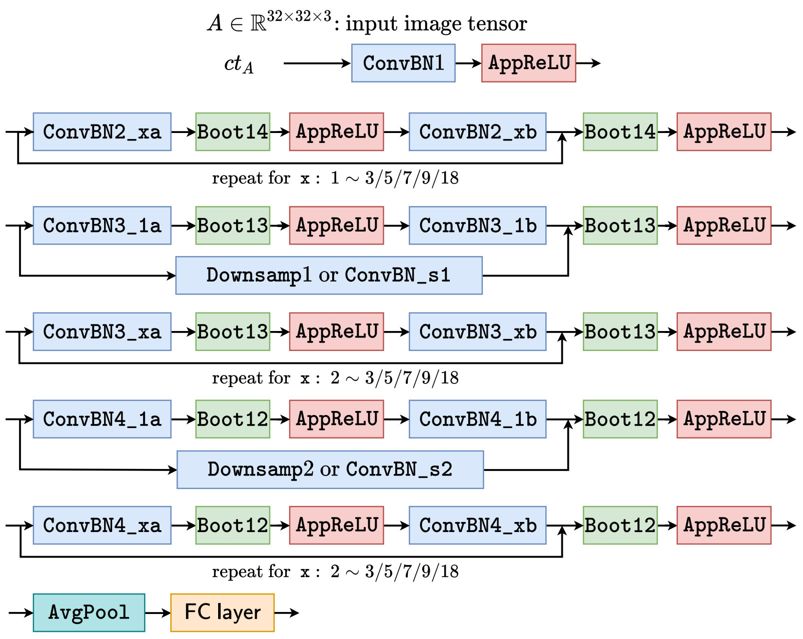


Figure 4.15: Structure of the proposed ResNet-20/32/44/56/110 on the RNS-CKKS scheme. The input image is packed in ct_A in a raster scan fasion and using RS packing.

for standard ResNet-20/32/44/56/110.

4.7.1 Latency

First, ResNet-20/32/44/56/110 are performed using the proposed architecture on the RNS-CKKS scheme. 3,306 KSOs are required for ResNet-20, which is $116\times$ smaller than 384,160 without the proposed techniques and optimizations. Table 4.7 shows the classification runtime for one CIFAR-10/CIFAR-100 image using ResNet models on the RNS-CKKS scheme. Due to the large reduction of the number of KSOs, while the previous implementation takes 10,602s with 64 CPU threads to perform ResNet-20 on the RNS-CKKS scheme, the proposed implementation takes 2,271s to perform ResNet-20 even with one CPU thread, which is $4.67\times$ reduction in latency. Considering that the proposed implementation only uses one CPU thread, more than $100\times$ and $1000\times$ lower latency can be expected on GPU and hardware accelerators, respectively

component	CIFAR-10												CIFAR-100	
	w/o mult. conv. (64 threads)		proposed (single thread)										proposed (single thread)	
	ResNet-20		ResNet-20		ResNet-32		ResNet-44		ResNet-56		ResNet-110		ResNet-32	
	runtime	percent	runtime	percent	runtime	percent	runtime	percent	runtime	percent	runtime	percent	runtime	percent
ConvBN	-	-	346s	15.2%	547s	14.7%	751s	14.3%	960s	14%	1,855s	14%	542s	13.7%
AppReLU	-	-	257s	11.3%	406s	10.9%	583s	11.2%	762s	11.1%	1,475s	11.1%	510s	12.9%
Boot	-	-	1,651s	72.6%	2,760s	74.0%	3,874s	74.1%	5,113s	74.6%	9,936s	74.8%	2,864s	72.7%
Downsamp	-	-	5s	0.2%	5s	0.1%	5s	0.09%	5s	0.07%	5s	0.04%	-	-
AvgPool	-	-	2s	0.1%	2s	0.06%	2s	0.05%	2s	0.04%	2s	0.02%	2s	0.05%
FC layer	-	-	10s	0.4%	10s	0.3%	10s	0.2%	10s	0.1%	10s	0.08%	24s	0.6%
total	10,602s	100%	2,271s	100%	3,730s	100%	5,224s	100%	6,852s	100%	13,282s	100%	3,942s	100%

Table 4.7: Classification runtime for one CIFAR-10/CIFAR-100 image using ResNet on the RNS-CKKS scheme

		model	runtime	amortized runtime
CIFAR-10	w/o mult. conv. (one image, 64 threads)	ResNet-20	10,602s	10,602s
	proposed (50 images, 50 threads)	ResNet-20	3,973s	79s
		ResNet-32	6,130s	122s
		ResNet-44	8,983s	179s
		ResNet-56	11,303s	226s
		ResNet-110	22,778s	455s
CIFAR-100	proposed (50 images, 50 threads)	ResNet-32	6,351s	127s

Table 4.8: Classification (amortized) runtime for multiple CIFAR-10/CIFAR-100 images using ResNet models on the RNS-CKKS scheme

[48, 52].

I also succeed in implementing the standard ResNet-32/44/56/110 on the RNS-CKKS scheme for the first time. Table 4.7 shows that the runtime increases *linearly* with the number of layers, which is quite difficult to be expected in leveled HEs.

4.7.2 Amortized Runtime

Since servers should classify multiple images of clients in many cases, not only the latency but also the amortized runtime for multiple images, i.e., runtime per image, is important. Since the proposed implementation requires only one thread, multiple threads allow me to classify multiple images simultaneously. Table 4.8 shows the runtime and amortized runtime of classification for multiple CIFAR-10/CIFAR-100 images using ResNet models on the RNS-CKKS scheme. The proposed implementation

dataset	model	#test images	#success	backbone accuracy	obtained accuracy
CIFAR-10	ResNet-20	10,000	9,132	91.52%	91.31%
	ResNet-32	10,000	9,240	92.49%	92.4%
	ResNet-44	2,000	1,852	92.76%	92.6%*
	ResNet-56	2,000	1,856	93.27%	92.8%*
	ResNet-110	2,000	1,858	93.5%	92.9%*
CIFAR-100	ResNet-32	10,000	6,943	69.5%	69.43%

Table 4.9: Classification accuracies for CIFAR-10/CIFAR-100 images using ResNet models on the RNS-CKKS scheme. An asterisk (*) implies that not all 10,000 test images have been tested

of ResNet-20 takes 3,973s to classify 50 images using 50 threads, which corresponds to amortized runtime 79s. This is $134\times$ faster than the amortized runtime 10,602s without proposed techniques and optimizations.

4.7.3 Accuracy

Table 4.9 presents the classification accuracies for CIFAR-10/CIFAR-100 images using ResNet models on the RNS-CKKS scheme. Thanks to resolving the catastrophic divergence phenomenon by the proposed imaginary-removing bootstrapping, all the obtained accuracies for ResNet-20/32/44/56/110 are very close to those of backbone CNNs. This implies that the proposed implementation of VDSCNNs on the RNS-CKKS scheme can benefit from high accuracies of various pre-trained VDSCNNs that have widely been developed already.

Chapter 5

HIERARCHICAL GALOIS KEY GENERATION

Among various FHE schemes, Brakerski/Fan-Vercauteran (BFV) [12, 33] and Cheon-Kim-Kim-Song (CKKS) [19, 21] schemes are two of the most practical FHE schemes. They can support arithmetic operations for integer or complex numbers in the single-instruction multiple-data (SIMD) manner. Thus, several data can be encrypted in one ciphertext, and one homomorphic operation can simultaneously perform component-wise operations on these multiple message data. The BFV scheme deals with integer data and supports exact computation on the encrypted integer data, and it fits the situation requiring exact computation. On the other hand, since the CKKS scheme deals with real or complex number data and supports approximate computation on the encrypted real or complex data, it fits the situation allowing approximate computation.

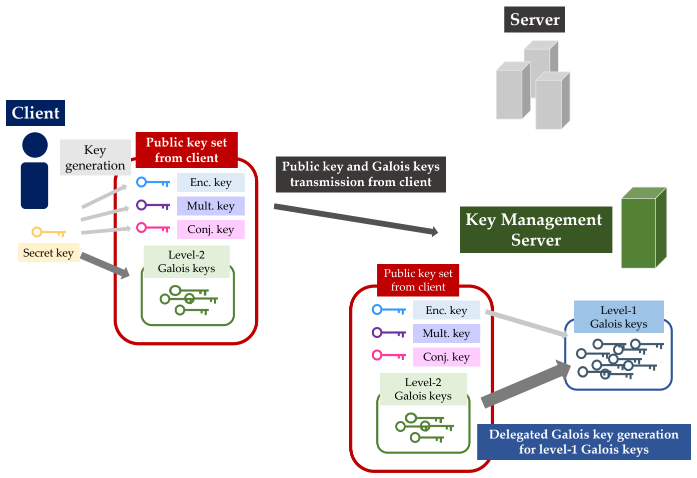
The BFV and CKKS schemes also support rotation operation which corresponds to a cyclic shift of message data within ciphertext. Many applications that require important operations such as bootstrapping, matrix multiplication, and convolution in convolutional neural networks can be achieved using this rotation. However, one of the main obstacles for using these applications using homomorphic encryption in the client-server system is heavy Galois keys. The Galois keys is evaluation keys for the homomorphic rotation operation, which is the cyclic shift operations for rows of the encrypted matrix in one ciphertext of the BFV scheme and for encrypted message vector in that of the CKKS scheme. The homomorphic rotation operation is inevitable if it is required to operate data with different positions in one ciphertext, such as the bootstrapping [8, 16, 20, 56, 58], the matrix multiplication [47], and the convolution in convolutional neural networks [49]. Since different Galois keys are required for all the different cyclic shift values for the homomorphic rotation operation, the number and the total size of Galois keys can be significantly large for the complex computational model. For example, if the standard ResNet-20 network for the CIFAR-10 dataset is implemented with pre-trained parameters with the CKKS scheme with the polynomial modulus degree $N = 2^{16}$, the server requires 265 Galois keys, which occupies 105.6GB of memory in the server. If the ResNet-18 network for the ImageNet dataset is designed using the same techniques, 617 Galois keys are required and it occupies 197.6GB of memory in the server.

In conventional homomorphic encryption schemes, clients with secret keys had to generate Galois keys for all necessary cyclic shift values, and thus lots of Galois keys impose a heavy burden on both servers and clients. First, clients do not have large computational resources in general, and thus requiring the clients to generate all of these Galois keys imposes a substantial computational burden on the clients. In addition, considerable communication amount between the client and the server is required because the client should send all generated Galois keys to the server.

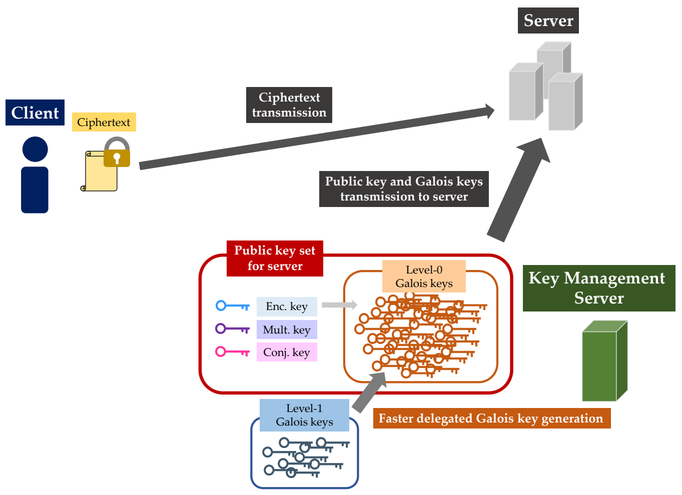
On the other hand, the server may not want to release the information of the required subset of Galois keys for the requested services because it can leak some information about the computation model of the server. Further, since a great deal of memory is used to store Galois keys of clients, the server dealing with a large number of clients requires lots of memory resources only to keep their Galois keys. The server may want to efficiently use memory resources by temporarily removing some inert Galois keys according to the services required by the client. Since the server does not have a secret key and permission to generate the Galois keys, it should ask clients to generate and send the required Galois keys to the server again if the server needs them for new services requested by the client. Otherwise, the server should store all the Galois keys received from the clients, which prevents the server from using memories efficiently. Therefore, a new Galois key generation scheme needs to be developed, enabling flexible management of the Galois keys in the client-server systems.

In the BFV and CKKS schemes, it is observed that the Galois keys can be generated from other Galois keys using key-switching operation. The crucial observation is that the Galois key can be regarded as a set of ciphertexts. If the key-switching operation is performed to each ciphertext in a Galois key, new Galois key for other cyclic shift can be derived. Since the key-switching operation requires a key-switching key with larger modulus (i.e., in the higher key level) than the ciphertext, the key-switching key for this Galois key generation should have higher level than the newly generated Galois key. This high-level key-switching key is also in the form of the Galois key, and thus it can also be generated by another higher-level key-switching key. Thus, a chain of Galois keys for various levels can be defined, where each Galois key may be used as a key-switching key for generating a lower-level Galois key.

From the above observations, a hierarchical Galois key generation system is proposed, which makes it possible to generate a lower-level Galois key using higher-level Galois key in the server. In this system, clients can generate only a small set of the highest-level Galois keys such as Galois keys for only power-of-2 cyclic shifts. Then they send the small set of Galois keys to the key management server (KMS) or the server. The server can generate a large set of lower-level Galois keys using the received set of Galois keys without any help from the clients and finally, a set of level-zero Galois keys is generated, which corresponds to the set of conventional Galois keys for the cyclic shifts of message data within a ciphertext in the server. In the server, inert Galois keys can be temporarily removed and re-generated only when needed to efficiently manage the storage of Galois keys. This proposed method can significantly reduce the computational burdens of the client, the communication cost between the client and server, and storage cost of all Galois keys in the server. To further optimize



(a) Public key and level-2 Galois key tranmission from the client and preparation for faster level-0 Galois key generation by generating level-1 Galois keys in advance



(b) Faster Galois key generation from public key and level-1 Galois keys

Figure 5.1: Efficient Galois key management in three-level hierarchical Galois key generation.

this Galois key generation, several optimization techniques also proposed, such as the hoisted Galois key generation and the reduction to graph-theoretic algorithms.

A general protocol capable of efficient Galois key management is presented, reflecting the activity of the clients using multi-level Galois key generation scheme. When a client frequently uses the service, it is important to generate the desired Galois keys quickly so that the service should not be delayed due to the Galois key generation. On the other hand, in the case of clients who do not use the service frequently, it may be better to store only the minimal Galois key set and reserve memory in the server for other services to active users. However, it is also required to prepare the inert client to become an active client at any time. In Figure 5.1 of three-level Galois key generation scheme, the client generates and transmits the minimum number of the level-2 Galois keys, and the server generates and retains an appropriate number of the level-1 Galois keys from the level-2 Galois keys reflecting how often the client uses services, where server and KMS can be collocated. With these level-1 Galois keys, the server can generate the level-0 Galois keys more efficiently. The role of the level-1 Galois keys is to give a trade-off between the efficiency of generating the level-0 Galois keys when requested and the memory used for storing Galois keys, and these level-1 Galois keys can be updated only by the server without any help from the client. The proposed protocol can enable this fine key management system to adjust in detail the trade-off between the memory usage of the Galois keys and the computational complexity of Galois key generation in the clients and the server.

The simulation is conducted with the proposed Galois key generation system for ResNet models with an appropriate computing environment for the client-server model. If a three-level hierarchical Galois key system is used, the Galois key size generated and transmitted by the client can be reduced from 105.6GB for 265 Galois keys to 3.4GB for 8 Galois keys for the ResNet-20 for CIFAR-10, and reduced from 197.6GB for 617 Galois keys to 3.9GB for 8 Galois keys for the ResNet-18 for ImageNet. While the generation of Galois keys for the ResNet-20 and the ResNet-18 by the client takes 368.5s and 786.0s in the conventional system, it is reduced to 12.1s (30×) and 15.7s (50×) in the three-level hierarchical Galois key system, respectively. The server with GPU accelerator only needs 25.3s and 22.0s to generate all required Galois keys in the online phase.

5.1 Hierarchical Galois Key System

In this section, an overview of the proposed hierarchical Galois key system are provided. Specific procedures in this system will be described in Sections 5.2 and 5.3.

5.1.1 Definition of Hierarchical Galois Key System

The hierarchical Galois key system is defined in the cloud computing using FHE. In a k-level hierarchical Galois key system, there are k sets of Galois keys with a hierarchy from a key level $k-1$ to 0, where the conventional Galois key corresponds to the Galois key in the key level 0 with $k = 1$. Each Galois key can be used to generate Galois keys in the lower levels. The additional algorithms for the hierarchical Galois key system are $\mathsf{InitGalKeyGen}$ and $\mathsf{GalKeyGen}$. The algorithm $\mathsf{InitGalKeyGen}$ generates a set of Galois keys in the highest key level using the secret key, which is performed by the client who has the secret key. The algorithm $\mathsf{GalKeyGen}$ generates a set of Galois keys in the intermediate key levels or the zero key level using the public key and the set of Galois keys in the higher key level. This algorithm is performed by the server or the key management server (KMS) having no secret key. Now, it is assumed that the public key and hierarchical Galois keys are managed by the KMS collocated with or separated from the server, and all protocols in the dissertation also make sense when the KMS and the server are united. These two algorithms are defined as follows, where k denotes the total number of key levels for the hierarchical Galois key system.

- $\mathsf{InitGalKeyGen}(s, \mathcal{T}_{k-1}) \to \{gk_i^{(k-1)}\}_{i \in \mathcal{T}_{k-1}}$: Given a secret key s and a set of cyclic shifts $\mathcal{T}_{k-1}$, generate the Galois keys with cyclic shifts in $\mathcal{T}_{k-1}$ in the highest key level in the client.

- $\mathsf{GalKeyGen}(\ell, \mathcal{U}_\ell, \{gk_i^{(\ell_i)}\}_{i \in \mathcal{U}_\ell}, pk, \mathcal{T}_\ell) \to \{gk_i^{(\ell)}\}_{i \in \mathcal{T}_\ell}$: Given a public key pk, a set of the Galois keys $\{gk_i^{(\ell_i)}\}_{i \in \mathcal{U}_\ell}$ with cyclic shifts in $\mathcal{U}_\ell$ in the key level ℓ_i higher that ℓ, and a set of cyclic shifts $\mathcal{T}_\ell$, generate the Galois keys with cyclic shifts in $\mathcal{T}_\ell$ in key level ℓ in the KMS.

The Galois key $gk_i^{(\ell)}$ denotes the Galois key for the cyclic shift i in the message vector in the key level ℓ, whose specific definition will be dealt with in Section 5.2. Although the public key pk is represented separately from the Galois keys, the Galois keys are also public in that these keys can open to the public. The set of cyclic shifts for each key level, which is an integer set, is denoted by $\mathcal{T}_0, \cdots, \mathcal{T}_{k-1}$, respectively. These sets are pairwisely disjoint. The set of cyclic shifts for each key level higher than ℓ whose Galois keys are generated in advance, is denoted by $\mathcal{U}_\ell$. If all desired Galois keys in the key level higher than ℓ are all generated, $\mathcal{U}_\ell$ equals to $\bigcup_{i=\ell+1}^{k-1} \mathcal{T}_i$. The conventional Galois key system can be seen as a special case of the proposed hierarchical Galois key system, where there is only the algorithm $\mathsf{InitGalKeyGen}$, and the number of key levels in the hierarchy is one.

5.1.2 Galois Key Generation Protocol in Hierarchical Galois Key System

A detailed Galois key generation protocol is proposed with a general hierarchical Galois key system. In this system, the server or the key management server can finely control the trade-off between the memory usage and the latency of the Galois key generation for required services according to how often the client uses the services. If the client requests the service more often, the server wants to provide the service to these types of clients as fast as possible and is willing to use more memory for it. To this end, the required Galois keys should be generated fast with a reduced computation amount just after the required service is determined from the request of the client. The more Galois keys in the key levels higher than zero, the smaller computation amount to generate the level-zero Galois keys in the server for specific services, but more memory is required in the server. In the environment of the limited computational resource in the client and the limited memory resource in the server, it is required to finely manage this trade-off for the Galois keys.

It is assumed that it is not known when and what model the client will request the service to the server after the key generation and transmission. The *offline phase* is defined as the generation of Galois key set in the key level $k-1$ in the client and intermediate Galois key sets of the key levels $k-2, \cdots, 2, 1$ in the server before determination of required services, and *online phase* as the generation of the level-zero Galois keys required for the service requested by the client. The specific protocols are described in Algorithm 25.

5.2 Proposed Hierarchical Galois Key Generation for BFV and CKKS Schemes

In this section, the hierarchical Galois key system for BFV and CKKS schemes is proposed. The BFV and CKKS schemes differ only in the packing structure, the decryption method, and the role of each operation for the encrypted data, but the key-switching operation itself is completely the same. Thus, I will deal with them at once.

The term *ciphertext* is used as a pair of ring elements $(b, a) \in R_q^2$ for some modulus q. A ciphertext $(b, a) \in R_q^2$ is defined to be a valid ciphertext of m with the secret key s if $b + a \cdot s = m + e \mod q$, where e is a polynomial with small coefficients compared to q.

Let $Q = \prod_{i=0}^{\mathtt{dnum}-1} Q_i$ be a product of several coprime positive integers Q_i's, and P be a positive integer which is coprime to and larger than Q_i's. A *Galois key* $\mathsf{gk}_r = \{\mathsf{gk}_{r,i}\}_{i=0,\cdots,\mathtt{dnum}-1}$ for cyclic shift r with the secret key polynomial $s \in R$ is defined to be valid if each $\mathsf{gk}_{r,i} = (b_{r,i}, a_{r,i}) \in R_{PQ}^2$ is a valid ciphertext of $P \cdot \hat{Q}_i \cdot [\hat{Q}_i^{-1}]_{Q_i} \cdot s(X^{5^r})$ with the secret key s, where $\hat{Q}_i = \prod_{j \neq i} Q_j$. This can be used for the key-

Algorithm 25 Key Management of Hierarchical Galois Key System with the k Key Levels

```
Input: Encryption parameters $params$ for $k$-level Galois key system (client and
    server), a set of cyclic shifts for Galois keys in the highest key level $\mathcal{T}_{k-1}$
    (client), sets of cyclic shifts for Galois keys in the intermediate key levels
    $\mathcal{T}_{k-2}, \cdots, \mathcal{T}_1$ (server), and a homomorphic service $\mathcal{S}$ (server)
Output: A set of Galois keys $\{gk_i^{(0)}\}_{i \in \mathcal{T}_0}$ (server)
Key generation and transmission in client
  1. $sk \leftarrow \mathsf{SecGen}(1^\lambda, params)$
  2. $pk \leftarrow \mathsf{PubGen}(sk)$
  3. $\{gk_i^{(k-1)}\}_{i \in \mathcal{T}_{k-1}} \leftarrow \mathsf{InitGalKeyGen}(s, \mathcal{T}_{k-1})$
  4. Transmit $(pk, \{gk_i^{(k-1)}\}_{i \in \mathcal{T}_{k-1}})$ to the server and let $\mathcal{G} = \{gk_i^{(k-1)}\}_{i \in \mathcal{T}_{k-1}}$
Offline phase: generating Galois keys in the key level $\ell$ for frequent users
  1. $\{gk_i^{(\ell)}\}_{i \in \mathcal{T}_\ell} \leftarrow \mathsf{GalKeyGen}(\ell, \mathcal{U}_\ell, \{gk_i^{(\ell_i)}\}_{i \in \mathcal{U}_\ell}, pk, \mathcal{T}_\ell)$
  2. $\mathcal{G} \leftarrow \mathcal{G} \cup \{gk_i^{(\ell)} : i \in \mathcal{T}_\ell\}$
Offline phase: removal of Galois keys in the key level lower than $\ell$ for non-frequent user
  1. $\mathcal{G} \leftarrow \{gk_i^{(\ell_i)} \in \mathcal{G} : i \in \bigcup_{j=\ell}^{k-1} \mathcal{T}_j, \ell_i \geq \ell\}$
Online phase: Galois key generation in server
  1. $\mathcal{T}_0 \leftarrow \mathsf{ExtractGalSet}(\mathcal{S})$
  2. $\{gk_i^{(0)}\}_{i \in \mathcal{T}_0} \leftarrow \mathsf{GalKeyGen}(0, \mathcal{U}_\ell, \{gk_i^{(1)}\}_{i \in \mathcal{T}_\ell}, pk, \mathcal{T}_0)$
```

switching operation to the ciphertext in the modulus q, where q is a divisor of Q. Q is called the evaluation modulus and P the special modulus.

These Galois keys are used in the rotation operation. The rotation operation in the BFV scheme is an operation mapping $(v_{i,j}) \mapsto (v_{i,(j+r)})$ while encrypted, where the addition operation of the subscript is in modulo $N/2$ and N is the polynomial modulus degree. The rotation operation of the CKKS scheme is an operation mapping $(v_i) \mapsto (v_{i+r})$ while encrypted. In terms of ring elements, these operations can be unified as operations mapping $m(X) \mapsto m(X^{5^r})$. For these operations, an operation of $(b(X), a(X)) \mapsto (b(X^{5^r}), a(X^{5^r}))$ is first performed. This processed ciphertext satisfies $b(X^{5^r}) + a(X^{5^r}) \cdot s(X^{5^r}) \approx m(X^{5^r})$, which means that it is a ciphertext of a plaintext polynomial $m(X^{5^r})$ with the secret key $s(X^{5^r})$. it is required to convert this ciphertext to a ciphertext of the same plaintext with the original secret key. This is done by taking the key-switching operation from $s(X^{5^r})$ to $s(X)$ using Galois key for cyclic shift r.

5.2.1 Hierarchical Special Modulus

The crucial idea in the proposed hierarchical Galois key is to apply the key-switching algorithm to the public key or the existing Galois keys in the corresponding key level, and the key-switching keys that are needed for this key-switching algorithm are the higher-level Galois keys. Since the key-switching operation requires a key-switching key with a larger modulus than that of ciphertexts, the larger modulus for the Galois key is set in the higher key level than that for the public key or the Galois keys to be key-switched.

Let Q_0 be the maximum modulus for the ciphertext, and let P_ℓ be the additional modulus for the Galois keys in the key level ℓ compared to that in the key level $\ell - 1$, which is called the *hierarchical special modulus for the key level* ℓ. The modulus P_ℓ is regarded as a special modulus in the key level ℓ, but it is regarded as a divisor of the evaluation modulus in the higher key level than ℓ. The modulus for the level-ℓ

Galois keys is $P_\ell Q_\ell$, where $Q_\ell = Q_0 \prod_{i=0}^{\ell-1} P_i$. The conventional special modulus corresponds to P_0, which is the hierarchical special modulus for the key level 0. Each hierarchical special modulus can be set independently from each other. $\texttt{hdnum}_\ell$ is defined as the number of the RNS moduli in Q_ℓ decomposed for the key-switching operation in the key level ℓ. P_ℓ is chosen to be larger than all the $\texttt{hdnum}_\ell$ decomposed moduli in Q_ℓ.

In the previous schemes, the number of RNS modulus is equally decomposed with dnum regardless of the size of each RNS modulus. However, in the CKKS scheme, the size of each RNS modulus is different from each other according to the required precision for each level, and the special modulus is not ensured to be a minimum size. Since the size of the special modulus can affect the security level of the scheme, it is desirable to minimize the size of the special modulus. An algorithm is proposed for obtaining the list of special modulus for the Galois key in each key level in Algorithms 26 and 27.

Given RNS moduli for ciphertexts and the decomposition numbers for each key level $\texttt{hdnum}_i$, the HModulusSelection algorithm chooses the set of RNS moduli for each hierarchical special modulus. The ModulusSelection algorithm is designed to minimize the bit-length of each special modulus, and it is used as a subroutine algorithm in the HModulusSelection algorithm as in Algorithm 27. The value maxmod is the maximum bit-length for each RNS primes, and this value is usually 60 for 64-bit computing system. The optimization for each modulus is meaningful for the security because the modulus for the highest-level Galois keys directly determines the security level. The RNS moduli for ciphertexts in each level are determined from the requested services by the client, in particular from the required precision for homomorphic multiplication in each level. This algorithm is also desirable to be used for the conventional FHE scheme using the special modulus without the hierarchical Galois key scheme. The correctness of the algorithm is formalized in the following theorem, which will be proved in Section 5.5.

Theorem 5.2.1. *Algorithm 26 outputs the list of indices* $I = \{u_0 = 0, u_1, \cdots, u_{\texttt{dnum}-1}\}$ *such that* $0 = u_0 < u_1 < \cdots < u_{\texttt{dnum}-1} \leq L$ *and minimizes the following formula*

$$\max\left\{\sum_{i=u_0}^{u_1-1} \log q_i, \sum_{i=u_1}^{u_2-1} \log q_i, \cdots, \sum_{i=u_{\texttt{dnum}-1}}^{L} \log q_i\right\}.$$

The notation for the modulus is set as follows. The list of moduli for ciphertexts is denoted as $\mathcal{C} = \{q_0, \cdots, q_L\}$, where L is the maximum level of the ciphertext. The additional list of moduli for the key level ℓ is denoted as $\mathcal{B}_\ell$ and its elements are denoted as $\{q_{L_{\ell-1}+1}, \cdots, q_{L_\ell}\}$, where L_ℓ is the number of extended levels in $\mathcal{C} \cup \bigcup_{j=0}^{\ell} \mathcal{B}_j$. The list of moduli for $\mathcal{C} \cup \bigcup_{j=0}^{\ell} \mathcal{B}_j$ can be denoted as $\{q_0, \cdots, q_{L_\ell}\}$. Let $I_\ell = \{u_{\ell,0} = 0, u_{\ell,1}, \cdots, u_{\ell,\texttt{hdnum}_\ell-1}\}$ be the list of the indices of boundary position derived by $\mathsf{HModulusSelection}$. Then, $Q_{\ell,i}$ denotes $\prod_{t=u_{\ell,i}}^{u_{\ell,i+1}-1} q_t$ for $0 \leq \ell \leq k-1$ and $0 \leq i \leq \texttt{hdnum}_\ell - 1$ and $\hat{Q}_{\ell,i}$ denotes $\prod_{j \neq i} Q_{\ell,j}$.

5.2.2 Generation of Public Key and Galois Keys in Client

The conventional schemes generate a public key (b, a) with the modulus $Q = \prod_{i=0}^{L} q_i$ because the special modulus is only used in the key-switching operation. In contrast, the proposed hierarchical Galois key generation scheme generates a public key (b, a) with $Q_{k-1} = \prod_{i=0}^{L_{k-2}} q_i$ to prepare to use it to generate Galois keys with key levels smaller than $k-1$. The Galois keys with the highest key level are generated by the client. The set of cyclic shifts $\mathcal{U}_\ell$ of Galois keys in the key level higher than ℓ should be the set that can generate all cyclic shifts $\mathcal{T}_\ell$ of Galois keys with the key level ℓ by the sum allowing repetition. The small size of $\mathcal{T}_{k-1}$ for the highest key level can reduce the computational burden and the communication cost of the client.

In the $\mathsf{InitGalKeyGen}$ operation for the proposed scheme, a single highest-level Galois key for cyclic shift r with the secret key polynomial $s \in R$ is the form of $\mathsf{gk}_r^{(k-1)} = \{\mathsf{gk}_{r,i}^{(k-1)}\}_{i=0,\cdots,\texttt{hdnum}_{k-1}-1}$, where $\mathsf{gk}_{r,i}^{(k-1)} = (b_{r,i}^{(k-1)}, a_{r,i}^{(k-1)}) \in R_{Q_{k-1}P_{k-1}}^2$ such that

Algorithm 26 ModulusSelection

```
Input: A list of modulus $\{q_0, \cdots, q_L\}$, decomposition number $\texttt{dnum}$
Output: Log of minimum special modulus $\log P$, the list of boundary indices $I = \{u_0 = 0, u_1, \cdots, u_{\texttt{dnum}-1}\}$
if $\texttt{dnum} = 1$ then
    return $\sum_{i=0}^{L} \log q_i$ and $\{0\}$
else
    Find $j$ minimizing the value $\left|\sum_{i=0}^{j} \log q_i - \frac{\texttt{dnum}-1}{\texttt{dnum}} \sum_{i=0}^{L} \log q_i\right|$.
    Perform ModulusSelection with $\{q_0, \cdots, q_j\}$ and $\texttt{dnum} - 1$ to output $u_j$ and $I_j$.
    $v_j \leftarrow \sum_{i=j+1}^{L} \log q_i$
    if $u_j = v_j$ then return $u_j$ and $I_j \cup \{j+1\}$ ;
    else if $u_j > v_j$ then
        while $u_j \geq v_j$ do
            $j \leftarrow j - 1$
            Perform ModulusSelection with $\{q_0, \cdots, q_j\}$ and $\texttt{dnum} - 1$ to output $u_j$ and $I_j$.
            $v_j \leftarrow \sum_{i=j+1}^{L} \log q_i$
        if $u_{j+1} > v_j$ then return $v_j$ and $I_j \cup \{j+1\}$ ;
        else return $u_{j+1}$ and $I_{j+1} \cup \{j+2\}$ ;
    else
        while $u_j \leq v_j$ do
            $j \leftarrow j + 1$
            Perform ModulusSelection for $\{q_0, \cdots, q_j\}$ and $\texttt{dnum} - 1$ to output $u_j$ and $I_j$.
            $v_j \leftarrow \sum_{i=j+1}^{L} \log q_i$
        if $u_j > v_{j-1}$ then return $v_{j-1}$ and $I_{j-1} \cup \{j\}$ ;
        else return $u_j$ and $I_j \cup \{j+1\}$ ;
```

Algorithm 27 HModulusSelection

```
Input: A list of modulus for ciphertexts $\mathcal{C} = \{q_0, \cdots, q_L\}$, hierarchical decomposition number $\texttt{hdnum}_\ell$ for each $\ell$, $0 \le \ell \le k-1$, the maximum bit-length of RNS modulus $\texttt{maxmod}$
Output: The lists of hierarchical special modulus $\mathcal{B}_\ell$ for each $\ell$, $0 \le \ell \le k-1$ and the lists of boundary indices $I_\ell$ for each $\ell$, $0 \le \ell \le k-1$
for $\ell = 0$ to $k-1$ do
    Perform ModulusSelection for $\mathcal{C} \cup \bigcup_{j=0}^{\ell-1} \mathcal{B}_j$ and $\texttt{hdnum}_\ell$ to output $\log P_\ell$ and $I_\ell$.
    $m_\ell \leftarrow \left\lceil \frac{\log P_\ell}{\texttt{maxmod}} \right\rceil$
    Sample $m_\ell$ primes with a bit-length of $\log P_\ell / m_\ell$ and insert them to $\mathcal{B}_\ell$.
```

$a_{r,i}^{(k-1)} \leftarrow R_{Q_{k-1}P_{k-1}}$ and $b_{r,i}^{(k-1)} = -a_{r,i}^{(k-1)} \cdot s + e_{r,i}^{(k-1)} + P_{k-1} \cdot \hat{Q}_{k-1,i} \cdot [\hat{Q}_{k-1,i}^{-1}]_{Q_{k-1,i}} \cdot s(X^{5^r})$ for $e_{r,i}^{(k-1)} \leftarrow \chi$. The RNS bases for $\mathsf{gk}_{r,i}$ are $\mathcal{C} \cup \bigcup_{j=0}^{k-1} \mathcal{B}_j$. Note that the distribution and the form of the Galois keys generated by the client is the same as those in the conventional Galois key generation.

5.2.3 GalToGal and PubToGal Operations

Two types of operations are required to make the level-ℓ Galois keys for ℓ less than $k-1$. One is the operation PubToGal, which generates a level-ℓ Galois key from the public key, and the other is the operation GalToGal, which generates a level-ℓ Galois key from the existing level-ℓ Galois keys for the other cyclic shifts. The combination of PubToGal operation and GalToGal operation will generate all Galois keys with only the public key and Galois keys in the key level higher than ℓ.

Let the shift-r Galois key be defined as the Galois key for cyclic shift r, and let (r, ℓ) Galois key be defined as the Galois key for cyclic shift r in the key level ℓ. For the convenience of explanation, the operation GalToGal will be first explained. The operation GalToGal is an operation that generates a $(r + r', \ell)$ Galois key from a (r, ℓ) Galois key in the key level ℓ with a shift-r' Galois key in the key level higher than ℓ. To understand this operation, keep in mind that rotation operation is a map $m(X) \mapsto$

$m(X^{5^r})$ from the perspective of the plaintext polynomial. In other words, the rotation operation can be seen as an operation that generates a ciphertext of $m(X^{5^r})$ from a ciphertext of $m(X)$ [19]. It is noted that the Galois key for cyclic shift r is a set of ciphertexts $\mathsf{gk}_r^{(\ell)} = \{\mathsf{gk}_{r,i}^{(\ell)}\}_{i=0,\cdots,\mathtt{hdnum}_\ell-1}$, where $\mathsf{gk}_{r,i}^{(\ell)} = (b_{r,i}^{(\ell)}, a_{r,i}^{(\ell)}) \in R_{Q_\ell P_\ell}^2$ and $b_{r,i}^{(\ell)} = -a_{r,i}^{(\ell)} \cdot s + e_{r,i}^{(\ell)} + P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}} \cdot s(X^{5^r})$. Each $\mathsf{gk}_{r,i}^{(\ell)}$ is a ciphertext of $P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}} \cdot s(X^{5^r})$. If the rotation operation is performed with cyclic shift r' on $\mathsf{gk}_{r,i}^{(\ell)}$, the output is a ciphertext of the following polynomial,

$$\begin{aligned} & P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}} \cdot s((X^{5^{r'}})^{5^r}) \\ = & P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}} \cdot s(X^{5^{r+r'}}). \end{aligned}$$

This rotation operation requires an (r', ℓ') Galois key $\mathsf{gk}_{r'}^{(\ell')}$, where ℓ' higher than ℓ. If this output is defined as $\mathsf{gk}_{r+r',i}^{(\ell)}$, the set $\mathsf{gk}_{r+r'}^{(\ell)} = \{\mathsf{gk}_{r+r',i}^{(\ell)}\}_{i=0,\cdots,\mathtt{hdnum}_\ell-1}$ is a valid $(r+r', \ell)$ Galois key.

This operation is called $\mathsf{GalToGal}$, as shown in Algorithm 28. The following theorem shows the correctness of $\mathsf{GalToGal}$ operation, which will be proved in Section 5.5.

Theorem 5.2.2. *The output of Algorithm 28 is a valid Galois key for the rotation operation for cyclic shift* $r + r'$.

Next, the operation $\mathsf{PubToGal}$ will be described. Note that the above operation is useful only when some Galois keys exist. However, since the server does not receive any Galois keys in the key level lower than $k-1$ from the client, the Galois key should be generated first with the public key and Galois keys in the higher levels in the server. To this end, a formal shift-0 Galois key can be thought of. If a shift-0 Galois key can be generated from a public key, a shift-r' Galois key can be generated by adding a $\mathsf{GalToGal}$ operation to the shift-0 Galois key for cyclic shift r'. By definition, the shift-0 Galois key should be the form of $\mathsf{gk}_0^{(\ell)} = \{\mathsf{gk}_{0,i}^{(\ell)}\}_{i=0,\cdots,\mathtt{hdnum}_\ell-1}$, where $\mathsf{gk}_{0,i}^{(\ell)} = (b_{0,i}^{(\ell)}, a_{0,i}^{(\ell)}) \in R_{Q_\ell P_\ell}^2$ and $b_{0,i}^{(\ell)} = -a_{0,i}^{(\ell)} \cdot s + e_{0,i}^{(\ell)} + P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}} \cdot s$.

To generate $\mathsf{gk}_{0,i}^{(\ell)}$ from the public key $(b, a) \in R_{Q_{k-1}}^2$, the public key is reduced to $(b', a') = (b \mod Q_\ell P_\ell, a \mod Q_\ell P_\ell) \in R_{Q_\ell P_\ell}^2$ by simply extracting values for corresponding RNS moduli. Then, $b_{0,i}^{(\ell)} = b'$ and $a_{0,i}^{(\ell)} = a' + P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}$. Then, it is obtained $b_{0,i}^{(\ell)} = -a_{0,i}^{(\ell)} \cdot s + e_{0,i}^{(\ell)} + P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}} \cdot s$. If $(b_{0,i}^{(\ell)}, a_{0,i}^{(\ell)})$ as $\mathsf{gk}_{0,i}^{(\ell)}$, the set $\mathsf{gk}_0^{(\ell)} = \{\mathsf{gk}_{0,i}^{(\ell)}\}_{i=0,\cdots,\mathtt{hdnum}_\ell - 1}$ is a valid formal $(0, \ell)$ Galois key. Then a shift-r Galois key can be generated by performing a GalToGal operation on it with the (r, ℓ) Galois key.

In addition, the operations can be optimized further by combining the decomposition processes in the key-switching operation. Trivially, the decomposition process is performed $\mathtt{hdnum}_\ell$ times if all the key-switching operations are performed in a black-box manner like GalToGal. Since the decomposition process is the heaviest operation in the key-switching operation [8], reducing the number of these processes is desirable. Rather than performing the decomposition process after adding $P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}$ to a' for each i, the decomposition process is performed to a' only once and add $[P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}}$ to the j-th decomposed component for each i, where this added value can be pre-computed. Since the number of the decomposition processes is reduced to one, this optimization effectively improves the running time performance. The PubToGal operation is shown in Algorithm 29. The correctness of this optimization is shown in the following theorem, which will be proved in Section 5.5, where $P_\ell Q_\ell = Q_{\ell+1} = (\prod_{j=0}^{\mu-2} Q_{\ell',j}) \cdot \bar{Q}_{\ell',\mu-1}$, $\bar{Q}_{\ell',\mu-1}$ is a divisor of $Q_{\ell',\mu-1}$, and $\mu \leq \mathtt{hdnum}_{\ell'}$

Theorem 5.2.3. *The output of Algorithm 29 is a valid Galois key for the rotation operation for cyclic shift r.*

5.2.4 Galois Key Generation in the Lower Key Level

The desired level-ℓ Galois keys can be generated with only the public key and the Galois keys in the key level higher than ℓ through GalToGal and PubToGal described

Algorithm 28 GalToGal

```
Input: An $(r,\ell)$ Galois key, $\mathsf{gk}_r^{(\ell)} = \{(b_{r,i}^{(\ell)}, a_{r,i}^{(\ell)})\}_{i=0,\cdots,\mathtt{hdnum}_\ell-1} \in (R_{Q_\ell P_\ell}^2)^{\mathtt{hdnum}_\ell}$
        and an $(r',\ell')$ Galois key, where $\ell'$ is higher than $\ell$, $\mathsf{gk}_{r'}^{(\ell')} =$
        $\{(b_{r',i}^{(\ell')}, a_{r',i}^{(\ell')})\}_{i=0,\cdots,\mathtt{hdnum}_\ell-1} \in (R_{Q_{\ell'} P_{\ell'}}^2)^{\mathtt{hdnum}_{\ell'}}$
Output: An $(r+r',\ell)$ Galois key, $\mathsf{gk}_{r+r'}^{(\ell)} = \{b_{r+r',i}^{(\ell)}, a_{r+r',i}^{(\ell)}\}_{i=0,\cdots,\mathtt{hdnum}_\ell-1} \in$
        $(R_{Q_\ell P_\ell}^2)^{\mathtt{hdnum}_\ell}$
for $i = 0$ to $\mathtt{hdnum}_\ell - 1$ do
    $(\tilde{b},\tilde{a}) \leftarrow (b_{r,i}^{(\ell)}(X^{5^{r'}}), a_{r,i}^{(\ell)}(X^{5^{r'}}))$
    $(b_{r+r',i}^{(\ell)}, a_{r+r',i}^{(\ell)}) \leftarrow$ key-switching operation to $(\tilde{b},\tilde{a})$ with the Galois key $\mathsf{gk}_{r'}^{(\ell')}$.
return $\{(b_{r+r',i}^{(\ell)}, a_{r+r',i}^{(\ell)})\}_{i=0,\cdots,\mathtt{hdnum}_\ell-1}$
```

in Algorithm 30. It is assumed that a cyclic shift r of a required Galois key can be represented as $r_0 + \cdots + r_{t-1}$, where each r_i is an element in $\mathcal{U}_\ell$, and I deal with the case when only one level-ℓ Galois key is generated. To generate the (r,ℓ) Galois key, The operation PubToGal is performed with the shift-r_0 Galois key and the public key. Then, a GalToGal operation is performed iteratively with the shift-r_i Galois key and the shift-$\sum_{j=0}^{i-1} r_j$ Galois key to generate a shift-$\sum_{j=0}^{i} r_j$ Galois key for $i = 1, \cdots, t-1$, which outputs the (r,ℓ) Galois key at last. The generation algorithm for one Galois key is described in Algorithm 30.

It is usually required to generate a bundle of Galois keys rather than only one Galois key for a specific service. The more efficient method is dealt with for the case when I need to make a set of Galois keys at once in Section 5.3.

5.2.5 Security Issues

One can be concerned that the server may be able to obtain some information about the secret key using the fact that the Galois keys for any cyclic shifts can be generated by the server indefinitely. However, according to the argument often used in the simulation paradigm in cryptography, if any new information can be efficiently obtained from existing information, this new information is considered to tell me nothing beyond the

Algorithm 29 PubToGal

```
Input: A public key $(b, a) \in R^2_{Q_{k-1}}$, a $(r, \ell')$ Galois key, $\mathsf{gk}_r^{(\ell')} = \{b_{r,j}^{(\ell')}, a_{r,j}^{(\ell')}\}_{j=0,\cdots,\mathtt{hdnum}_{\ell'}-1} \in (R^2_{Q_{\ell'}P_{\ell'}})^{\mathtt{hdnum}_{\ell'}}$, and the key level $\ell$
Output: A $(r, \ell)$ Galois key, $\mathsf{gk}_r^{(\ell)} = \{b_{r,i}^{(\ell)}, a_{r,i}^{(\ell)}\}_{i=0,\cdots,\mathtt{hdnum}_\ell-1} \in (R^2_{Q_\ell P_\ell})^{\mathtt{hdnum}_\ell}$
$(b', a') \leftarrow ([b(X^{5^r})]_{Q_\ell P_\ell}, [a(X^{5^r})]_{Q_\ell P_\ell}) \in R^2_{Q_\ell P_\ell}$
Decompose $a'$ into a vector $(a_0, \cdots, a_{\mu-1}) \in R^\mu_{P_{\ell'}Q_{\ell+1}}$, where $a_j = [a']_{Q_{\ell',j}} + Q_{\ell',j} \cdot \tilde{e}_j$ for small $\tilde{e}_j$'s for $0 \le j \le \mu - 2$ and $a_{\mu-1} = [a']_{\bar{Q}_{\ell',\mu-1}} + \bar{Q}_{\ell',\mu-1} \cdot \tilde{e}_{\mu-1}$ for small $\tilde{e}_{\mu-1}$.
for $i = 0$ to $\mathtt{hdnum}_\ell - 1$ do
    $(\bar{b}, \bar{a}) \leftarrow (0, 0) \in R^2_{P_{\ell'}Q_{\ell'}}$
    for $j \leftarrow 0$ to $\mu - 1$ do
        if $j = \mu - 1$ then
            $(\bar{b}, \bar{a}) \leftarrow (\bar{b}, \bar{a}) + (a_{\mu-1} + [P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{\bar{Q}_{\ell',\mu-1}}) \cdot ([b_{r,\mu-1}^{(\ell')}]_{P_{\ell'}Q_{\ell+1}}, [a_{r,\mu-1}^{(\ell')}]_{P_{\ell'}Q_{\ell+1}})$
        else
            $(\bar{b}, \bar{a}) \leftarrow (\bar{b}, \bar{a}) + (a_j + [P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}}) \cdot ([b_{r,j}^{(\ell')}]_{P_{\ell'}Q_{\ell+1}}, [a_{r,j}^{(\ell')}]_{P_{\ell'}Q_{\ell+1}})$
    $(b_{r,i}^{(\ell)}, a_{r,i}^{(\ell)}) \leftarrow (\lfloor P_{\ell'}^{-1} \cdot \bar{b} \rceil, \lfloor P_{\ell'}^{-1} \cdot \bar{a} \rceil) \in R^2_{Q_{\ell+1}} = R^2_{P_\ell Q_\ell}$
    $b_{r,i}^{(\ell)} \leftarrow b_{r,i}^{(\ell)} + b'$
return $\{(b_{r,i}^{(\ell)}, a_{r,i}^{(\ell)})\}_{i=0,\cdots,\mathtt{hdnum}_\ell-1}$
```

existing information [59]. Thus, even if new Galois keys are generated indefinitely with the proposed algorithms from the Galois keys sent by the client, these new Galois keys do not give the server any new information beyond the public keys and the Galois keys in the highest key level sent by the client.

Thus, it is only required to consider the security of the public key and the Galois keys at the highest key level sent by the client. As mentioned in Section 5.2.2, the generating method for the public key and the Galois key in the highest key level by the client is exactly the same as those of the conventional FHE schemes. Just as the conventional FHE schemes are based on the circular security assumption, the proposed hierarchical Galois key generation scheme also requires the circular security assumption. The public key is an element of $R^2_{Q_{k-1}}$, and the Galois key in the highest key level is an element of $(R_{Q_{k-1}P_{k-1}})^2)^{\mathtt{hdnum}_{k-1}}$. Since the main factor that affects the security level is the maximum modulus bit-length of rings, the value of $Q_{k-1}P_{k-1}$ is the main factor for security. For a given polynomial modulus degree N and the secret key Hamming weight h, the maximum modulus bit length can be given to guarantee the security level λ [8, 22], and the bit-length of $Q_{k-1}P_{k-1}$ should not exceed this bit length.

5.3 Efficient Generation Method of Galois Key Set

In the previous section, I dealt with the specific algorithms needed to make a Galois key in the lower key level using Galois keys in the higher key level. However, I often require many Galois keys at once, especially for certain services requested by the client. Thus, it is necessary to efficiently generate a set of Galois keys using the Galois keys in the higher key level. I need to reduce the number of these GalToGal operations and PubToGal operations to efficiently generate hierarchical Galois keys. Note that there are many intermediate Galois keys in the hierarchical Galois key system. Given a certain fixed set of Galois keys in the higher key level, the key problem is how to

Algorithm 30 GalKeyGen for one Galois key

```
Input: A public key $(b, a) \in R^2_{Q_{k-1}}$, a set of Galois keys $\mathcal{G}_{\mathcal{U}_\ell} = \{\mathsf{gk}_r^{(\ell_r)} = \{(b_{r,i}^{(\ell_r)}, a_{r,i}^{(\ell_r)})\}_{i=0,\cdots,\mathtt{hdnum}_{\ell_r}-1} \in (R^2_{Q_{\ell_r}P_{\ell_r}})^{\mathtt{hdnum}_{\ell_r}} | r \in \mathcal{U}_\ell\}$ for cyclic shift generator set $\mathcal{U}_\ell$ in the key level higher than $\ell$, and a cyclic shift $r = \sum_{u=0}^{t-1} r_u$ for $r_u \in \mathcal{U}_\ell$
Output: An $(r, \ell)$ Galois key, $\mathsf{gk}_r^{(\ell)} = \{b_{r,i}^{(\ell)}, a_{r,i}^{(\ell)}\}_{i=0,\cdots,\mathtt{hdnum}_\ell-1} \in (R^2_{Q_\ell P_\ell})^{\mathtt{hdnum}_\ell}$
$\{(b_{r_0,i}^{(\ell)}, a_{r_0,i}^{(\ell)})_{i=0,\cdots,\mathtt{hdnum}_\ell}\} \leftarrow$ PubToGal with the public key and the $(r_0, \ell_{r_0})$ Galois key
for $h = 1$ to $t-1$ do
    $\{(b^{(\ell)}_{\sum_{j=0}^{h} r_j,i}, a^{(\ell)}_{\sum_{j=0}^{h} r_j,i})_{i=0,\cdots,\mathtt{hdnum}_\ell}\} \leftarrow$ GalToGal with $\{(b^{(\ell)}_{\sum_{j=0}^{h-1} r_j,i}, a^{(\ell)}_{\sum_{j=0}^{h-1} r_j,i})_{i=0,\cdots,\mathtt{hdnum}_\ell}\}$ and the $(r_h, \ell_{r_h})$ Galois key
return $\mathsf{gk}_r^{\ell} = \{(b_{r,i}^{(\ell)}, a_{r,i}^{(\ell)})_{i=0,\cdots,\mathtt{hdnum}_\ell}\}$
```

minimize the number of operations for generating these intermediate Galois keys by systematically organizing the generating sequence of the Galois keys.

5.3.1 Reduction to Minimum Spanning Arborescence Problem and Minimum Spanning Tree Problem

Given a set $\mathcal{U}_\ell$ of specific fixed Galois keys in the higher key level than ℓ, generating level-ℓ Galois keys with as few operations as possible is desirable. In other words, it becomes important to use the least amount of operations of GalToGal and PubToGal by arranging the order in which the Galois keys in the set are generated. I propose an algorithm that can determine the order of generating Galois keys in the set in the hierarchical Galois key system to reduce the number of operations.

To this end, I reduce the problem of determining the order of generation of Galois keys to the minimal spanning arborescence problem, a well-known graph-theoretic computational problem. First, set the $|\mathcal{T}_\ell| + 1$ nodes for each element in the $\mathcal{T}_\ell \cup \{0\}$, and then set the directed edge weight between any two nodes a, b as the minimum number of elements in $\mathcal{U}_\ell$ required to add up to $|a - b|$ allowing repetition. The method for setting this edge weight will be given in the next subsection. There are some identical points to the minimum arborescence problem in the ordering of the Galois key generation problem as follows.

- I need to generate each Galois key only once. This fact is related to the property of the arborescence that any node has only one path from the root node.
- Each Galois key can be generated by using a GalToGal or a PubToGal from the public key or existing Galois keys. An edge from the node a to the node b with weight w means that the (b, ℓ) Galois key can be generated from the (a, ℓ) Galois key with w operations of GalToGal and PubToGal.
- All Galois keys should be generated from the public key and the higher level Galois keys. An arborescence has only one root node that is the source of all nodes, and this root node corresponds to the public key.
- I need to minimize the total number of key-switching operations to generate all Galois keys. The minimum spanning arborescence problem is to find the arborescence with the minimum total weight, which corresponds to the total number of PubToGal operations or GalToGal operations.

Therefore, the graph produced in this way can be seen as a directed graph, and the key problem is to find a spanning arborescence with a minimum sum of edges, which is the goal of the minimum spanning arborescence problem. If I find a spanning arborescence in the graph, I can view the node with zero as a public key and generate the Galois keys along the obtained tree. The minimum spanning arborescence problem can be solved by Edmonds' algorithm [29], and thus an answer to this problem can be efficiently obtained.

If the Galois keys in the higher key level exist in pairs of different signs of the same absolute value, a faster and more efficient solution for generating the Galois keys in the lower key level can be obtained by reducing to another computation problem. If a shift-r_1 Galois key can be generated with m operations from a shift-r_2 Galois key, I can generate the shift-r_2 Galois key from that of cyclic shift r_1 with the higher-level Galois keys for cyclic shifts having the same absolute value with the different sign. In view of the corresponding graph, any pairs of two edges (r_1, r_2) and (r_2, r_1) exist and have the same edge weight. Thus, I can replace the directed graph with the undirected graph with the same nodes in which each edge has the same weight as the corresponding edge in the directed graph. For the undirected graph, I can reduce this problem to the minimum spanning tree problem, which can be solved by Prim's algorithm [63].

It is noted that this solution is not exactly the optimal solution since the insertion of additional nodes can reduce the operations further. If I set the nodes for all cyclic shifts (i.e., $\pm 1, \pm 2, \pm 3, \cdots$) in the graph, the key problem is to find the minimum Steiner tree for required cyclic shifts. The Steiner tree in a graph is a tree connecting a subset of designated nodes, and the problem of finding the Steiner tree is known as an NP-hard problem. Thus, I choose the near-optimal solution using a more practically feasible algorithm. Designing a fast algorithm to find the solution closer to the optimal solution in the proposed situation is an important future work.

5.3.2 Edge Weight for p-ary Galois Keys

A method to compute the edge of each graph have to be considered, where I need to find a way to represent the difference between two nodes as a sum of the minimum number of elements in $\mathcal{U}_\ell$, allowing repetition. In general, the server can ask the client for a well-designed set of $\mathcal{U}_\ell$ so that it can be easy to represent any given number as the desired sum in $\mathcal{U}_\ell$. Rather than proposing the general method for unstructured $\mathcal{U}_\ell$, I suggest a specific example of key management system with $\mathcal{U}_\ell$ with power-of-p integers within the desired interval. I will discuss how to obtain edges for both cases when $\mathcal{U}_\ell$ consists of power-of-p integers with both signs and when it consists of only positive power-of-p integers.

The easier case is considered first, a set of positive power-of-p, in which the ro-

```
Algorithm 31 ComputeEdgePos
Input: A power base $p$ for the set $\mathcal{U}_\ell$ with only positive numbers and a number $t$ to be
        summed
Output: The minimum number of elements in $\mathcal{U}_\ell$ summed to $t$ allowing repetition
$(t_0 t_1 \cdots t_{\ell-1})_{(p)} \leftarrow p$-ary representation of $t$
return $\sum_{i=0}^{\ell-1} t_i$
```

tation graph is a directed graph. In this case, each edge can be computed as follows. First, I can find the difference between the end node and the start node of the edge, and then express this difference in the p-ary representation, and then set the sum of the digits as the edge weight. This algorithm is described in Algorithm 31 without proof.

Next, I consider the case of power-of-p integers with both signs in which the rotation graph is an undirected graph as Subsection 5.3.1. In this case, since the power-of-two integers with different signs can add up to the value, expressing p-ary representation is not enough to find the optimal solution. Instead, I propose the following algorithm to obtain the edge weight between any two nodes, which is efficient enough for the input range. Assume that r is the difference between the two given nodes. If r is a multiple of p, then recursively output a value of $\mathsf{Alg}(r/p)$, otherwise find r_1 such that $pr_1 \leq r < p(r_1+1)$ and recursively output $\min\{\mathsf{Alg}(r_1) + (r - pr_1), \mathsf{Alg}(r_1+1) + (p(r_1+1) - r)\}$. This algorithm is described in Algorithm 32. To help understanding the graph-theoretic algorithms for Galois key generation in Section 5.3, I depict the corresponding graph and the minimum spanning tree for $\mathcal{T}_\ell = \{1, 13, 16, 17, 19\}$ and $\mathcal{U}_\ell = \{\pm1, \pm2, \pm4, \pm8, \pm16\}$ in Figure 5.2.

5.3.3 Hoisted Galois Key Generation

The previous subsections focus on the reduction of the number of GalToGal and PubToGal operations. In this subsection, I further reduce the number of the decompose processes by the hoisting technique. The hoisting technique is a method for minimizing the number of operations by interchanging or combining operations without

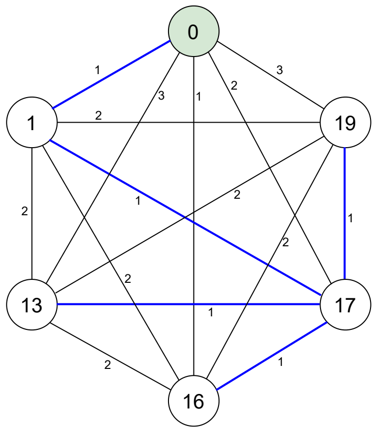


Figure 5.2: Galois key graph for $\mathcal{T}_\ell = \{1, 13, 16, 17, 19\}$ and $\mathcal{U}_\ell = \{\pm1, \pm2, \pm4, \pm8, \pm16\}$.

Algorithm 32 ComputeEdgeBoth

Input: A power base p for the set $\mathcal{S}$ with both signs and the number t to be summed

Output: The minimum number of elements in $\mathcal{S}$ summed to t allowing repetition

```
if $p|t$ then
    return ComputeEdgeBoth$(p, t/p)$
else
    $r \leftarrow \lfloor |t|/p \rfloor$
    if $r = 0$ then
        return $|t|$
    else
        return $\min\{$ComputeEdgeBoth$(p, r) + (|t| - pr)$, ComputeEdgeBoth$(p, r + 1) + (p(r+1) - |t|)\}$
```

changing functionalities. This technique has been used in the linear transformation in the FHE schemes, and the optimization of the bootstrapping of the FHE schemes is one of its important applications [8, 43]. I propose the hoisting method for generating of the Galois key set in the hierarchical Galois key generation systems.

The target situation is when several level-ℓ Galois keys are generated from the public key or one level-ℓ Galois key with Galois keys in the key level ℓ' higher than ℓ. If I want to generate d Galois keys, I can naively perform exactly d $\mathsf{PubToGal}$ operations or d $\mathsf{GalToGal}$ operations. As I stated in Subsection 5.2.3, the decomposition process is the most time-consuming process in the key-switching operation, and thus the decomposition process is desirable to be reduced further. To this end, I postpone the process of automorphism in line 2 of Algorithm 28 or line 1 of Algorithm 29 to the last of the operations to combine the decomposition processes into one process. To maintain the functionality of the operation, I conduct the automorphism inversely to the Galois keys in the key level higher than ℓ before the inner-product with the decomposed components. If the source Galois key is the public key, I reduce the number of decomposition processes from d to one for generating d Galois keys. If the source Galois key is the other Galois key in the same key level, I reduce the number of decomposition processes from $d \cdot \texttt{hdnum}_\ell$ to $\texttt{hdnum}_\ell$. The hoisted version of $\mathsf{GalToGal}$ and $\mathsf{PubToGal}$ operations are described in Algorithms 33 and 34. The whole generation algorithm is described in Algorithm 35. I use breath-first search when I search each node in the output arborescence. This search method is desirable for the hoisted generation of Galois keys.

5.4 Simulation Results with ResNet Models

In this section, I numerically verify the validity of the proposed hierarchical Galois key generation method with an appropriate computing environment for the client-server model with the ResNet standard neural network and the CKKS scheme. In the cloud

Algorithm 33 HoistedGalToGal

Input: An (r,ℓ) Galois key, $\mathsf{gk}_r^{(\ell)} = \{(b_{r,i}^{(\ell)}, a_{r,i}^{(\ell)})\}_{i=0,\cdots,\mathtt{hdnum}_\ell-1} \in (R_{Q_\ell P_\ell}^2)^{\mathtt{hdnum}_\ell}$ and d (r'_α,ℓ') Galois keys, where ℓ' is higher than ℓ, $\mathsf{gk}_{r'_\alpha}^{(\ell')} = \{(b_{r'_\alpha,i}^{(\ell')}, a_{r'_\alpha,i}^{(\ell')})\}_{i=0,\cdots,\mathtt{hdnum}_\ell-1} \in (R_{Q_{\ell'}P_{\ell'}}^2)^{\mathtt{hdnum}_{\ell'}}$ for $\alpha = 0,\cdots,d-1$

Output: d $(r+r'_\alpha,\ell)$ Galois keys, $\mathsf{gk}_{r+r'_\alpha}^{(\ell)} = \{b_{r+r'_\alpha,i}^{(\ell)}, a_{r+r'_\alpha,i}^{(\ell)}\}_{i=0,\cdots,\mathtt{hdnum}_\ell-1} \in (R_{Q_\ell P_\ell}^2)^{\mathtt{hdnum}_\ell}$ for $\alpha = 0,\cdots,d-1$

```
for $i = 0$ to $\mathtt{hdnum}_\ell - 1$ do
    Decompose $a_{r,j}^{(\ell)}$ into a vector $(a_0,\cdots,a_{\mu-1}) \in R_{P_{\ell'}Q_{\ell+1}}^{\mu}$, where $a_j = [a]_{Q_{\ell',j}} + Q_{\ell',j}\cdot\tilde{e}_j$ for small $\tilde{e}_j$'s for $0 \le j \le \mu-2$ and $a_{\mu-1} = [a]_{\bar{Q}_{\ell',\mu-1}} + \bar{Q}_{\ell',\mu-1}\cdot\tilde{e}_{\mu-1}$ for small $\tilde{e}_{\mu-1}$.
    for $\alpha = 0$ to $d-1$ do
        $(\bar{b},\bar{a}) \leftarrow (0,0) \in R_{P_\ell Q_\ell}^2$
        for $j \leftarrow 0$ to $\mu-1$ do
            $(\bar{b},\bar{a}) \leftarrow (\bar{b},\bar{a}) + a_j\cdot([b_{r'_\alpha,j}^{(\ell')}(X^{5^{-r'_\alpha}})]_{P_{\ell'}Q_{\ell+1}}, [a_{r'_\alpha,j}^{(\ell')}(X^{5^{-r'_\alpha}})]_{P_{\ell'}Q_{\ell+1}})$
        $(b_{r+r'_\alpha,i}^{(\ell)}, a_{r+r'_\alpha,i}^{(\ell)}) \leftarrow (\lfloor P_{\ell'}^{-1}\cdot\bar{b}\rceil, \lfloor P_{\ell'}^{-1}\cdot\bar{a}\rceil) \in R_{Q_{\ell+1}}^2$
        $b_{r+r'_\alpha,i}^{(\ell)} \leftarrow b_{r+r'_\alpha,i}^{(\ell)} + b_{r,i}^{(\ell)}$
        $(b_{r+r'_\alpha,i}^{(\ell)}, a_{r+r'_\alpha,i}^{(\ell)}) \leftarrow (b_{r+r'_\alpha,i}^{(\ell)}(X^{5^{r'_\alpha}}), a_{r+r'_\alpha,i}^{(\ell)}(X^{5^{r'_\alpha}}))$
return $\{b_{r+r'_\alpha,i}^{(\ell)}, a_{r+r'_\alpha,i}^{(\ell)}\}_{i=0,\cdots,\mathtt{hdnum}_\ell-1}$ for $\alpha = 0,\cdots,d-1$
```

Algorithm 34 HoistedPubToGal

Input: A public key $(b, a) \in R^2_{Q_{k-1}}$, d (r_α, ℓ') Galois keys, where ℓ' is higher than ℓ, $\mathsf{gk}^{(\ell')}_{r_\alpha} = \{(b^{(\ell')}_{r_\alpha,i}, a^{(\ell')}_{r_\alpha,i})\}_{i=0,\cdots,\mathtt{hdnum}_\ell-1} \in (R^2_{Q_{\ell'}P_{\ell'}})^{\mathtt{hdnum}_{\ell'}}$ for $\alpha = 0, \cdots, d-1$, and the key level ℓ

Output: d (r_α, ℓ) Galois keys, $\mathsf{gk}^{(\ell)}_{r_\alpha} = \{b^{(\ell)}_{r_\alpha,i}, a^{(\ell)}_{r_\alpha,i}\}_{i=0,\cdots,\mathtt{hdnum}_\ell-1} \in (R^2_{Q_\ell P_\ell})^{\mathtt{hdnum}_\ell}$ for $\alpha = 0, \cdots, d-1$

```
Decompose $a$ into a vector $(a_0, \cdots, a_{\mu-1}) \in R^\mu_{P_{\ell'}Q_{\ell+1}}$, where $a_j = [a]_{Q_{\ell',j}} + Q_{\ell',j} \cdot \tilde{e}_j$ for small $\tilde{e}_j$'s for $0 \le j \le \mu-2$ and $a_{\mu-1} = [a]_{\bar{Q}_{\ell',\mu-1}} + \bar{Q}_{\ell',\mu-1} \cdot \tilde{e}_{\mu-1}$ for small $\tilde{e}_{\mu-1}$.
for $i = 0$ to $\mathtt{hdnum}_\ell - 1$ do
    $(\bar{b}, \bar{a}) \leftarrow (0, 0) \in R^2_{P_\ell Q_\ell}$
    for $\alpha = 0$ to $d-1$ do
        for $j \leftarrow 0$ to $\mu - 1$ do
            if $j = \mu - 1$ then
                $(\bar{b}, \bar{a}) \leftarrow (\bar{b}, \bar{a}) + (a_{\mu-1} + [P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}^{-1}_{\ell,i}]_{Q_{\ell,i}}]_{\bar{Q}_{\ell',\mu-1}}) \cdot ([b^{(\ell')}_{r_\alpha,j}(X^{5^{-r_\alpha}})]_{P_{\ell'}Q_{\ell+1}}, [a^{(\ell')}_{r_\alpha,j}(X^{5^{-r_\alpha}})]_{P_{\ell'}Q_{\ell+1}})$
            else
                $(\bar{b}, \bar{a}) \leftarrow (\bar{b}, \bar{a}) + (a_j + [P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}^{-1}_{\ell,i}]_{Q_{\ell,i}}]_{Q_{\ell',j}}) \cdot ([b^{(\ell')}_{r_\alpha,j}(X^{5^{-r_\alpha}})]_{P_{\ell'}Q_{\ell+1}}, [a^{(\ell')}_{r_\alpha,j}(X^{5^{-r_\alpha}})]_{P_{\ell'}Q_{\ell+1}})$
        $(b^{(\ell)}_{r_\alpha,i}, a^{(\ell)}_{r_\alpha,i}) \leftarrow (\lfloor P^{-1}_{\ell'} \cdot \bar{b} \rceil, \lfloor P^{-1}_{\ell'} \cdot \bar{a} \rceil) \in R^2_{Q_{\ell+1}}$
        $b^{(\ell)}_{r_\alpha,i} \leftarrow b^{(\ell)}_{r_\alpha,i} + [b]_{Q_{\ell+1}}$
        $(b^{(\ell)}_{r_\alpha,i}, a^{(\ell)}_{r_\alpha,i}) \leftarrow (b^{(\ell)}_{r_\alpha,i}(X^{5^{r_\alpha}}), a^{(\ell)}_{r_\alpha,i}(X^{5^{r_\alpha}}))$
return $\{b^{(\ell)}_{r_\alpha,i}, a^{(\ell)}_{r_\alpha,i}\}_{i=0,\cdots,\mathtt{hdnum}_\ell-1}$ for $\alpha = 0, \cdots, d-1$
```

```
Algorithm 35 GalKeyGen
Input: A cyclic shift set $\mathcal{T}_\ell$ for the key level $\ell$, a cyclic shift generator set $\mathcal{U}_\ell$ for the
       key level higher than $\ell$, a set of Galois keys $\mathcal{G}_{\mathcal{U}_\ell}$ for cyclic shifts in $\mathcal{U}_\ell$ in the
       key level higher than $\ell$, and a public key $(b, a) \in R_{Q_{k-1}}$
Output: A set of Galois keys $\mathcal{G}_{\mathcal{T}_\ell}$ for a cyclic shift set $\mathcal{T}_\ell$
$V \leftarrow \mathcal{T} \cup \{0\}$
$E \leftarrow \{(v, w) | v, w \in V\}$
$w(v, w) \leftarrow$ the minimum number of elements in $\mathcal{S}$ summed to $w - v$ allowing repetition
$G' = (V, E') \leftarrow$ Edmonds' algorithm with $G = (V, E)$ ;   // It can be replaced with Prim's algorithm when $\mathcal{U}_\ell$ is symmetric around zero.
$Q[] \leftarrow$ empty queue for nodes
$\mathcal{G}_{\mathcal{T}_\ell} \leftarrow \varnothing$
while $Q$ is not empty do
    $v \leftarrow$ dequeue from $Q$
    $W \leftarrow$ the set of nodes adjacent to $v$.
    if $v = 0$ then
        Generate the set of Galois keys $\mathcal{G}_W = \{\mathsf{gk}_w^{(\ell)} | w \in W\}$ from $(b, a)$ using PubToGal or HoistedPubToGal
    else
        Generate the set of Galois keys $\mathcal{G}_W = \{\mathsf{gk}_w^{(\ell)} | w \in W\}$ from $\mathsf{gk}_v^{(\ell)}$ using GalToGal or HoistedGalToGal
    $\mathcal{G}_{\mathcal{T}_\ell} \leftarrow \mathcal{G}_{\mathcal{T}_\ell} \cup \mathcal{G}_W$
    Enqueue elements in $W$ to $Q$.
return $\mathcal{G}_{\mathcal{T}_\ell}$
```

computing model, the server usually has high-performance computing resources, and the client has only a general-purpose personal computer. To simulate this environment, I use a PC with Intel(R) Core(TM) i7-10700 CPU and no accelerator as a client and a high-performance server with a AMD Ryzen Threadripper PRO 3995WX CPU processor and a NVIDIA GeForce RTX 3090 GPU accelerator.

As a representative example of complex computation models, I assume that the service requested by the client requires the ResNet-20 model for the CIFAR-10 dataset or the ResNet-18 model for the ImageNet dataset. In previous chapter, several techniques for minimizing homomorphic operations for ResNet models with the CKKS scheme are proposed. I proposed a multiplexed parallel convolution technique, an index-arranging average pooling technique, to effectively perform ResNet models dealing with three-dimensional tensor structures on CKKS schemes with one-dimensional vector structures. They enable efficient computation of each component of the ResNet model. In addition, when the bootstrapping is performed, sparse-slot bootstrapping [20] is performed with different slots for different layers, where I can require different Galois keys for the bootstrapping operation with different number of sparse slots. Also, the sparse-secret encapsulation method [9] is assumed to be used for the bootstrapping with the dense secret key with more reduced running time and higher precision. I use the baby-step giant-step algorithm for fully connected layers [20]. When using all of these methods with CKKS algorithms using $N = 2^{16}$ for the polynomial modulus degree, I found that 265 Galois keys for different cyclic shifts are required in the ResNet-20 model processing the CIFAR-10 dataset, and the set of cyclic shifts required to perform the ResNet-20 for the CIFAR-10 dataset is enumerated as follows.

- $\mathcal{T}_0^{\mathsf{ResNet-20}}$ = {1, -1, 2, -2, 3, 4, -4, 5, 6, 7, 8, -8, 9, 12, 16, -16, 18, 27, 28, 32, -32, 36, 45, 48, 54, 56, 63, 64, -64, 72, 80, 84, 96, -96, 112, 128, -128, 192, 256, 384, 512, 768, 959, 960, 990, 991, -994, 1008, 1023, 1024, -1024, -1025, 1036, -1056, 1064, -1088, 1092, -1120, 1536, 1952, 1982, 1983, 2016, 2044, 2047, 2048, -2048, 2072, 2078, -2080, 2100, -2112, -2144, 3007, 3024, 3040, 3052, 3070, 3071, 3072, -3072, 3080, -3104, 3108, -3136, -3168, 3840, 3904, 3968, 4031, 4032, 4062, 4063, 4080, 4084, 4088, 4092, 4095, 4096, -4096, 4104, -4128, -4131, -4195, 5023, 5024, 5054, 5055, 5087, 5118, 5119, 5120, -5120, -5152, -5155, -5219, 6047, 6078, 6079, 6111, 6112, 6142, 6143, 6144, -6144, -6176, -6179, -6243, 7071, 7102, 7103, 7135, 7166, 7167, 7168, -7168, -7200, -7203, -7267, 7936, 8000, 8064, 8095, 8126, 8127, 8128, 8159, 8176, 8180, 8184, 8188, 8190, 8191, 8192, -8192, -8195, 8200, -8225, -8226, -8227, -8259, -8290, -8291, 9149, 9183, 9184, 9213, 9215, 9216, -9219, -9249, -9250, -9251, -9283, -9314, -9315, 10173, 10207, 10208, 10237, 10239, 10240, -10240, -10243, -10273, -10274, -10275, -10307, -10338, -10339, 11197, 11231, 11232, 11261, 11263, 11264, -11264, -11267, -11297, -11298, -11299, -11331, -11362, -11363, 12221, 12255, 12256, 12285, 12287, 12288, -12288, -12321, -12385, 13214, 13216, 13246, 13278, 13279, 13280, 13310, 13311, 13312, -13345, -13409, 14238, 14240, 14270, 14302, 14303, 14304, 14334, 14335, 14336, -14336, -14369, -14433, 15262, 15264, 15294, 15326, 15327, 15328, 15358, 15359, 15360, -15393, -15457, 15872, 16000, 16128, 16256, 16286, 16288, 16318, 16350, 16351, 16352, 16368, 16372, 16376, -16376, 16380, 16382, 16383, 16384}

The set of cyclic shifts required to perform the ResNet-18 for the ImageNet dataset is enumerated as follows.

- $\mathcal{T}_0^{\mathsf{ResNet-18}} = \{$1, -1, 2, -2, 3, -3, 4, -4, 5, -5, 6, -6, 7, -7, 8, -8, -9, -10, -11, -12, -13, -14, -15, 16, -16, -17, -18, -19, -20, -21, -22, -23, 24, -24, -25, -26, -27, -28, -29, -30, -31, 32, -32, 39, 64, 78, 96, 117, 128, 156, 160, 192, 195, 221, 222, 224, -224, -225, -226, -227, -228, -229, -230, -231, -232, -233, 234, -234, -235, -236, -237, -238, -239, -240, -241, -242, -243, -244, -245, -246, -247, -248, -249, -250, -251, -252, -253, -254, -255, 256, -256, 273, 312, 351, 384, 390, 429, 445, 448, -448, -449, -450, -451, -452, -453, -454, -455, -456, -457, -458, -459, -460, -461, -462, -463, -464, -465, -466, -467, 468, -468, -469, -470, -471, -472, -473,

-474, -475, -476, -477, -478, -479, 507, 512, -512, 546, 576, 585, 624, 663, 669, -672, -673, -674, -675, -676, -677, -678, -679, -680, -681, -682, -683, -684, -685, -686, -687, -688, -689, -690, -691, -692, -693, -694, -695, -696, -697, -698, -699, -700, -701, 702, -702, -703, 741, 768, -768, 780, 819, 858, 896, -896, 897, -897, -898, -899, -900, -901, -902, -903, -904, -905, -906, -907, -908, -909, -910, -911, -912, -913, -914, -915, -916, -917, -918, -919, -920, -921, -922, -923, -924, -925, -926, -927, 936, 960, 975, -999, 1014, 1024, -1024, 1053, 1092, -1120, -1121, -1122, -1123, -1124, -1125, -1126, -1127, -1128, -1129, -1130, 1131, -1131, -1132, -1133, -1134, -1135, -1136, -1137, -1138, -1139, -1140, -1141, -1142, -1143, -1144, -1145, -1146, -1147, -1148, -1149, -1150, -1151, 1152, 1170, 1209, 1248, 1287, 1326, 1344, -1344, -1345, -1346, -1347, -1348, -1349, -1350, -1351, -1352, -1353, -1354, -1355, -1356, -1357, -1358, -1359, -1360, -1361, -1362, -1363, -1364, 1365, -1365, -1366, -1367, -1368, -1369, -1370, -1371, -1372, -1373, -1374, -1375, 1404, 1443, 1482, 1536, -1568, -1569, -1570, -1571, -1572, -1573, -1574, -1575, -1576, -1577, -1578, -1579, -1580, -1581, -1582, -1583, -1584, -1585, -1586, -1587, -1588, -1589, -1590, -1591, -1592, -1593, -1594, -1595, -1596, -1597, -1598, -1599, 1728, 1792, -1792, -1793, -1794, -1795, -1796, -1797, -1798, -1799, -1800, -1801, -1802, -1803, -1804, -1805, -1806, -1807, -1808, -1809, -1810, -1811, -1812, -1813, -1814, -1815, -1816, -1817, -1818, -1819, -1820, -1821, -1822, -1823, 1920, -2016, -2017, -2018, -2019, -2020, -2021, -2022, -2023, -2024, -2025, -2026, -2027, -2028, -2029, -2030, -2031, -2032, -2033, -2034, -2035, -2036, -2037, -2038, -2039, -2040, -2041, -2042, -2043, -2044, -2045, -2046, -2047, 2048, 2112, -2240, -2241, -2242, -2243, -2244, -2245, -2246, -2247, -2248, -2249, -2250, -2251, -2252, -2253, -2254, -2255, -2256, -2257, -2258, -2259, -2260, -2261, -2262, -2263, -2264, -2265, -2266, -2267, -2268, -2269, -2270, -2271, 2304, -2464, -2465, -2466, -2467, -2468, -2469, -2470, -2471, -2472, -2473, -2474, -2475, -2476, -2477, -2478, -2479, -2480, -2481, -2482, -2483, -2484, -2485, -2486,

Table 5.1: Encryption parameters in the CKKS scheme for ResNet models

Parameters	ResNet-20 for CIFAR-10	ResNet-18 for ImageNet
Polynomial modulus degree	2^{16}	2^{17}
Secret key Hamming weight	2^{15}	2^{16}
Gaussian error stand. dev.	3.2	3.2
Minimum security level	128-bit	128-bit
Maximum modulus bit-length	1792	3220

-2487, -2488, -2489, -2490, -2491, -2492, -2493, -2494, -2495, 2496, 2688, -2688, -2689, -2690, -2691, -2692, -2693, -2694, -2695, -2696, -2697, -2698, -2699, -2700, -2701, -2702, -2703, -2704, -2705, -2706, -2707, -2708, -2709, -2710, -2711, -2712, -2713, -2714, -2715, -2716, -2717, -2718, -2719, 2880, -2912, -2913, -2914, -2915, -2916, -2917, -2918, -2919, -2920, -2921, -2922, -2923, -2924, -2925, -2926, -2927, -2928, -2929, -2930, -2931, -2932, -2933, -2934, -2935, -2936, -2937, -2938, -2939, -2940, -2941, -2942, -2943, 3072, -3136, -3137, -3138, -3139, -3140, -3141, -3142, -3143, -3144, -3145, -3146, -3147, -3148, -3149, -3150, -3151, -3152, -3153, -3154, -3155, -3156, -3157, -3158, -3159, -3160, -3161, -3162, -3163, -3164, -3165, -3166, -3167, -3360, -3361, -3362, -3363, -3364, -3365, -3366, -3367, -3368, -3369, -3370, -3371, -3372, -3373, -3374, -3375, -3376, -3377, -3378, -3379, -3380, -3381, -3382, -3383, -3384, -3385, -3386, -3387, -3388, -3389, -3390, -3391, 3584, -3584, 4096, 5120, 6144, 7168, -7168, 8192, -8192, 14336, 16384, -16384, 21504, -22528, 24576, -24576, 28672, -29696, 32768}

The CKKS scheme is used for the simulation, and the parameters of the CKKS scheme I use for the simulation are shown in Table 5.1. The `lattigo` library [1] is used for the simulation, and the `CUDA` library by NVIDIA is used for GPU acceleration of the Galois key generation. The Galois key generation with the GPU processor is

Table 5.2: Modulus bit-lengths and decomposition numbers for each Galois key generation scheme in ResNet-20 for CIFAR-10

Galois key generation	Modulus bit-length	Decomposition number
Conventional	$(\log Q_0, \log P_0)$ = (1345, 129)	12
Two-level	$(\log Q_0, \log P_0, \log P_1)$ = (1345, 129, 158)	(12, 11)
Three-level	$(\log Q_0, \log P_0, \log P_1, \log P_2)$ = (1345, 129, 158, 160)	$(12, 11, 11)$

implemented based on [48]. In the server, algorithms for the preparation of Galois key generation are executed on the CPU processor and all actual Galois key generation is computed by the GPU processor. The running time for Galois key generation by the client, the communication amount between the client and the server, the required storage for Galois keys in the server, and the running time for Galois key generation by the server are measured and presented in this section.

In the conventional CKKS scheme, the client generates all the required Galois keys and transmits them to the server. In the two-level hierarchical Galois key scheme, the client generates the quaternary level-1 Galois key set with both signs and transmits them to the server, where the set of the cyclic shifts is $\{\pm 1, \pm 4, \pm 16, \cdots, \pm 2^{12}\}$. Then, the server generates the required Galois keys for the ResNet models from this quaternary level-1 Galois key set. In the three-level hierarchical Galois key scheme, the client generates the 16-ary level-2 Galois key set with both signs and transmits them to the server, where the set of the cyclic shifts is $\{\pm 1, \pm 16, \pm 256, \pm 2^{12}\}$. If the server needs to give services to the client immediately, the server generates the required Galois keys for the ResNet models from this 16-ary level-2 Galois key set. If the services do not need to be given to the client immediately and does not determined

Table 5.3: Modulus bit-lengths and decomposition numbers for each Galois key generation scheme in ResNet-18 for ImageNet

Galois key generation	Modulus bit-length	Decomposition number
Conventional	$(\log Q_0, \log P_0)$ = (1465, 392)	4
Two-level	$(\log Q_0, \log P_0, \log P_1)$ = (1465, 392, 637)	(4, 3)
Three-level	$(\log Q_0, \log P_0, \log P_1, \log P_2)$ = (1465, 392, 637, 637)	$(4, 3, 4)$

just after receiving the level-2 Galois keys, the server can generate more level-1 Galois keys for faster Galois key generation in the offline phase before the services so that level-2 Galois keys and level-1 Galois keys constitutes a quaternary Galois key set. Then, the server generates the required level-0 Galois keys for the ResNet models from this quaternary level-2 and level-1 Galois key set just after the services are requested, which is the online phase. Tables 5.2 and 5.3 show the evaluation modulus in the key level zero, the hierarchical special moduli for each key level, and each decomposition number for each key level used in the simulation.

Table 5.4 shows the number of core operations for generation of each Galois key set for ResNet models using 4-ary or 16-ary Galois key set, and it shows the effectiveness of the hoisted Galois key generation and the Prim's algorithm. Note that the total numbers of GalToGal and PubToGal operations are close to the number of Galois keys. Roughly speaking, 1.04 and 1.07 numbers of key-switching operations for a Galois key are needed on average if I use 4-ary Galois key generation set, and 1.43 and 1.18 numbers of key-switching operations for a Galois key are needed on average if I use 16-ary Galois key generation set. It means that most of the Galois keys can be generated by only one GalToGal or PubToGal operation from other Galois key, which

Table 5.4: Number of core operations optimized by hoisted Galois key generation and Prim's algorithm

		ResNet-20 for CIFAR-10		ResNet-18 for ImageNet	
		4-ary	16-ary	4-ary	16-ary
No. of Galois Keys		265		617	
GalToGal	Total	263	372	649	721
	Decompose	149	292	347	529
PubToGal	Total	14	7	14	8
	Decompose	1	1	1	1

is the result of Prim's algorithm.

Note that the number of the decompose processes is effectively reduced compared to the total numbers of GalToGal and PubToGal operations by the hoisted Galois key generation. The decompose processes are the most time-consuming process in the key-switching operation. If I do not use the hoisted Galois key generation method, the number of the decompose processes is the same as the total number of GalToGal and PubToGal operations. For example, in the two-level Galois key generation for the ResNet-20, it takes 35.0s to generate all Galois keys using 4-ary level-1 Galois keys if I do not use the hoisted method. If I use the hoisted method, it takes 24.4s to generate all Galois keys with the same level-1 Galois keys, which is reduced by 31.4%.

Table 5.5 shows the various performances with the ResNet-20 for the CIFAR-10 dataset when using the conventional system, the two-level Galois key generation system, and the three-level Galois key generation system. As the number of Galois keys generated by the client is reduced to only 15 in the two-level system, the running time required for the client to generate level-1 Galois keys is reduced to 20.9s, and the size of the total level-1 Galois keys to be transmitted is also reduced to 6.0GB. It shows that the computational and communication burden of the client is significantly reduced,

Table 5.5: Simulation results with various Galois key system with ResNet-20 for CIFAR-10 dataset

	Running time by client(sec)	Communication cost (GB)	Running time by server (offline, sec)	Memory usage by server (offline, GB)	Running time by server (online, sec)	Memory usage by server (online, GB)
Conventional	368.5	105.6	-	105.6	-	105.6
Two-level system	20.9	6.0	-	6.0	24.4	105.6
Three-level system (w/o offline)	12.1	3.4	-	3.4	44.0	105.8
Three-level system (w/ offline)	12.1	3.4	3.5	6.2	25.3	105.8

Table 5.6: Simulation results with various Galois key system with ResNet-18 for ImageNet dataset

	Running time by client(sec)	Communication cost (GB)	Running time by server (offline, sec)	Memory usage by server (offline, GB)	Running time by server (online, sec)	Memory usage by server (online, GB)
Conventional	786.0	197.6	-	197.6	-	197.6
Two-level system	18.4	4.6	-	4.6	22.0	197.4
Three-level system (w/o offline)	15.7	3.9	-	3.9	42.1	199.0
Three-level system (w/ offline)	15.7	3.9	1.8	6.1	22.0	198.9

and a large part of computations goes to the high-performance server, which balances the computation tasks and the communication amount according to the environment. As the number of Galois keys generated by the client is reduced to 8 in the three-level system, the running time to generate the level-2 Galois keys and the communication amount required for transmission of these keys are more reduced. For the case when services have not yet been requested just after the server received the level-2 Galois keys from the client, the level-1 Galois keys can be generated in advance to reduce the actual required level-0 Galois key generation time for 3.5s and with the required memory of 6.2GB for storing level-2 and level-1 Galois keys. The running time to generate the level-0 Galois keys required for ResNet is reduced to 25.3s with the level-2 and level-1 Galois keys. Therefore, the computational and communication amount in the client and the amount of computation for the online phase required to generate the necessary Galois keys are both improved.

The simulation also investigate the case when using Lee et al.'s idea to implement the ResNet-18 model for the larger ImageNet dataset. Because of the larger image and channel size in the dataset, 617 cyclic shifts are required for the polynomial modulus degree $N = 2^{17}$. Table 5.6 shows the various performances with the ResNet-18 for the ImageNet dataset. The effectiveness of the proposed hierarchical Galois key system can be validated also for the ResNet models with the ImageNet dataset in the same manner as the case of ResNet-20 model for the CIFAR-10 dataset.

5.5 Correctness Proofs

To prove Theorem 5.2.1, I need the following three lemmas. After proving the lemmas, I prove Theorem 5.2.1 using these lemmas.

Lemma 5.5.1. *For any positive numbers $v_0, \cdots, v_L$ and integer $m \geq 1$, let $\alpha_m(j)$ be* *the function defined as*

$$\alpha_m(j) = \min_{0=u_0<u_1<\cdots<u_{m-1}\leq j}\left\{\max\left\{\sum_{i=0}^{u_1-1} v_i, \sum_{i=u_1}^{u_2-1} v_i, \cdots, \sum_{i=u_{m-1}}^{j} v_i\right\}\right\},$$

where I define $\min_{i\in I} f(i) = \infty$ *for* $I = \varnothing$ *and any function* f. *Then,* $\alpha_m(j)$ *is a monotonously increasing function.*

Proof. Assume that $\alpha(j) > \alpha(j+1)$ for some positive numbers $(v_0, \cdots, v_L)$, $m \geq 1$, and j. Let

$$U_{j+1} = \{u_{j+1,0} = 0, u_{j+1,1}, \cdots, u_{j+1,m-1}\}$$

be the set of indices minimizing the value

$$\beta_{j+1}(u_0 = 0, u_1, \cdots, u_{m-1}) = \max\left\{\sum_{i=0}^{u_1-1} v_i, \sum_{i=u_1}^{u_2-1} v_i, \cdots, \sum_{i=u_{m-1}}^{j+1} v_i\right\}.$$

If I replace the last term $\sum_{i=u_{m-1}}^{j+1} v_i$ with $\sum_{i=u_{m-1}}^{j} v_i$, this minimum value decreases as $v_i > 0$, and thus I have $\alpha_m(j+1) \geq \beta_j(u_{j+1,0} = 0, u_{j+1,1}, \cdots, u_{j+1,m-1})$. Because of the definition of $\alpha_m(j)$, I also have $\beta_j(u_{j+1,0} = 0, u_{j+1,1}, \cdots, u_{j+1,m-1}) \geq f(j)$. Therefore, I have $\alpha_m(j) \leq \alpha_m(j+1)$, which contradicts the assumption. □

Lemma 5.5.2. *Let* $f, g : \mathbb{Z} \cap [0, L] \to \mathbb{Z}^+$ *be functions satisfying the following conditions.*

1. $f(0) = g(L) = 0$.

2. $f(L)$, $g(0) > 0$.

3. f *is a monotonously increasing function.*

4. g *is a strictly decreasing function.*

Then, there is a value $\ell \in \mathbb{Z} \cap [0, L)$ *such that* $f(n) \leq g(n)$ *for* n *in* $[0, \ell]$, $f(n) > g(n)$ *for* n *in* $[\ell + 1, L]$, *and* $\min\{g(\ell), f(\ell + 1)\}$ *is the minimum value of* $h(n) = \max\{f(n), g(n)\}$ *in* $[0, L]$.

Proof. Let $s(n) = g(n) - f(n)$, and then $s(n)$ is a strictly decreasing function. Since $s(0) > 0, s(L) < 0$, there is an integer $\ell \in \mathbb{Z} \cap [0, L)$ such that $s(\ell) \geq 0, s(\ell+1) < 0$. Since $s(n)$ is a strictly decreasing function, $s(n) \geq s(\ell) \geq 0$, $f(n) \leq g(n)$ for $n \leq \ell$, and $s(n) \leq s(\ell + 1) < 0$, $f(n) > g(n)$ for $n \geq \ell + 1$. Note that $h(n) = g(n)$ for $n \leq \ell$ and $h(n) = f(n)$ for $n \geq \ell + 1$. Since $g(n)$ is strictly decreasing, $h(n)$ is strictly decreasing in $n \leq \ell$, and thus $h(n) \geq h(\ell) = g(\ell)$. Since $f(n)$ is monotonously increasing, $h(n)$ is monotonously increasing in $n \geq \ell + 1$, and thus $h(n) \geq h(\ell + 1) = f(\ell + 1)$. Therefore, $\min\{g(\ell), f(\ell + 1)\}$ is the minimum value of $h(n) = \max\{f(n), g(n)\}$ in $[0, L]$.

□

Lemma 5.5.3. *For positive numbers $v_0, \cdots, v_L$ and integer $m > 1$, the function $\alpha_m(j)$ is defined as Lemma 5.5.1. Then, the following equation is satisfied as*

$$\alpha_m(L) = \min_{0<u\leq L} \left\{ \max \left\{ \alpha_m(u-1), \sum_{i=u}^{L} v_i \right\} \right\}.$$

Proof. Let $\alpha_{m,u}(j)$ be defined as

$$\alpha_{m,u}(j) = \min_{0<u_1<\cdots<u_{m-2}<u\leq j} \left\{ \max \left\{ \sum_{i=0}^{u_1-1} v_i, \sum_{i=u_1}^{u_2-1} v_i, \cdots, \sum_{i=u}^{j} v_i \right\} \right\}. \tag{5.1}$$

If a set A is decomposed into a union of disjoint sets A_i for index set I, that is, $A = \bigsqcup_{i\in I} A_i$, I have $\min_{x\in A} f(x) = \min_{i\in I}(\min_{x\in A_i} f(x))$. Since the following set formula holds

$$\begin{aligned} &\{(u_1, \cdots, u_{m-1}) : 0 < u_1 < \cdots < u_{m-1} \leq L\} \\ =& \bigsqcup_{u=1}^{L} \{(u_1, \cdots, u_{m-2}, u) : 0 < u_1 < \cdots < u_{m-2} < u \leq L\}, \end{aligned}$$

we have $\alpha_m(L) = \min_{0\leq u\leq L} \alpha_{m,u}(L)$. I want to show that for all u, $0 \leq u \leq L$, I have

$$\alpha_{m,u}(L) = \max \left\{ \alpha_{m-1}(u-1), \sum_{i=u}^{L} v_i \right\}. \tag{5.2}$$

If I fix u in (5.2), the term $\sum_{i=u}^{L} v_i$ in the right side of (5.2) is a constant. I now consider the two cases; one case is when $\alpha_{m-1}(u-1) \leq \sum_{i=u}^{L} v_i$, and the other case is when $\alpha_{m-1}(u-1) > \sum_{i=u}^{L} v_i$.

1. $\alpha_{m-1}(u-1) \leq \sum_{i=u}^{L} v_i$: For the left side of (5.2), I know $\alpha_{m,u}(L) \geq \sum_{i=u}^{L} v_i$ since the term in the min function in (5.1) is always larger than $\sum_{i=u}^{L} v_i$ when $j = L$. If $\alpha_{m-1}(u-1) \leq \sum_{i=u}^{L} v_i$, there is an index set $\{u_0 = 0, u_1, u_2, \cdots, u_{m-2}\}$ such that

$$\max\left\{\sum_{i=0}^{u_1-1} v_i, \sum_{i=u_1}^{u_2-1} v_i, \cdots, \sum_{i=m-2}^{u-1} v_i\right\} \leq \sum_{i=u}^{L} v_i.$$

   With this indices, the term in the min function in (5.1) is $\sum_{i=u}^{L} v_i$ when $j = L$. Thus, $\alpha_{m,u}(L) \leq \sum_{i=u}^{L}$, and thus $\alpha_{m,u}(L) = \sum_{i=u}^{L}$. Since the right side of (5.2) is $\sum_{i=u}^{L}$, (5.2) holds.

2. $\alpha_{m-1}(u-1) > \sum_{i=u}^{L} v_i$: In this case, for all index sets $\{u_0 = 0, u_1, \cdots, u_{m-2}\}$ such that $u_0 < u_1 < \cdots < u_{m-2} \leq u-1$, the following inequality holds

$$\max\left\{\sum_{i=0}^{u_1-1} v_i, \sum_{i=u_1}^{u_2-1} v_i, \cdots, \sum_{i=m-2}^{u-1} v_i\right\} > \sum_{i=u}^{L} v_i.$$

   Then (5.1) for $j = L$ holds even if I remove the last term $\sum_{i=u}^{L} v_i$, where the right term becomes exactly $\alpha_{m-1}(u-1)$. Since the right side of (5.2) is $\alpha_{m-1}(u-1)$, (5.2) holds.

$\square$

Theorem 5.5.4. *(*Restatement of Theorem 5.2.1*) Algorithm 26 outputs the list of indices* $I = \{u_0 = 0, u_1, \cdots, u_{\mathtt{dnum}-1}\}$ *such that* $0 = u_0 < u_1 < \cdots < u_{\mathtt{dnum}-1} \leq L$ *and minimizes the following formula*

$$\max\left\{\sum_{i=u_0}^{u_1-1} \log q_i, \sum_{i=u_1}^{u_2-1} \log q_i, \cdots, \sum_{i=u_{\mathtt{dnum}-1}}^{L} \log q_i\right\}.$$

Proof. I now prove Theorem 5.5.4 by induction with $\texttt{dnum}$.

1. $\texttt{dnum} = 1$: The equality trivially holds.

2. $\texttt{dnum} > 1$: Assume that the equality holds for $\texttt{dnum} - 1$. If I set $v_i = \log q_i$, $\alpha_{\texttt{dnum}-1}(j)$ is a monotonously increasing function. If I define $\gamma(j) = \sum_{i=j+1}^{L} \log q_i$, $\gamma(j)$ is a strictly decreasing function as $v_i > 0$ for all i. Since the function $\alpha_{\texttt{dnum}-1}$ and γ satisfy the conditions in Lemma 5.5.2, there is an integer $\ell \in \mathbb{Z} \cap [0, L)$ such that $\alpha_{\texttt{dnum}-1}(n) \leq \gamma(n)$ for $n \in [0, \ell]$, $\alpha_{\texttt{dnum}-1}(n) > \gamma(n)$ for $n \in [\ell + 1, L]$, and the minimum value of $h(n) = \max\{\alpha_{\texttt{dnum}-1}(n), \gamma(n)\}$ is $\min\{\gamma(\ell), \alpha_{\texttt{dnum}-1}(\ell + 1)\}$.

   If $j = j_0$ in line 4 of Algorithm 26 satisfies $\alpha_{\texttt{dnum}-1}(j_0) = \gamma(j_0)$, I have $j_0 = \ell$, and the value $\alpha_{\texttt{dnum}-1}(j_0) = \gamma(j_0)$ is the minimum value of $h(n)$. If $\alpha_{\texttt{dnum}-1}(j_0) > \gamma(j_0)$, I have $j_0 \geq \ell + 1$. I can find ℓ at the point when $\alpha_{\texttt{dnum}-1}(j) \leq \gamma(j)$ first holds as j decreases from j_0 by 1. Then, the value $\min\{\alpha_{\texttt{dnum}-1}(\ell + 1), \gamma(\ell)\}$ is the minimum value of $h(n)$. If $\alpha_{\texttt{dnum}-1}(j_0) < \gamma(j_0)$, I have $j_0 \leq \ell$. I can find $\ell' = \ell + 1$ at the point when $\alpha_{\texttt{dnum}-1}(j) > \gamma(j)$ first holds as j increases from j_0 by 1. Then, the value $\min\{\alpha_{\texttt{dnum}-1}(\ell'), \gamma(\ell' - 1)\} = \min\{\alpha_{\texttt{dnum}-1}(\ell + 1), \gamma(\ell)\}$ is the minimum value of $h(n)$.

   For all cases, I can find the minimum value of $h(n)$. This minimum value is actually the minimum value of $f(u_0 = 0, \cdots, u_{\texttt{dnum}-1})$ by Lemma 5.5.3, which proves the theorem.

□

Next, I prove Theorem 5.2.2.

Theorem 5.5.5. (Restatement of Theorem 5.2.2) *The output of Algorithm 28 is a valid Galois key for the rotation operation for cyclic shift $r + r'$.*

Proof. The proof for $\ell' = \ell + 1$ is given. The proof for $\ell' > \ell + 1$ is the same as the case of $\ell' = \ell + 1$ by Theorem 2.2. A Galois key

$$\mathsf{gk}_r^{(\ell)} = \{(b_{r,i}^{(\ell)}, a_{r,i}^{(\ell)})\}_{i=0,\cdots,\mathtt{hdnum}_\ell - 1} \in (R_{Q_\ell P_\ell^2})^{\mathtt{hdnum}_\ell}$$

for cyclic shift r in the key level ℓ is valid if and only if

$$b_{r,i}^{(\ell)} + a_{r,i}^{(\ell)} \cdot s = P_\ell \cdot \hat{Q}_\ell \cdot [\hat{Q}_\ell^{-1}]_{Q_\ell} \cdot s(X^{5^r}) + e_{r,i}^{(\ell)}$$

for small errors $e_{r,i}^{(\ell)}$. If I perform

$$(\tilde{b}_{r,i}, \tilde{a}_{r,i}) \leftarrow (b_{r,i}^{(\ell)}(X^{5^{r'}}), a_{r,i}^{(\ell)}(X^{5^{r'}}))$$

as in line 2 of Algorithm 28, I have

$$\begin{aligned}
\tilde{b}_{r,i} + \tilde{a}_{r,i} \cdot s(X^{5^{r'}}) &= b_{r,i}^{(\ell)}(X^{5^{r'}}) + a_{r,i}^{(\ell)}(X^{5^{r'}}) \cdot s(X^{5^{r'}}) \\
&= P_\ell \cdot \hat{Q}_\ell \cdot [\hat{Q}_\ell^{-1}]_{Q_\ell} \cdot s((X^{5^{r'}})^{5^r}) + e_{r,i}^{(\ell)}(X^{5^{r'}}) \\
&= P_\ell \cdot \hat{Q}_\ell \cdot [\hat{Q}_\ell^{-1}]_{Q_\ell} \cdot s(X^{5^{r+r'}}) + e_{r,i}^{(\ell)}(X^{5^{r'}}).
\end{aligned}$$

If I perform the key-switching operation to $(\tilde{b}_{r,i}, \tilde{a}_{r,i})$ from $s(X^{5^{r'}})$ to $s(X)$ as in line 3 of Algorithm 28, the output $(b_{r+r',i}^{(\ell)}, a_{r+r',i}^{(\ell)})$ satisfies

$$\begin{aligned}
b_{r+r',i}^{(\ell)} + a_{r+r',i}^{(\ell)} \cdot s &= \tilde{b}_{r,i} + \tilde{a}_{r,i} \cdot s(X^{5^{r'}}) + e'_{r,i} \\
&= P_\ell \cdot \hat{Q}_\ell \cdot [\hat{Q}_\ell^{-1}]_{Q_\ell} \cdot s(X^{5^{r+r'}}) + e_{r,i}^{(\ell)}(X^{5^{r'}}) + e'_{r,i}
\end{aligned}$$

for small errors $e'_{r,i}$ generated from the key-switching operation. Since $e_{r,i}^{(\ell)}(X^{5^{r'}}) + e'_{r,i}$ is a small polynomial,

$$\mathsf{gk}_{r+r'}^{(\ell)} = \{b_{r+r',i}^{(\ell)}, a_{r+r',i}^{(\ell)}\}_{i=0,\cdots,\mathtt{hdnum}_\ell - 1}$$

is a valid Galois key for cyclic shift $r + r'$ in the key level ℓ.

To use the Galois key $\mathsf{gk}_{r'}^{(\ell')} \in (R_{Q_{\ell'} P_{\ell'}}^2)^{\mathtt{hdnum}_{\ell'}}$ in this key-switching operation, the modulus of the ciphertext to be key-switched should be a divisor of $Q_{\ell'}$. Note that $Q_\ell = P_{\ell-1} Q_{\ell-1}$ for all ℓ. If the key level ℓ is less than ℓ', $P_\ell Q_\ell$ is a divisor of $Q_{\ell'}$. Therefore, $(\tilde{b}_{r,i}, \tilde{a}_{r,i})$ can be key-switched by $\mathsf{gk}_{r'}^{(\ell')}$.

$\square$

Next, I prove Theorem 5.2.2.

Theorem 5.5.6. (Restatement of Theorem 5.2.3)*The output of Algorithm 29 is a valid Galois key for the rotation operation for cyclic shift r.*

Proof. The proof for $\ell' = \ell + 1$ is given. The proof for $\ell' > \ell + 1$ is the same as the case of $\ell' = \ell + 1$ by Theorem 2.2. A public key $(b, a) \in R_{Q_{k-1}}^2$ is valid if and only if

$$b + a \cdot s = e$$

for small $e \in R_{Q_{k-1}}$. Note that ℓ is less than $k - 1$, and $P_\ell Q_\ell$ is a divisor of Q_{k-1}. If I perform

$$(b', a') \leftarrow ([b(X^{5^r})]_{P_\ell Q_\ell}, [a(X^{5^r})]_{P_\ell Q_\ell}) \in R_{P_\ell Q_\ell}^2$$

as in line 1 of Algorithm 29, I have

$$b' + a' \cdot s(X^{5^r}) = b(X^{5^r}) + a(X^{5^r}) \cdot s(X^{5^r}) = e(X^{5^r}) \in R_{P_\ell Q_\ell}.$$

If I decompose a' into a vector $(a_0, \cdots, a_{\mathtt{hdnum}_{\ell'}-1}) \in R_{P_{\ell'} Q_{\ell'}}^{\mathtt{hdnum}_{\ell'}}$ using ModUp opeation, I have $a_j = [a']_{Q_{\ell',j}} + Q_{\ell',j} \cdot \tilde{e}_j$ for small $\tilde{e}_j$'s, rather than $[a']_{Q_{\ell',j}}$. The reason for this is the fast basis conversion technique [3], which omits the modular reduction by the product of moduli in the CRT merge process to remove the need for transforming to non-RNS representation.

On the other hand, the Galois key

$$\mathsf{gk}_r^{(\ell')} = \{b_{r,j}^{(\ell')}, a_{r,j}^{(\ell')}\}_{j=0,\cdots,\mathtt{hdnum}_{\ell'}-1} \in (R_{Q_{\ell'} P_{\ell'}}^2)^{\mathtt{hdnum}_{\ell'}}$$

for cyclic shift r in the key level ℓ' satisfies

$$b_{r,j}^{(\ell')} + a_{r,j}^{(\ell')} \cdot s = P_{\ell'} \cdot \hat{Q}_{\ell',j} \cdot [\hat{Q}_{\ell',j}^{-1}]_{Q_{\ell',j}} \cdot s(X^{5^r}) + e_{r,j}^{(\ell')}$$

for small errors $e_{r,j}^{(\ell')}$. lines 4-7 compute

$$\sum_{j=0}^{\mathtt{hdnum}_{\ell'}-1} (a_j + [P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}}) \cdot (b_{r,j}^{(\ell')}, a_{r,j}^{(\ell')}) \in R_{Q_{\ell'} P_{\ell'}}^2,$$

which I denote as $(\bar{b}_{r,i}^{(\ell)}, \bar{a}_{r,i}^{(\ell)})$. The term $a_j + [P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}}$ can be arranged as

$$
\begin{aligned}
&a_j + [P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}} \\
=&[a']_{Q_{\ell',j}} + Q_{\ell',j} \cdot \tilde{e}_j + [P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}} \\
=&[a' + P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}} + Q_{\ell',j} \cdot e'_{i,j} + Q_{\ell',j} \cdot \tilde{e}_j \\
=&[a' + P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}} + Q_{\ell',j} \cdot (e'_{i,j} + \tilde{e}_j),
\end{aligned}
$$

where $Q_{\ell',j} \cdot e'_{i,j}$ denotes the difference between $[a']_{Q_{\ell',j}} + [P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}}$ and $[a' + P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}}$. The difference occurs because these operations are performed in $R_{P_{\ell'}Q_{\ell'}}$, rather than $R_{Q_{\ell',j}}$. Since $[a']_{Q_{\ell',j}}$ and $[P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}}$ are positive integer polynomials less than $Q_{\ell',j}$, $[a']_{Q_{\ell',j}} + [P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}}$ is a positive integer polynomial less than $2Q_{\ell',j}$. The polynomial $[a' + P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}}$ is a positive integer polynomial less than $Q_{\ell',j}$, and $[a']_{Q_{\ell',j}} + [P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}}$ and $[a' + P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}}$ are the same in modulo $Q_{\ell',j}$. Thus, the difference is the multiple of $Q_{\ell',j}$ so that it has the form of $Q_{\ell',j} \cdot e'_{i,j}$, and $e'_{i,j}$ is polynomials having zero or one as its coefficients.

If I compute $\bar{b}_{r,i}^{(\ell)} + \bar{a}_{r,i}^{(\ell)} \cdot s$, I have

$$\begin{aligned}
&\bar{b}_{r,i}^{(\ell)} + \bar{a}_{r,i}^{(\ell)} \cdot s \\
&= \sum_{j=0}^{\mathtt{hdnum}_{\ell'}-1} (a_j + [P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}}) \cdot (b_{r,j}^{(\ell')} + a_{r,j}^{(\ell')} \cdot s) \\
&= \sum_{j=0}^{\mathtt{hdnum}_{\ell'}-1} ([a' + P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}} + Q_{\ell',j} \cdot (e'_{i,j} + \tilde{e}_j)) \cdot \\
&\qquad\qquad (P_{\ell'} \cdot \hat{Q}_{\ell',j} \cdot [\hat{Q}_{\ell',j}^{-1}]_{Q_{\ell',j}} \cdot s(X^{5^r}) + e_{r,j}^{(\ell')}) \\
&= P_{\ell'} \cdot \left(\sum_{j=0}^{\mathtt{hdnum}_{\ell'}-1} [a' + P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}} \right. \\
&\qquad\qquad \left. \cdot \hat{Q}_{\ell',j} \cdot [\hat{Q}_{\ell',j}^{-1}]_{Q_{\ell',j}} \cdot s(X^{5^r}) \right) \\
&\quad + ([a' + P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}} + Q_{\ell',j} \cdot (e'_{i,j} + \tilde{e}_j)) \cdot e_{r,j}^{(\ell')} \\
&= P_{\ell'} \cdot (a' + P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}) \cdot s(X^{5^r}) + E_{i,j} \\
&= P_{\ell'} \cdot (-b' + P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}} \cdot s(X^{5^r}) + e(X^{5^r})) + E_{i,j},
\end{aligned}$$

where $E_{i,j}$ denotes the remaining error term.

If I perform ModDown operation to $(\bar{b}_{r,i}^{(\ell)}, \bar{a}_{r,i}^{(\ell)})$ to divide the terms and the modulus by $P_{\ell'}$ and add $(b', 0)$, which I denotes $(b_{r,i}^{(\ell)}, a_{r,i}^{(\ell)})$, I have

$$\begin{aligned}
&b_{r,i}^{(\ell)} + a_{r,i}^{(\ell)} \cdot s \\
&= P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}} \cdot s(X^{5^r}) + e(X^{5^r}) + \lfloor P_{\ell'}^{-1} \cdot E_{i,j} \rceil.
\end{aligned}$$

Since I choose $P_{\ell'}$ as larger than $Q_{\ell',j}$ for all j, the term $\lfloor P_{\ell'}^{-1} \cdot E_{i,j} \rceil$ is only small error. Thus,

$$\mathsf{gk}_r^{(\ell)} = \{b_{r,i}^{(\ell)}, a_{r,i}^{(\ell)}\}_{i=0,\cdots,\mathtt{hdnum}_\ell - 1} \in (R_{Q_\ell P_\ell}^2)^{\mathtt{hdnum}_\ell}$$

is a valid Galois key for cyclic shift r in the key level ℓ.

□

Chapter 6

MODIFIED SHELL SORT FOR FHE

When processing large amounts of ciphertexts in cloud systems, it is frequently required to process the sorted data rather than unaligned data. Thus, one of the most essential operations on the FHE data is the sorting algorithm, which is generally used as a subroutine algorithm of many algorithms. However, most sorting algorithms are not suitable for the FHE data. For example, because the quick sort algorithm, one of the most popularly used sorting algorithms, is not oblivious, it cannot be used on the FHE data. Although numerous studies have been conducted to render the quick sort algorithm oblivious, its running time complexity becomes $O(n^2)$, where n is the input array length. Its actual running time is even longer than that of the bubble sort, which is considered to have the longest running time among all the known sorting algorithms. Therefore, modifying conventional sorting algorithms to make them suitable for the FHE data is necessary. Several studies have been conducted for this purpose [13, 15, 31].

Since the oblivious sorting algorithm can be applied for encrypted data with the FHE, Emmadi et al. [31] compared several oblivious algorithms for sorting the FHE data. I can divide sorting algorithms for the FHE data into two classes of oblivious sorting algorithms: in-place algorithm and recursive algorithm. The bubble sort and insertion sort are basic in-place oblivious algorithms, and the bitonic sort and odd-even merge sort are recursive oblivious algorithms. The recursive oblivious algorithms are much better than the in-place oblivious algorithms in the aspect of both the asymptotic performance and practical performance.

However, the recursive algorithms may have inefficiency in some cases. Since many function calls are caused in the recursive algorithms and the amount of memory for the ciphertext array is quite big, the total transmission of data in the memory bus must be somewhat large. When the bandwidth of the memory bus is restricted, this transmission time can be a bottleneck for sorting with encrypted data. This situation can occur in lightweight IoT devices, whose memory or bandwidth cannot be large enough. For this reason, it is desirable to devise an efficient in-place sorting algorithm for the FHE data. The Shell sort [69, 70], which is one of the oldest sorting algorithms, is the generalized version of the insertion sort. The Shell sort algorithm is an in-place algorithm, which is fast and easy to implement, and thus, many systems use it as a sorting algorithm.

It is known that Shell sort uses insertion sort as a subroutine algorithm, and insertion sort can be performed on the FHE data [14, 15]. However, the Shell sort should be modified to be used in the FHE setting. If I do not allow any error in sorting, then insertion sort is expected to be quite conservative, i.e., the number of operations for sorting must be set for the worst case, because the insertion sort algorithm in the FHE setting is an oblivious algorithm. Thus, if I use insertion sort in the Shell sort, the running time complexity of Shell sort in the FHE setting must be $O(n^2)$, which makes the use of Shell sort ineffective. Therefore, it is important to devise a sorting algorithm that is better than the Shell sort on the FHE data in terms of running time complexity.

Goodrich [39] suggested an asymptotically optimal randomized oblivious Shell sort. He proved that its running time complexity is $O(n \log n)$ and sorting failure probability (SFP) is $O(1/n^b)$ for some constant $b \geq 1$, where n is the length of an array. While it is pretty efficient in the asymptotic sense, there are two points to be considered. First, the analytically induced SFP is an inverse polynomial of the length of an array. When I sort the array having a small length with this algorithm, the induced SFP may not be the practically allowable value. Further, the inverse polynomial SFP is not considered a small probability in the asymptotic sense. Since users are often conservative with the SFP in the sorting, the exponentially decaying SFP is more desirable. Second, it can be inefficient for the array of the small length. For lowering the SFP, many processes are required in the randomized Shell sort. This causes rather large additional operations for an array with a small length. Considering that the running time of the homomorphic operations in the FHE is quite large, sorting a large number of encrypted data is not a practical situation yet. Thus, it can be desirable to devise an oblivious variant of Shell sort which is practically efficient and has truly negligible SFP, which is independent of the array length.

On the other hand, it is known [14] that I can reduce the running time of insertion sort on the FHE data by allowing very small sorting failure probability using what is known as the window technique. According to this technique, in each insertion sort, instead of inserting the ith element into the subarray of $(a_1, a_2, \cdots, a_{i-1})$, I insert the ith element into the subarray $(a_{i-k}, a_{i-k+1}, \cdots, a_{i-1})$ of length k, called the window length, immediately to the left of the ith element. I call this subarray a “window” with window length k. This technique is used to reduce the number of bootstrapping operations in [14]. Since the insertion sort is the subroutine algorithm for the Shell sort, the window technique is adequate to be applied to the Shell sort. However, the effective application of the window technique in the Shell sort for homomorphic encryption has not been proposed.

In this section, I devise a method to modify the Shell sort in the FHE setting using the window technique, which is proved to be effective in the theoretical aspect and the practical aspect. It is referred to as a “modified Shell sort”. The window technique in [15] is applied to each subroutine insertion sort in the modified Shell sort for FHE setting. I note that the role of the window technique in the algorithm is different from the original use of the window technique. The proposed algorithm does not reduce the bootstrapping itself compared to the number of the homomorphic gates, but I reduce the number of the comparison operations with the window technique. For this reason, the homomorphic comparison operation in the proposed sorting algorithm does not generate a comparison error.

For theoretical view, the running time complexity of the modified Shell sort is $O(n^{3/2}\sqrt{\alpha + \log\log n})$ with SFP $2^{-\alpha}$ when the gap sequence is powers-of-two, which is close to the average-case time complexity $O(n^{3/2})$ of the original Shell sort. The value of α is the additional parameter that controls the trade-off between the running time and the SFP. This trade-off is quite effective because the SFP is decreased exponentially with α but the running time is proportional only to $\sqrt{\alpha}$. To this end, I use the exact distribution of window lengths of subarrays in each gap for successful sorting in the Shell sort. If the length of the subarray for the insertion sort in some gap is s, it is discovered that the average of the required window length for successful sorting is proportional to $\sqrt{s}$, and the right tail of its probability distribution is very thin. In the sorting process, the window length is provided as a constant multiple of $\sqrt{s}$, which ensures a negligible SFP. If the window length is close to $\beta\sqrt{s}$, the SFP decays as $e^{-\beta^2}$, which signifies a very fast-decaying function. Therefore, with a fixed negligible SFP, I can set a small window length so that the running time is asymptotically faster than that of the naive version of the Shell sort on the FHE data.

For the practical view, the running time of the modified Shell sort is effectively reduced even in the small arrays, compared to the basic in-place sorting algorithms on the FHE data, bubble sort, and insertion sort.

In this work, I address only the gap sequence of powers of two in the analysis of the modified Shell sort, i.e., $2^h, h = 1, 2, 3, \cdots$. Although this gap sequence is not optimal in terms of running time complexity, I first analyze the running time complexity of the modified Shell sort on the FHE data, which is important for the FHE in cloud systems. The performance of the modified Shell sort is numerically compared with the cases of near-optimal window lengths obtained through convex optimization and

Ciura's optimal gap sequence [27], which was evaluated numerically as an optimal gap sequence in the non-FHE settings. Although I do not analyze this case, the method of deriving the near-optimal window length in the modified Shell sort functions well for Ciura's optimal gap sequence.

The convex optimization method is suggested to derive a tighter window length. In other words, the window length obtained by the convex optimization method makes the running time of the modified Shell sort to be less than that of the case employing the analytical method in the modified Shell sort. The running time of the proposed modified Shell sort is compared with that of the conventional algorithms with the TFHE scheme, and it is shown that this modified Shell sort has the best performance in running time among in-place sorting algorithms on the TFHE scheme.

Thus, my contributions are summarized as follows:

- I propose a modified Shell sort with an additional parameter α on the FHE data, and derive its theoretical trade-off between the running time complexity $O(n^{3/2}\sqrt{\alpha + \log\log n})$ and the SFP $2^{-\alpha}$ when the gap sequence is power-of-two sequence.
- The near-optimal window length of each gap in the modified Shell sort is derived via the convex optimization technique.
- The numerical simulation with TFHE homomorphic encryption scheme is performed, and the modified Shell sort with Ciura's gap sequence is proven to have the best running time performance in the practical situation among the in-place sorting algorithms for the FHE setting.

6.1 Modification of Shell Sort over FHE

As insertion sort can be performed on the FHE data, the Shell sort, which uses the insertion sort as a subroutine algorithm, can also be performed on the FHE data. The

Algorithm 36 FHEShellSort($\mathtt{A}[1:n]$)

```
Input : An encrypted array $\mathtt{A}[1:n]$ with $n$ elements, gap sequence $G[1:p]$ with
        decreasing order, and $\alpha$
Output: Sorted array of encrypted data $\mathtt{A}[1:n]$ without sorting failure
for $\ell \leftarrow 1$ to $p$ do
    $g \leftarrow G[\ell]$
    for $i \leftarrow g+1$ to $n$ do
        for $j \leftarrow \lceil \frac{i}{g} \rceil - 1$ to $1$ do
            SortTwo($\mathtt{A}[i-gj], \mathtt{A}[i]$)
        end
    end
end
```

Shell sort using the FHE variant insertion sort as subroutine algorithms is shown in Algorithm 36. However, if the Shell sort is to be employed without any sorting failure in the FHE setting, it is expected to be pretty conservative. In other words, as I need to consider the worst case for each gap, its running time complexity becomes $O(n^2)$, which does not provide any advantage in comparison with a simple insertion sort. This is because the FHE variant insertion sort cannot be performed adaptively with the intermediate situations. Thus, designing the Shell sort with a negligible SFP and a running time complexity close to the original average-case time complexity is necessary.

To this end, I employ the window technique [14, 15] in the Shell sort. During the insertion sort in each gap, instead of searching the position of each element in the whole partially sorted array, I search for its position in the partially sorted subarray of a certain window length, located to the left of the pivot element, as shown in Fig. 6.1. Fig. 6.1 shows an example of the modified Shell sort using the window technique, where the gap is 4 and the window length is 2. Subarrays consisting of elements that are separated by the gap are sorted using the insertion sort. To sort each subarray, it is compared only with the elements that are located to its left, within a distance equal to the window length from the pivot element to be inserted, which is called the modified Shell sort.

The proposed modified Shell sort is described in Algorithm 37. As the minimum and maximum functions can be computed without knowing their plaintext in the FHE setting, neither of the operations in Algorithm 37 require any knowledge of the contents of elements in the array $A[i]$. Thus, Algorithm 37 can be executed in the FHE setting. In designing this algorithm, deciding the window length in each gap for successfully sorting each subarray in the Shell sort for the given SFP $2^{-\alpha}$ is not a trivial problem. Along with the design of the window length for each gap, I propose a modified Shell sort with an additional parameter α.

It is proved that the running time complexity of the modified Shell sort is determined to be $O(n^{3/2}\sqrt{\alpha + \log\log n})$ with an SFP of $2^{-\alpha}$ for powers-of-two gap sequence, which consists of all powers of two less than the length of the array. Note that the average-case time complexity of the classical Shell sort with powers-of-two gap sequence is $O(n^{3/2})$ [32]. The parameter α is determined only from the SFP, regardless of the input length n. In fact, α is considerably smaller than n and should be larger than or equal to $\sqrt{6}\log e - 1 \simeq 2.534$, the derivation of which is provided in a subsequent section of this dissertation. It is noted that the proposed modified Shell sort considers the trade-off between the running time complexity and the SFP.

Before analyzing the running time complexity and the sorting failure probability of the modified Shell sort with power-of-two gap sequence, I introduce the main idea of the analysis to help readers to understand the following theorems and lemmas.

Lemma 6.2.1 and 6.2.2 deal with the useful properties of the intermediate arrays to make it easy to induce the exact number of acceptable arrays for a certain window length, which is dealt with in Lemma 6.2.3 and Theorem 6.2.4. Theorem 6.2.4 is the special case of Lemma 6.2.3 that matches my aim.

While the number of acceptable arrays is represented with some binomial coefficients, the resultant running time complexity has to be represented with some analytic

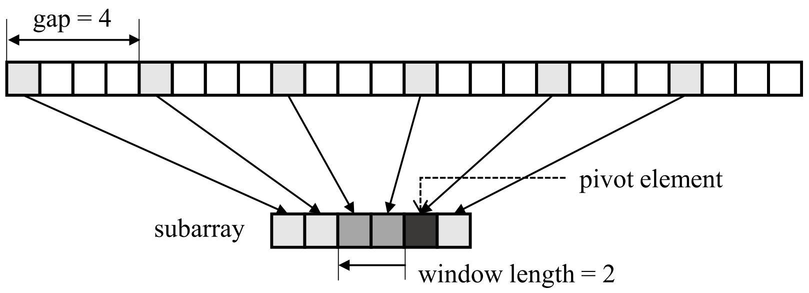


Figure 6.1: Modified Shell sort using the window technique.

function. To this end, Lemma 6.2.5 and 6.2.6 relate some formulas with binomial coefficients with some exponential function. With the help of these lemmas, Theorem 6.2.7 suggests the running time complexity and the sorting failure probability of the modified Shell sort.

Algorithm 37 ModifiedShellSort($A[1:n], \alpha$)

```
Input  : An encrypted array A[1 : n] with n elements, gap sequence G[1 : p] with
         decreasing order, and α
Output: Sorted array of encrypted data A[1 : n] with SFP $2^{-\alpha}$

$c \leftarrow \alpha + 1 + \log\log n$
for $\ell \leftarrow 1$ to $p$ do
    $g \leftarrow G[\ell]$
    $k \leftarrow \min\left\{\left\lceil\sqrt{\lceil\frac{n}{2g}\rceil\cdot(c+\ell)\cdot\frac{1}{\log e}}\right\rceil, \left\lceil\frac{n}{2g}\right\rceil\right\}$
    for $i \leftarrow g+1$ to $n$ do
        $u \leftarrow \min\left\{k, \lceil\frac{i}{g}\rceil - 1\right\}$
        for $j \leftarrow u$ to $1$ do
            SortTwo($A[i-gj], A[i]$)
        end
    end
end
```

Remark. I assume that the modified Shell sort is performed with the bootstrapped homomorphic gate, a homomorphic boolean gate followed by the bootstrapping. The bootstrapping always removes the noise in the ciphertext, and thus I do not have to consider the noise amplification in the ciphertext when processing any homomorphic evaluations. In addition, the ciphertext size does not grow in the fully homomorphic encryption scheme, and I also do not have to consider the amplification of the ciphertext size. Thus, I focus on the validity of the modified Shell sort itself in the following analysis. If I use the leveled homomorphic encryption instead of the FHE, the analysis for the noise growth or the ciphertext size growth will be needed, and this analysis becomes an interesting future work.

6.2 Analysis of Modified Shell Sort

6.2.1 Probability Distribution of Required Window Length

In this subsection, I derive the probability distribution of required window length in each gap required for successfully sorting each subarray in the Shell sort. This probability distribution is essential in determining the window length of each gap in the modified Shell sort, because the properties of the tail of the probability distribution must be used to obtain the required window length.

The array of n elements is denoted by its index vector $(a_1, a_2, \cdots, a_n)$, which is a permuted vector of $(1, 2, \cdots, n)$. If I handle the real data, I map each datum to its respective index in $\{1, 2, \cdots, n\}$. Moreover, I assume that n is an even integer. If n is odd, the same analysis can be applied, with an additional dummy element inserted in the rightmost position with the largest element. Several lemmas are needed for devising the main theorem of the probability distribution for the required window length.

Before obtaining the probability distribution, the meaning of the probability distribution in this subsection has to be clarified. For each permutation of $(1, 2, \cdots, n)$, the required window length is defined as the minimum window length such that the insertion sort with the window length returns a perfectly sorted array. The required window length is a random variable when the sample space is the set of all permutations of $(1, 2, \cdots, n)$ or its subset. For analyzing the modified Shell sort, I am interested in the case when the sample space is the set of all permutations $(a_1, a_2, \cdots, a_n)$ of $(1, 2, \cdots, n)$ which satisfy $a_i < a_{i+2}$.

While the analysis of the conventional Shell sort is performed for an average number of operations, the analysis of the window length in the modified Shell sort involves the maximum number of insertion operations for each subarray.

It is assumed that the gap sequence is powers of two, i.e., $2^{\lfloor \log n \rfloor}, 2^{\lfloor \log n \rfloor - 1}$, $\cdots, 2^2, 2, 1$. With this gap sequence, each subarray that is sorted using insertion sort has the following structure. The elements in odd positions of the subarray for a gap 2^h are already sorted and the elements in even positions are also sorted for a gap 2^h using the previous insertion sort for a gap 2^{h+1}. I analyze the insertion sort under this special situation.

The following Lemma 6.2.1 and Lemma 6.2.2 suggest that the required window length of a permuted vector $(a_1, a_2, \cdots, a_{2m})$ of $(1, 2, \cdots, 2m)$ is equal to the maximum distance of the current position and the right position in this situation. After Lemma 6.2.1 and Lemma 6.2.2, I identify these two notions as equivalent notions.

Lemma 6.2.1. *Let* $\mathbf{a} = (a_1, a_2, \cdots, a_{2m})$ *be a subarray in each gap in the Shell sort with a gap sequence* 2^h*, which is permuted from* $(1, 2, \cdots, 2m)$*, satisfying* $a_i < a_{i+2}$ *for* $i = 1, 2, \cdots, 2m-2$*. Let* $M(\mathbf{a}) = \max_{1 \le i \le 2m} |a_i - i|$*. Then there exists an even integer* j *and an odd integer* k *such that*

$$M(\mathbf{a}) = |a_j - j| = |a_k - k|$$

and $(a_j - j)(a_k - k) \le 0$*.*

Proof. Let M_1, M_2, M_3, and M_4 be defined as

$$M_1 = \max_{1 \le i \le m} (a_{2i-1} - (2i-1))$$

$$M_2 = -\min_{1 \le i \le m} (a_{2i-1} - (2i-1))$$

$$M_3 = \max_{1 \le i \le m} (a_{2i} - 2i)$$

$$M_4 = -\min_{1 \le i \le m} (a_{2i} - 2i).$$

It is clear that at least one of M_1 and M_2 as well M_3 and M_4 is a non-negative integer. If I establish that $M_1 = M_4$ and $M_2 = M_3$, the lemma can be proved by the following argument. If $M_1 \ge M_2$, I obtain $M(\mathbf{a}) = M_1 = M_4 \ge M_2 = M_3$, and thus, there exist an odd index j and an even index k, such that $M(\mathbf{a}) = a_j - j = -(a_k - k)$. If $M_1 < M_2$, $M(\mathbf{a}) = M_2 = M_3 \ge M_1 = M_4$ holds, then there exist an odd index j and an even index k, such that $M(\mathbf{a}) = -(a_j - j) = a_k - k$. Thus, it is sufficient to prove that $M_1 = M_4$ and $M_2 = M_3$.

To show $M_1 = M_4$ and $M_2 = M_3$, I prove the following four inequalities; $M_1 \ge M_4$, $M_1 \le M_4$, $M_2 \ge M_3$, and $M_2 \le M_3$.

i) Firstly, I show that $M_1 \ge M_4$. Consider an index l, such that $a_{2l} - 2l = \min_{1 \le i \le m}(a_{2i} - 2i)$, which is $-M_4$. I establish this case for $a_{2\ell} = 2m$ or $a_{2\ell} < 2m$.

i)-1 If $a_{2l} = 2m$, l must be m, as $2m$ is the largest element. Thus, I obtain $\min_{1 \le i \le m}(a_{2i} - 2i) = 0$ and $a_{2i} \ge 2i$ for all i, $1 \le i \le m$, which implies that 1 cannot be in the even index and must be in the first index, and $a_1 - 1 = 0$. Therefore, $M_1 = \max_{1 \le i \le m}(a_{2i-1} - (2i-1)) \ge 0 = -\min_{1 \le i \le m}(a_{2i} - 2i) = M_4$.

i)-2 If $a_{2l} < 2m$, I show that $a_{2l} + 1$ must be in the odd index. Let $a_{2l} + 1$ be in the even index; this implies that $a_{2l} + 1 = a_{2l+2}$, because all the elements in the even indices are already sorted. Then, I obtain $a_{2l+2} - (2l + 2) =$

$(a_{2l}+1)-(2l+2) = a_{2l}-2l-1 < a_{2l}-2l$, which is a contradiction to the assumption that $a_{2l}-2l$ is the minimum value, and thus, $a_{2l}+1$ must be in the odd index. Among $\{1, 2, \cdots, a_{2l}-1\}$, $l-1$ elements have to be placed in the even indices in the left-side of a_{2l}. The remaining $a_{2l}-l$ elements must be placed in the odd indices in the increasing order from the first index 1. Thus, the index of $a_{2l}+1$ must be $2(a_{2l}-l)+1$. As $a_{2(a_{2l}-l)+1}-(2(a_{2l}-l)+1) = (a_{2l}+1)-(2(a_{2l}-l)+1) = 2l-a_{2l}$, I obtain $M_1 = \max_{1\leq i\leq m}(a_{2i-1}-(2i-1)) \geq 2l-a_{2l} = -\min_{1\leq i\leq m}(a_{2i}-2i) = M_4$.

ii) I then show that $M_2 \geq M_3$. Consider an index l, such that $a_{2l}-2l = \max_{1\leq i\leq m}(a_{2i}-2i)$, which is M_3. I establish this case for $a_{2\ell}=1$ or $a_{2\ell}>1$.

ii)-1 If $a_{2l} = 1$, l must be 1, as 1 is the smallest element, and therefore, $\max_{1\leq i\leq m}(a_{2i}-2i) = -1$. As $a_{2i}-2i \leq -1$ for all i, $1 \leq i \leq m$, $2m$ cannot belong to the even index. Thus, $2m$ must be in the $(2m-1)$-th index, and $a_{2m-1}-(2m-1) = 1$. Therefore, $M_2 = -\min_{1\leq i\leq m}(a_{2i-1}-(2i-1)) \geq -1 = \max_{1\leq i\leq m}(a_{2i}-2i) = M_3$.

ii)-2 If $a_{2l} > 1$, I show that $a_{2l}-1$ must be in the odd index. Let $a_{2l}-1$ be in the even index. I have $a_{2l}-1 = a_{2l-2}$, because all the elements in the even indices are already sorted. Then, I obtain $a_{2l-2}-(2l-2) = (a_{2l}-1)-(2l-2) = a_{2l}-2l+1 > a_{2l}-2l$, which is a contradiction to the assumption that $a_{2l}-2l$ is the maximum value, and thus, $a_{2l}-1$ must be in the odd index. Among $\{1, 2, \cdots, a_{2l}-2\}$, $l-1$ elements need to be placed in the even indices in the left-side of a_{2l}. The remaining $a_{2l}-l-1$ elements have to be placed in the odd indices from the first index 1. Thus, the index of $a_{2l}-1$ must be $2(a_{2l}-l)-1$. As $a_{2(a_{2l}-l)-1}-(2(a_{2l}-l)-1) = (a_{2l}-1)-(2(a_{2l}-l)-1) = 2l-a_{2l}$, I obtain $M_2 = -\min_{1\leq i\leq m}(a_{2i-1}-(2i-1)) \geq -(2l-a_{2l}) = \max_{1\leq i\leq m}(a_{2i}-2i) = M_3$.

Similarly, I can establish that $M_1 \leq M_4$ and $M_2 \leq M_3$ by swapping the even indices with the odd indices. Therefore, I can prove that $M_3 = \max_{1\leq i\leq m}(a_{2i} - 2i) \geq -\min_{1\leq i\leq m}(a_{2i-1} - (2i-1)) = M_2$, and $M_4 = -\min_{1\leq i\leq m}(a_{2i} - 2i) \geq \max_{1\leq i\leq m}(a_{2i-1} - (2i-1)) = M_1$. Therefore, I establish that $M_1 = M_4$ and $M_2 = M_3$. $\square$

Lemma 6.2.2. *Let* $\mathbf{a} = (a_1, a_2, \cdots, a_{2m})$ *be a subarray in the Shell sort with a gap sequence* 2^h*, which is permuted from* $(1, 2, \cdots, 2m)$ *satisfying* $a_i < a_{i+2}$ *for* $i = 1, 2, \cdots, 2m-2$*. Let* $W(\mathbf{a})$ *be the required minimum window length to sort the subarray successfully. Then, I have*

$$W(\mathbf{a}) = \max_{1\leq i\leq 2m}(i - a_i).$$

Proof. When I insert a_i into the partially sorted subarray, the following scenarios can be given; if $a_i < i$, I require a window length of $i - a_i$, and if $a_i \geq i$, a_i stays in place regardless of the window length.

Consider the first case, where $a_i < i$. First, I assume that i is even. Consider the elements to the left of a_i. From the condition $a_i < a_{i+2}$, it is clear that all the elements in even indices to the left of a_i are less than a_i. As there are $i/2 - 1$ even indices to the left of a_i, the remaining $a_i - i/2$ elements in $\{1, 2, \cdots, a_i - 1\}$ have to be placed in odd indices in increasing order from the leftmost odd index. As the number of odd indices to the left of a_i is $i/2$ and $i/2 > a_i - i/2$, all the elements less than a_i are located to the left of a_i.

Then, it is assumed that i is odd. The proof is almost the same as that for the scenario in which i is even. As there are $(i-1)/2$ odd indices to the left of a_i, the remaining $a_i - (i+1)/2$ elements in $\{1, 2, \cdots, a_i - 1\}$ must be placed in even indices from the first even index, in increasing order. As the number of even indices to the left of a_i is $(i-1)/2$ and $(i-1)/2 > a_i - (i+1)/2$, all the elements less than a_i are located to the left of a_i.

Thus, I prove that all the elements less than a_i are located to the left of a_i. The partially sorted subarray, therefore, must include the elements $\{1, 2, \cdots, a_i - 1\}$ in the indices $\{1, 2, \cdots, a_i - 1\}$ in the appropriate order. This implies that a_i moves to the index a_i, and thus, I require a minimum window length of $i - a_i$.

Consider the second case, in which $a_i \geq i$. It is evident that $i/2 \leq a_i - i/2$, when i is even, and $(i-1)/2 \leq a_i - (i+1)/2$, when i is odd. This implies that all the elements to the left of a_i are less than a_i. Thus, the partially sorted subarray in the indices $\{1, 2, \cdots, i-1\}$ comprises elements smaller than a_i. Therefore, a_i does not move to the left but stays in its position, regardless of the window length. $\square$

From Lemma 6.2.1, it is noted that $M(\mathbf{a})$ is equal to $W(\mathbf{a})$. Lemma 6.2.3 is needed to obtain the exact number of the arrays whose required window length is some non-negative number k. Theorem 6.2.4 corresponds to the conclusion of this subsection, and this can be obtained by only considering the special case of Lemma 6.2.3.

Lemma 6.2.3. *Let $p_k(n, m)$ be the number of distinct arrays $(a_1, a_2, \cdots, a_m)$ of length m, whose elements from $\{1, 2, \cdots, n\}$ are sorted in increasing order, $a_i < a_{i+1}$, and $\max_{1 \leq i \leq m} |a_i - 2i| \leq k$ is satisfied for a positive integer k and $n \geq m$. Let $(b_0, b_1, \cdots)$ and $(c_0, c_1, \cdots)$ be the two arrays defined as*

$$b_0 = c_0 = 0$$

$$b_{i+1} = \begin{cases} b_i + (k+1) & \text{if } i \text{ is even} \\ b_i + (k+2) & \text{if } i \text{ is odd} \end{cases}$$

$$c_{i+1} = \begin{cases} c_i + (k+2) & \text{if } i \text{ is even} \\ c_i + (k+1) & \text{if } i \text{ is odd.} \end{cases}$$

For $2m - k \leq n \leq 2m + k$, I obtain

$$p_k(n, m) = \binom{n}{m} - \sum_{1 \leq b_i \leq m} (-1)^{i+1} \binom{n}{m - b_i} - \sum_{1 \leq c_i \leq m} (-1)^{i+1} \binom{n}{m + c_i}. \tag{1}$$

Proof. It is clear that $p_k(1,1) = 1$ for all $k \geq 1$, and $p_k(n,1) = n$ for $n \leq k+2$. As the element a_m in the last index must be $2m-k \leq a_m \leq 2m+k$ from $\max_{1\leq i\leq m} |a_i - 2i| \leq k$, the following can be determined from the condition $2m-k \leq a_m \leq 2m+k$:

i) For $n < 2m-k$,

$$p_k(n,m) = 0,$$

because the minimum possible value of a_m must be $2m-k$.

ii) For $n > 2m+k+1$,

$$p_k(n,m) = p_k(2m+k,m),$$

because the maximum possible value of a_m must be $2m+k$.

iii) For $2m-k \leq n \leq 2m+k+1$,

I derive the recurrence relation of $p_k(n,m)$ using the following three cases:

iii)-1 For $n = 2m+k+1$,

It is easy to derive that

$$p_k(2m+k+1,m) = p_k(2m+k,m). \tag{2}$$

Note that this case can be included in ii). Although this separation of the case appears unnatural, it enables me to analyze $p_k(n,m)$ well.

iii)-2 For $2m-k+1 \leq n \leq 2m+k$,

If the element in the last index m is n, the elements in the remaining indices should be selected from $\{1,2,\cdots,n-1\}$, and thus, there are $p_k(n-1,m-1)$ possible arrays. If the element in the last index m is not n, the element n cannot be located in one of the indices $\{1,2,\cdots,m-1\}$, because the elements are sorted in increasing order. Thus, $\{1,2,\cdots,n-1\}$ should be located in the indices $\{1,2,\cdots,m\}$, and there are $p_k(n-1,m)$ possible arrays. Therefore, I obtain

$$p_k(n,m) = p_k(n-1,m) + p_k(n-1,m-1). \tag{3}$$

iii)-3 For $n = 2m - k$,

I obtain

$$p_k(2m-k, m) = p_k(2m-k-1, m-1), \tag{4}$$

because the element $2m - k$ must be located in the index m.

Let $q_k(n, m)$ be the right-hand side in (1). Then, I prove that $q_k(2m+k+2, m) = 0$ and $q_k(2m - k - 1, m) = 0$. First, $q_k(2m + k + 2, m)$ can be written as

$$\sum_{i=1}^{j} (-1)^i \binom{2m+k+2}{m+c_i} + \sum_{i=1}^{j} (-1)^{i-1} \binom{2m+k+2}{m-b_{i-1}}. \tag{5}$$

As $(m+c_i)+(m-b_{i-1}) = 2m+k+2$, $\binom{2m+k+2}{m+c_i} = \binom{2m+k+2}{m-b_{i-1}}$ holds, and $q_k(2m+k+2, m)$ is equal to 0.

Next, $q_k(2m - k - 1, m)$ can be written as

$$\sum_{i=1}^{j} (-1)^{i-1} \binom{2m-k-1}{m+c_{i-1}} + \sum_{i=1}^{j} (-1)^{i} \binom{2m-k-1}{m-b_i}. \tag{6}$$

As $(m+c_{i-1})+(m-b_i) = 2m-k-1$, $\binom{2m-k-1}{m+c_{i-1}} = \binom{2m-k-1}{m-b_i}$ holds, and $q_k(2m-k-1, m)$ is equal to 0.

It is proved that $p_k(n, m) = q_k(n, m)$ when $2m - k \leq n \leq 2m + k + 1$. Using the fact that $q_k(n, m)$ is simply a linear combination of binomial coefficients and the property of the binomial coefficients, I easily know that

$$q_k(n, m) = q_k(n-1, m) + q_k(n-1, m-1).$$

When $n = 2m + k + 1$, I know that $q_k(n - 1, m - 1) = 0$ from (5). Thus, I have

$$q_k(2m+k+1, m) = q_k(2m+k, m).$$

When $n = 2m - k$, I know that $q_k(n - 1, m) = 0$ from (5). Thus, I have

$$q_k(2m-k, m) = q_k(2m-k-1, m-1).$$

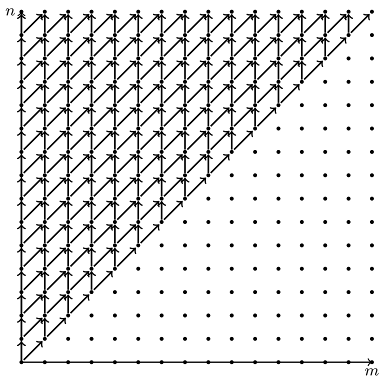
Clearly, $p_k(1, 1) = q_k(1, 1)$. The recurrence relation of $p_k(n, m)$ is identical to that of $q_k(n, m)$ when $2m - k \leq n \leq 2m + k + 1$ and the initial value is also identical.

Therefore, I prove that $p_k(n,m) = q_k(n,m)$ when $2m-k \leq n \leq 2m+k+1$, which proves it. $\square$

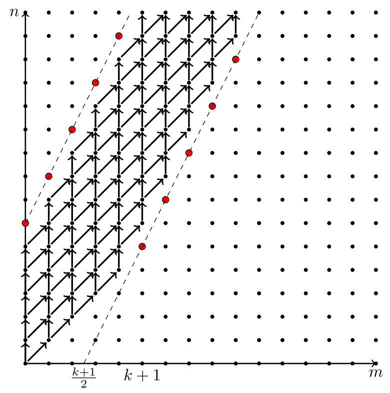
Remark: In order to understand intuitionally the proof of Lemma 6.2.3, I add the additional explanation of proof of Lemma 6.2.3. The recurrence relations (2)-(4) of $p_k(n,m)$ are similar to the Pascal's triangle $\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1}$ shown in Fig. 6.2(a), except that the width of the triangle for $p_k(n,m)$ is limited, as shown in Fig. 6.2(b) as well as (2) and (4). This recursive relation can then be transformed into overlapped Pascal's triangles. Fig. 6.2(c) shows a part of Fig. 6.2(b) near the boundary of the lower dotted line. Here, I only consider the lower dotted line. I then establish that this recursive relation near the boundary in Fig. 6.2(c) is equivalent to the situation of Fig. 6.2(d), which is two overlapped Pascal's triangles, in which $p_k(0,0) = 1$ and $p_k(0,k+1) = -1$.

First, it can be obtained that the values on the dotted line in Fig. 6.2(d) are always 0, because of the symmetry of Pascal's triangles. As adding a 0 does not change the value, the cases of Fig. 6.2(c) and Fig. 6.2(d) are equivalent regarding the area to the left of the dotted line.

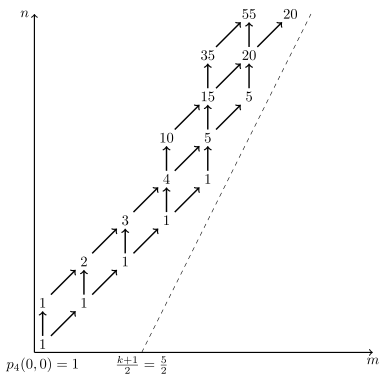
However, the values on both the dotted lines in Fig. 6.2(b) must be 0. To satisfy the other boundary condition $p_k(2m+k+2,m) = 0$ on the upper dotted line in Fig. 6.2(b), I consider another Pascal's triangle translated by $-(k+2)$ with $p_k(0,-(k+2)) = -1$. If I add these three Pascal's triangles P_{-1}, P_0, and P_1 shown in Fig. 6.2(e), there are zero boundary values on the lines from Q_1 to Q_2 and from R_1 to R_2. However, the boundary value after Q_2 or R_2 is not equal to 0. To obtain the boundary values on the lines from Q_2 to Q_3 and from R_2 to R_3, I must add the Pascal's triangles P_2 and P_{-2}. Therefore, I repeat this process, as shown in Fig. 6.2(e). The sequence $\{b_i\}$ in Lemma 6.2.3 is the distance from the initial vertex of P_0 to that of P_i, while $\{c_i\}$ is the distance from the initial vertex of P_0 to that of P_{-i}. The initial value at the initial vertex of P_i is 1 if i is even, and -1 if i is odd. Q_i is defined as the intersection of the boundaries of the two Pascal's triangles starting from the initial vertices of P_{i-1} and P_{-i}, and R_i is



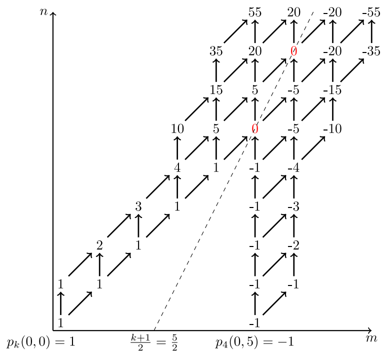
(a) Pascal's triangle for $\binom{n}{m}$



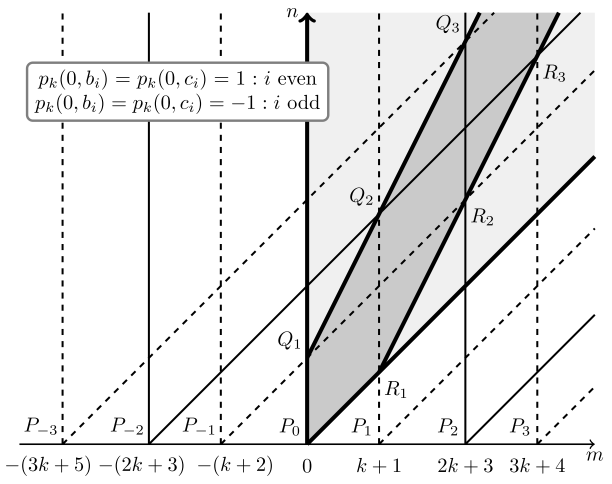
(b) $p_k(n,m)$ with $k = 4$ for $2m - k \leq n \leq 2m + k$



(c) Boundary case of $p_k(n,m)$ in (b)



(d) Equivalent diagram of boundary of $p_k(n,m)$ in (b)



(e) $p_k(n,m)$ using overlapped Pascal's triangles

Figure 6.2: $p_k(n,m)$ using Pascal's triangle.

defined as the intersection of the boundaries of the two Pascal's triangles starting from the initial vertices of P_i and $P_{-(i-1)}$.

It is derived that if the Pascal's triangles P_i's, $i = \cdots, -1, 0, 1, \cdots$, are overlapped, all of the integer points on the half-lines of $\overrightarrow{Q_1Q_2}$ and $\overrightarrow{R_1R_2}$ must be 0s. The integer points on the upper half-line of $\overrightarrow{Q_1Q_2}$ exhibit the form $n = 2m + k + 2$ for all non-negative integers m, and those on the lower half-line of $\overrightarrow{R_1R_2}$ exhibit the form $n = 2m - k - 1$ for all $m \geq k + 1$. First, in the case of the points on the half-line of $\overrightarrow{Q_1Q_2}$, I consider the integer points on $\overline{Q_jQ_{j+1}}$, which can be denoted as $n_1 = 2m_1 + k + 2$ and $b_{i-1} \leq m_1 \leq b_i$. Then, I can only consider Pascal's triangles $P_{-j}, \cdots, P_{j-1}$. Considering the parallel translation of each Pascal's triangle, the overlapped values on the points are defined as

$$\sum_{i=1}^{j}(-1)^i\binom{2m_1+k+2}{m_1+c_i} + \sum_{i=1}^{j}(-1)^{i-1}\binom{2m_1+k+2}{m_1-b_{i-1}}. \tag{5}$$

As $(m_1 + c_i) + (m_1 - b_{i-1}) = 2m_1 + k + 2$, $\binom{2m_1+k+2}{m_1+c_i} = \binom{2m_1+k+2}{m_1-b_{i-1}}$ holds, and (5) is equal to 0.

In the case of the points on the half-line of $\overrightarrow{R_1R_2}$, I consider the integer points on $\overline{R_jR_{j+1}}$, which can be denoted as $n_2 = 2m_2 - k - 1$ and $b_i \leq b_{i+1}$. Then, I can only consider Pascal's triangles $P_{j-1}, \cdots, P_j$. The overlapped values on the points are defined as

$$\sum_{i=1}^{j}(-1)^{i-1}\binom{2m_2-k-1}{m_1+c_{i-1}} + \sum_{i=1}^{j}(-1)^i\binom{2m_2-k-1}{m_1-b_i}. \tag{6}$$

As $(m_2 + c_{i-1}) + (m_2 - b_i) = 2m_2 - k - 1$, $\binom{2m_2-k-1}{m_1+c_{i-1}} = \binom{2m_2-k-1}{m_1-b_i}$ holds, and (6) is also equal to 0.

Therefore, I establish that with respect to the region between the two dotted lines in Fig. 6.2(b), Fig. 6.2(b) is exactly equivalent to the hashed part of Fig. 6.2(e). I obtain $p_k(n, m)$ by adding the values of points of several Pascal's triangles as in (1), where the first term is from the central Pascal's triangle P_0; the second term is from the right-side Pascal's triangles P_i's for the positive integer i; and the third term is from the left-side Pascal's triangles P_{-i}'s for the positive integer i.

From the previous lemmas, I have the following theorem.

Theorem 6.2.4. *Let $C(2m,k)$ be the number of the permutations $\mathbf{a}$ of $\{1,2,\cdots,2m\}$, such that $a_i < a_{i+2}$ for all possible i, and $W(\mathbf{a}) \le k$. Then, I have*

$$C(2m,k) = \binom{2m}{m} - \sum_{1\le b_i \le m} (-1)^{i+1}\binom{2m}{m-b_i} - \sum_{1\le c_i \le m} (-1)^{i+1}\binom{2m}{m-c_i},$$

where b_i and c_i are defined in Lemma 6.2.3.

Proof. As $M(\mathbf{a})$ of the odd indices is equal to that of the even indices from Lemma 6.2.1, I consider only the even indices. Thus, I can consider this situation to be equivalent to the following simple situation; I consider distinct m elements from $\{1,2,\cdots,2m\}$ randomly, sort them in increasing order, and consider $a_i - 2i$ rather than $a_i - i$. Then, $C(2m,k)$ is identical to $p_k(2m,m)$ in Lemma 6.2.3. This is established as $\binom{2m}{m+b_i} = \binom{2m}{m-b_i}$. $\square$

In fact, $C(2m,k)$ denotes the number of arrays for gap 2^h, which can be successfully sorted using the proposed modified Shell sort with a window length of k. Clearly, the exact number of arrays with $W(\mathbf{a}) = k$, such that $a_i < a_{i+2}$ for all i can be obtained by computing $C(2m,k) - C(2m,k-1)$. With this result, I derive the running time complexity of the modified Shell sort in the next subsection.

6.2.2 Derivation of Running Time Complexity for a Specific SFP

In this subsection, I derive the running time complexity $O(n^{3/2}\sqrt{\alpha + \log\log n})$ of the proposed modified Shell sort with powers-of-two gap sequence, considering the optimal trade-off with the SFP $2^{-\alpha}$, in which α is the parameter that controls the window length of each gap. In the running time complexity, $\log\log n$ increases gradually as n increases. Therefore, the running time complexity is approximately proportional to

$n^{3/2}\sqrt{\alpha}$. However, the probability that the output is not successfully sorted decreases exponentially as α increases. It is noted that the SFP $2^{-\alpha}$ is not related to the number of the input data. One of the advantages of the modified Shell sort algorithm is irrespective of the number of the input data, and thus I can obtain a trade-off between the SFP and running time complexity by considering an appropriate α.

It is important to prove the following lemmas to determine the relation between the binomial coefficients and exponential function. It is a well-known fact from the central limit theorem in statistics that the closer n is to infinity, the closer a binomial distribution is to a normal distribution. Even though the binomial and normal distributions are similar, I should establish that some binomial coefficients are upper-bounded by the probability distribution function of the normal distribution. The following Lemma 6.2.5 is used in the proof of Lemma 6.2.6, and Lemma 6.2.6 is used to prove Theorem 6.2.7.

Lemma 6.2.5. *Let $f : [a, \infty) \to \mathbb{R}$ be a function of some real number a satisfying the following;*

i) $\lim_{x\to\infty} f(x) = M$ *for some real number M.*

ii) There exists a positive integer n, such that the n-th order derivative $f^{(n)}(x)$ exists on (a, ∞), and $(-1)^n f^{(n)}(x) > 0$ for all $x \in (a, \infty)$.

Then, $f(x) > M$ for all $x \in [a, \infty)$.

Proof. It is sufficient to show that $f^{(m)}(x) \to 0$ as $x \to \infty$ and $(-1)^m f^{(m)}(x)$ is a monotonically decreasing function for m, $1 \le m \le n-1$. If this is proved, then $f(x)$ is a monotonically decreasing function and is larger than the limit value M from the first condition in Lemma 6.2.5, as $f'(x)$ is negative for (a, ∞). Since it is true for $m = n$ that $(-1)^m f^{(m)}(x) > 0$, I will prove the following: if it is true for $2 \le k \le n$ that $(-1)^k f^{(k)}(x) > 0$, then I have $\lim_{x\to\infty} f^{(k-1)}(x) = 0$, and it is true that $(-1)^{k-1} f^{(k-1)}(x)$ is a monotonically decreasing function.

Let $g_k(x) = (-1)^k f^{(k)}(x)$. As $(-1)^{k-1} f^{(k)}(x) = g'_{k-1}(x) < 0$ on (a, ∞), $g_{k-1}(x)$ is a monotonically decreasing function. As a monotonically decreasing function always converges to a certain value, if it possesses some lower bound, I obtain $\lim_{x\to\infty} g_{k-1}(x) = T$ for some T, or $\lim_{x\to\infty} g_{k-1}(x) = -\infty$. I assume that $\lim_{x\to\infty} g_{k-1}(x) = T$ for some $T \neq 0$, or $\lim_{x\to\infty} g_{k-1}(x) = -\infty$. Then, I can deduce some $N \in (a, \infty), R > 0$, such that $|g_{k-1}(x)| > R$, i.e., $f^{(k-1)}(x) > R$ for all $x > N$, or $f^{(k-1)}(x) < -R$ for all $x > N$.

Consider the case of $f^{(k-1)}(x) > R$. If I integrate both terms from N to $x \in (N, \infty)$ iteratively as

$$\begin{aligned} f^{(k-2)}(x) - f^{(k-2)}(N) &= \int_N^x f^{(k-1)}(t)dt \\ &> \int_N^x R dx = R(x-N) \end{aligned}$$

$$\begin{aligned} f^{(k-3)}(x) - f^{(k-3)}(N) &= \int_N^x f^{(k-2)}(t)dt \\ &> \int_N^x \left(R(x-N) + f^{(k-2)}(N) \right) dx \\ &= \frac{R}{2}(x-N)^2 + f^{(k-2)}(N)(x-N), \end{aligned}$$

we obtain

$$f(x) > \frac{R}{(k-1)!}(x-N)^{k-1} + \sum_{i=0}^{k-2} \frac{f^{(i)}(N)}{i!}(x-N)^i,$$

whose right-hand side tends to infinity, as $x \to \infty$. In this case, $f(x)$ tends to infinity as well, which contradicts the first condition. If I consider the case of $f^{(m)}(x) < -R$, the inequality is changed to

$$f(x) < -\frac{R}{(k-1)!}(x-N)^{k-1} + \sum_{i=0}^{k-2} \frac{f^{(i)}(N)}{i!}(x-N)^i,$$

whose right-hand side tends to negative infinity, as $x \to \infty$. Then $f(x)$ tends to negative infinity as well, which also contradicts the first condition.

Thus, I obtain $\lim_{x\to\infty} g_{k-1}(x) = 0$. As $g_{k-1}(x)$ is a monotonically decreasing function, $g_{k-1}(x) > 0$ on (a, ∞), which completes the proof. $\square$

Lemma 6.2.6 directly uses Lemma 6.2.5. To prove the inequality in Lemma 6.2.6, I only prove that the condition of Lemma 6.2.5 holds for some function.

Lemma 6.2.6. *For any real number $\alpha \geq \sqrt{6}$ and any positive integer $n \geq \lceil \alpha^2 \rceil$, the following inequality holds*

$$\binom{2n}{n - \lceil \alpha\sqrt{n} \rceil} < e^{-\alpha^2}\binom{2n}{n}.$$

Proof. It can be derived that

$$\begin{aligned}\frac{\binom{2n}{n}}{\binom{2n}{n-\lceil\alpha\sqrt{n}\rceil}} &= \frac{(n+\lceil\alpha\sqrt{n}\rceil)(n+\lceil\alpha\sqrt{n}\rceil-1)\cdots(n+1)}{n(n-1)\cdots(n-\lceil\alpha\sqrt{n}\rceil+1)}\\ &= \prod_{k=0}^{\lceil\alpha\sqrt{n}\rceil-1}\left(1+\frac{\lceil\alpha\sqrt{n}\rceil}{n-k}\right).\end{aligned}$$

It should be proved that

$$\prod_{k=0}^{\lceil\alpha\sqrt{n}\rceil-1}\left(1+\frac{\lceil\alpha\sqrt{n}\rceil}{n-k}\right) > e^{\alpha^2}. \tag{7}$$

If I consider the logarithm on the left-hand side and change the form, I obtain

$$\sum_{k=0}^{\lceil\alpha\sqrt{n}\rceil-1} \ln\left(1+\frac{\lceil\alpha\sqrt{n}\rceil}{n-k}\right) \geq \sum_{k=0}^{\lceil\alpha\sqrt{n}\rceil-1} \ln\left(1+\frac{\alpha}{\sqrt{n}-\frac{k}{\sqrt{n}}}\right). \tag{8}$$

Let $f(x) = \log\left(1+\frac{\alpha}{x}\right)$. Then, the right-hand side of (8) can be defined as

$$\sqrt{n}\sum_{k=1}^{\lceil\alpha\sqrt{n}\rceil}\frac{1}{\sqrt{n}}f(\sqrt{n}+\frac{1}{\sqrt{n}}-\frac{k}{\sqrt{n}}),$$

which is a type of Riemann sum of $f(x)$. As $f(x)$ is a monotonically decreasing function, the Riemann sum demonstrates its lower bound as the integration of $f(x)$ from

$\sqrt{n}+\frac{1}{\sqrt{n}}-\frac{\lceil\alpha\sqrt{n}\rceil}{\sqrt{n}}$ to $\sqrt{n}+\frac{1}{\sqrt{n}}$. As $\sqrt{n}+\frac{1}{\sqrt{n}}-\frac{\lceil\alpha\sqrt{n}\rceil}{\sqrt{n}} \leq \sqrt{n}+\frac{1}{\sqrt{n}}-\alpha$, I obtain

$$\begin{aligned}\sum_{k=0}^{\lceil\alpha\sqrt{n}\rceil-1} &\ln\left(1+\frac{\alpha}{\sqrt{n}-\frac{k}{\sqrt{n}}}\right)\\ &\geq \sqrt{n}\int_{\sqrt{n}+\frac{1}{\sqrt{n}}-\frac{\lceil\alpha\sqrt{n}\rceil}{\sqrt{n}}}^{\sqrt{n}+\frac{1}{\sqrt{n}}} \ln\left(1+\frac{\alpha}{x}\right)dx\\ &\geq \sqrt{n}\int_{\sqrt{n}+\frac{1}{\sqrt{n}}-\alpha}^{\sqrt{n}+\frac{1}{\sqrt{n}}} \ln\left(1+\frac{\alpha}{x}\right)dx. \qquad (9)\end{aligned}$$

To integrate right-hand side of (9), let $g(x)=x\ln x$. I then obtain

$$\begin{aligned}&\sqrt{n}\int_{\sqrt{n}+\frac{1}{\sqrt{n}}-\alpha}^{\sqrt{n}+\frac{1}{\sqrt{n}}} \ln\left(1+\frac{\alpha}{x}\right)dx\\ &= g(n+\alpha\sqrt{n}+1)+g(n-\alpha\sqrt{n}+1)-2g(n+1).\end{aligned}$$

Let $h(x)=g(x^2+\alpha x+1)+g(x^2-\alpha x+1)-2g(x^2+1)$ in $[\alpha,\infty)$. If I prove $\lim_{x\to\infty} h(x)=\alpha^2$, and $h^{(3)}(x)<0$ in (α,∞), I obtain $h(x)>\alpha^2$ in $[\alpha,\infty)$ using Lemma 6.2.5. As $\sqrt{n}\geq\alpha$, I obtain $h(\sqrt{n})>\alpha^2$, which proves (7). I must establish $\lim_{x\to\infty} h(x)=\alpha^2$, and $h^{(3)}(x)<0$ in (α,∞). To prove $\lim_{x\to\infty} h(x)=\alpha^2$, I consider $e^{h(x)}$. Using $g(x)=x\ln x$ and $h(x)$, I obtain

$$e^{h(x)}=\left(1-\frac{\alpha^2x^2}{(x^2+1)^2}\right)^{x^2-\alpha x+1}\left(1+\frac{\alpha x}{x^2+1}\right)^{2\alpha x}.$$

From $\lim_{x\to\infty}\left(1+\frac{p}{x}\right)^x=e^p$, I obtain $\lim_{x\to\infty} e^{h(x)}=e^{\alpha^2}$, and thus, $\lim_{x\to\infty} h(x)=\alpha^2$. Moreover, $h^{(3)}(x)$ can be computed as

$$h^{(3)}(x)=-\frac{4\alpha^2x(x^2-1)\{\alpha^2(x^4+4x^2+1)-6(x^2+1)^2\}}{(x^2+1)^2(x^2-\alpha x+1)^2(x^2+\alpha x+1)^2}.$$

As $\alpha\geq\sqrt{6}$, I obtain $h^{(3)}(x)<0$ in (α,∞). Thus, I complete the proof. $\square$

The following theorem is presented, which is the main theorem of this subsection. The situation in Theorem 6.2.4 occurs in Theorem 6.2.7, so that I can directly use Theorem 6.2.4. Then, I use Lemma 6.2.6 to obtain a simple upper bound of the complicated formula.

Theorem 6.2.7. *The running time complexity of the proposed modified Shell sort algorithm is obtained as $O(n^{3/2}\sqrt{\alpha + \log\log n})$. For $\alpha \geq \sqrt{6}\log e - 1$, its SFP is upper-bounded by $2^{-\alpha}$.*

Proof. As the swapping operation in the modified Shell sort algorithm can be performed within a certain constant time, the running time complexity of the modified Shell sort in Algorithm 37 is determined from the number of swapping operations. Let $S(n)$ be the number of the swapping operations with an input length n. Then, $S(n)$ can be upper-bounded as

$$S(n) \leq n \sum_{\ell=0}^{\lfloor \log n \rfloor} k_\ell$$

where the window length k_ℓ of each gap is defined as

$$\left\lceil \sqrt{\left\lceil \frac{n}{2^{\ell+1}} \right\rceil \cdot (\alpha + 1 + \log\log n + \ell) \cdot \frac{1}{\log e}} \right\rceil .$$

Thus, $S(n)$ can be expressed as

$$S(n) = O\left(n^{\frac{3}{2}} \sum_{\ell=0}^{\lfloor \log n \rfloor} \sqrt{\frac{\alpha + \log\log n + \ell + 1}{2^{\ell+1}}} \right).$$

Using $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$, I obtain that $T(n)$ is

$$O\left(n^{\frac{3}{2}} \left(\sqrt{\alpha + \log\log n} \sum_{\ell=1}^{\lfloor \log n \rfloor + 1} \frac{1}{2^{\frac{\ell}{2}}} + \sum_{\ell=1}^{\lfloor \log n \rfloor + 1} \frac{\sqrt{\ell}}{2^{\frac{\ell}{2}}} \right) \right).$$

Thus, I obtain $S(n) = O(n^{3/2}\sqrt{\alpha + \log\log n})$, because $\sum_{\ell=1}^{\infty} \frac{1}{2^{\frac{\ell}{2}}}$ and $\sum_{\ell=1}^{\infty} \frac{\sqrt{\ell}}{2^{\frac{\ell}{2}}}$ are both finite.

At this point, I consider the SFP. Let B denote the event that the output of the sorting algorithm is not successfully sorted and let B_ℓ denote the event that at least one subarray for the gap 2^ℓ is not successfully sorted. As $\mathsf{B} \subseteq \bigcup_{\ell=0}^{\lfloor \log n \rfloor} \mathsf{B}_\ell =$

$\bigcup_{\ell=0}^{\lfloor \log n \rfloor} \left(\mathsf{B}_\ell \cap \bigcap_{u=\ell+1}^{\lfloor \log n \rfloor} \mathsf{B}_u^c \right)$, I obtain

$$\begin{aligned} \Pr[\mathsf{B}] &\leq \sum_{\ell=0}^{\lfloor \log n \rfloor} \Pr\left[\mathsf{B}_\ell \cap \bigcap_{u=\ell+1}^{\lfloor \log n \rfloor} \mathsf{B}_u^c \right] \\ &\leq \sum_{\ell=0}^{\lfloor \log n \rfloor} \Pr\left[\mathsf{B}_\ell \middle| \bigcap_{u=\ell+1}^{\lfloor \log n \rfloor} \mathsf{B}_u^c \right], \end{aligned}$$

where $\bigcap_{u=\ell+1}^{\lfloor \log n \rfloor} \mathsf{B}_u^c$ implies the event that the sorting is successful for the gaps $2^{\ell+1}, \cdots, 2^{\lfloor \log n \rfloor}$. All of the subarrays satisfy the condition $a_i < a_{i+2}$ in Theorem 6.2.4, before I perform the insertion sort for the gap 2^ℓ. Clearly, there are 2^ℓ subarrays when the gap is 2^ℓ, and the length of subarray is less than or equal to $2\lceil \frac{n}{2^{\ell+1}} \rceil$. Let $m_\ell = \lceil \frac{n}{2^{\ell+1}} \rceil$, and $\beta_\ell = \sqrt{(\alpha + 1 + \log\log n + \ell) \cdot \frac{1}{\log e}}$. As $\beta_\ell \geq \sqrt{6}$, the probability that one subarray of length $2m_\ell$ is not successfully sorted can be upper-bounded as

$$1 - \frac{C(2m_\ell, \beta_\ell \sqrt{m_\ell})}{\binom{2m_\ell}{m_\ell}} \leq 2 \frac{\binom{2m_\ell}{m_\ell - \beta_\ell \sqrt{m_\ell}}}{\binom{2m_\ell}{m_\ell}} \leq 2e^{-\beta_\ell^2}, \tag{10}$$

where the second inequality is obtained from Lemma 6.2.6 if $m_\ell \geq \lceil \beta_\ell^2 \rceil$. If $m_\ell < \lceil \beta_\ell^2 \rceil$, the left term of (10) is 0, and thus (10) trivially holds. I then obtain

$$\begin{aligned} \sum_{\ell=0}^{\lfloor \log n \rfloor} \Pr\left[\mathsf{B}_\ell \middle| \bigcap_{u=\ell+1}^{\lfloor \log n \rfloor} \mathsf{B}_u^c \right] &\leq \sum_{\ell=0}^{\lfloor \log n \rfloor} 2^\ell \cdot 2e^{-\beta_\ell^2} \\ &= \frac{2^{-\alpha} \lfloor \log n \rfloor}{\log n} \leq 2^{-\alpha}, \end{aligned}$$

and thus, the theorem is proved. □

Remark. Theorem 6.2.7 states that the asymptotic running time complexity of Algorithm 37 is lower than the trivially modified Shell sort in Algorithm 36, which is $O(n^2)$. The reduction of the running time in a concrete sense is rather clear, in that the number of iterative steps in the last **for** statement in Algorithm 37 is lower than that in Algorithm 36. On the other hand, the asymptotic running time of the modified Shell sort is lower than that of the insertion sort for the FHE setting, but the concrete comparison for them in a practical situation is not clear in this theoretical analysis. In Section 6.4, I numerically compare their running time using TFHE homomorphic encryption scheme.

6.3 Near-Optimal Window Length by Convex Optimization

It is necessary to find the shortest window length for the SFP so that the least running time complexity of the modified Shell sort is obtained. Generally, it is not easy to derive the optimal window length in closed form. In this section, I obtain the near-optimal window length using convex optimization [10]. Let $\beta_\ell\sqrt{\lceil n/2^{\ell+1}\rceil}$ be the window length for the gap 2^ℓ, and $\Pr\left[\mathsf{B}_\ell \middle| \bigcap_{u=\ell+1}^{\lfloor\log n\rfloor}\mathsf{B}_u^c\right]$ be the SFP for the gap 2^ℓ, when sorting is successful for the gaps $2^{\ell+1},\cdots,2^{\lfloor\log n\rfloor}$. From Theorem 6.2.4 and Lemma 6.2.6, I obtain

$$\Pr\left[\mathsf{B}_\ell \middle| \bigcap_{u=\ell+1}^{\lfloor\log n\rfloor}\mathsf{B}_u^c\right] \le 2^\ell e^{-\beta_\ell^2}.$$

The objective function that needs to be minimized is the total number of swap operations, which determines the running time. As the exact running time formula is rather complicated, I consider a tight upper bound of the running time, $n\sum_{\ell=0}^{\lfloor\log n\rfloor}\beta_\ell\sqrt{\lceil n/2^{\ell+1}\rceil}$, which is used in the proof of Theorem 6.2.7. Let $p_\ell = 2^\ell e^{-\beta_\ell^2}$. Then, I have $\beta_\ell = \sqrt{(\ell+\log(1/p_\ell))/\log e}$. As it is sufficient to minimize $\sum_{\ell=0}^{\lfloor\log n\rfloor}\sqrt{\lceil n/2^{\ell+1}\rceil(\ell+\log(1/p_\ell))}$, the problem of the near-optimal window length can be formulated as follows;

$$\begin{aligned}\textbf{minimize}\quad & \sum_{\ell=0}^{\lfloor\log n\rfloor}\sqrt{\lceil n/2^{\ell+1}\rceil(\ell+\log(1/p_\ell))}\\ \textbf{s.t.}\quad & \sum_{\ell=1}^{k-1}p_\ell \le p_{\mathrm{err}}.\end{aligned}$$

This formulation implies that the total running time with SFP upper-bounded by p_{err} needs to be minimized. I can validate that $\sqrt{c+\log\frac{1}{x}}$ is a convex function on small positive values, where c is a constant. As the weighted sum of convex functions is also a convex function, the objective function is a convex function, and the constraint is also convex. Thus, this can be termed as a convex optimization problem. As every convex optimization problem can be solved using numerical analysis, it is easy to obtain the near-optimal window length. Then, I can deduce p_ℓ, and the near-optimal window length is determined to be $\lceil \sqrt{\lceil n/2^{\ell+1} \rceil (\ell + \log(1/p_\ell))/\log e} \rceil$ for each gap 2^ℓ. It is noted that the above formulation is not sufficiently tight, because it still uses the union bound. Constructing a tighter formulation, which can be solved easily, can be a focus for future research.

6.4 Simulation Results

The performance of the proposed modified Shell sort is numerically verified using a personal computer with an AMD Ryzen 9 5950X CPU running at 2.04GHz, and 128GB RAM. First, I validate the running time and SFP when the array length varies. Then, the running time and SFP are numerically obtained when the parameter α is varied. Finally, the performance of the modified Shell sort is compared with the cases corresponding to the near-optimal window length, which is obtained using convex optimization, and Ciura's optimal gap sequence, which has been validated numerically as an optimal gap sequence in the non-FHE settings. I firstly simulate these sorting algorithms without homomorphic encryption schemes, i.e., in the plaintext region. Since the use of homomorphic encryption schemes can affect only the running time, the result of SFP values in this simulation has the same meaning in the case of using homomorphic encryption.

Fig. 6.3 shows the relation between the running time and SFP against various array lengths for $\alpha = 3$. It is observed that the array length increases from 50 to 1000. The input arrays are randomly generated, and 10^5 input arrays are generated for each array length. It is observed from Fig. 6.3 that the running time increases in proportion to $n^{3/2}$, and the SFP is independent of the array length. This numerical result coincides

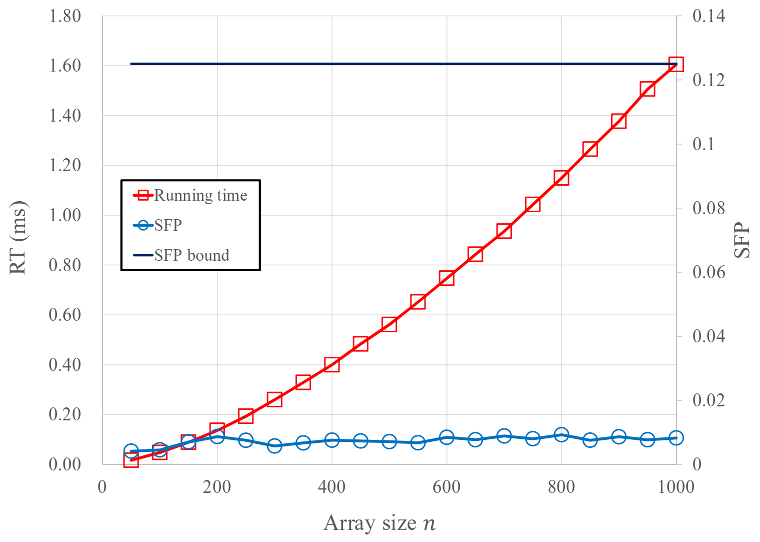


Figure 6.3: Running time and SFP of the modified Shell sort for varied array lengths.

well with the proposed analysis of the modified Shell sort. Note that the value $c = \alpha + 1 + \log\log n$ increases slightly as the length of the array increases.

Fig. 6.4 shows the relation between the running time and SFP for various α values, in which g2p denotes the power of the 2-gap sequence, gop denotes Ciura's optimal gap sequence [27], and a-win and o-win denote the analytically derived window length and near-optimal window length derived by convex optimization, respectively. The input array length is fixed at 1000. Similar to the previous simulation, 10^5 input arrays are randomly generated for each α value. Algorithm 37 and the case corresponding to the Ciura's optimal gap sequence or near-optimal window length are simulated, with the near-optimal window length derived using the convex optimization discussed in Section 6.3.

From Fig. 6.4, it is observed that the running time of Algorithm 37 increases as α increases and the growth rate decreases. This observation coincides with the proposed analysis, i.e., the running time is approximately proportional to $\sqrt{\alpha}$. The logarithms of the SFP values of Algorithm 37 are parallel to that of the SFP bounds. This implies that the SFP is proportional to $2^{-\alpha}$ with some small proportional constant.

When the gap sequence is replaced with Ciura's gap sequence, the running time is reduced by approximately 0.5 ms. Sorting failure is not detected in the case of the simulation that uses Ciura's gap sequence. This implies that the order of the SFP of Ciura's optimal gap sequence is less than or equal to 10^{-5}. Although the window lengths of each gap in this dissertation are analytically derived for the power of the 2-gap sequence, a better result is obtained when Ciura's optimal gap sequence is used.

The value $c = \alpha + 1 + \log\log n$ is numerically found when the SFP value reaches 10^{-5} for some length of an array, and Table 6.1 shows the values. While the value c increases slightly as the length of the array increases when the gap sequence is powers-of-two and the SFP value is fixed, the value c decreases sharply as the length of the array increases. It suggests that the trade-off in the case of Ciura's gap sequence is asymptotically better than the case of the powers-of-two gap sequence. The exact asymptotical analysis of Ciura's gap sequence is an open problem.

The near-optimal window length is derived using the convex optimization problem described in Section 6.3. The running time in this case is marginally reduced compared with the case using the analytically obtained window length. However, their values become closer as α increases. The SFP of the case using the near-optimal window length for the power of the 2-gap sequence is closer to the SFP bound than that of the case using the analytically obtained window length. Thus, the running time can be reduced, while the SFP remains less than the SFP bound.

In this subsection, I measure the running time of several sorting algorithms on encrypted data, including the modified Shell sort algorithm. I implement each sorting algorithm with the TFHE library [24]. The security parameter in the TFHE scheme is set to be 128, and the number of bits for each data is set to be 10. Table 6.2 shows the main parameters used in the simulation satisfying 128-bit security. For the modified Shell sort, I set the value of c to make the SFP 10^{-5}, and the Ciura's gap sequence is used rather than the powers-of-two gap sequence. The unit of each running time result is in seconds.

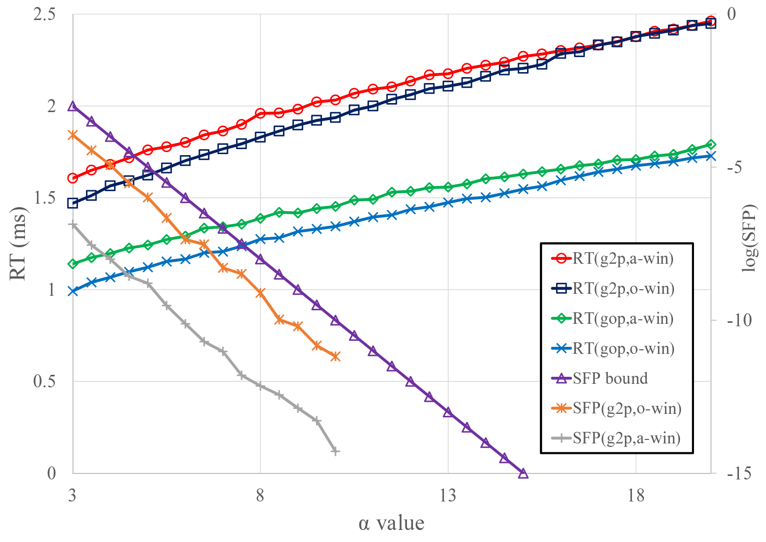


Figure 6.4: Running time and SFP of the modified Shell sort for varied α values and comparison of these values with those obtained from the cases of Ciura's optimal gap sequence and near-optimal window length derived by convex optimization.

Table 6.1: The value $c = \alpha + 1 + \log \log n$ for array of various lengths when the Ciura's gap sequence is used and the SFP is 10^{-5}

Array length	Value c
100	4.93
200	2.73
300	2.04
400	1.41
500	1.26

Table 6.2: TFHE parameters with 128-bit security

	Ciphertext dimension	Noise standard deviation
Ciphertext and Key-switching key (LWE)	630	2^{-15}
Bootstrapping key (Ring-LWE)	1024	2^{-25}

The sorting algorithm to be compared with the modified Shell sort is chosen as follows. Since the modified Shell sort can be the generalized algorithm for the insertion sort, I choose to compare the insertion sort. The randomized Shell sort [39] is the most related sorting algorithm to the proposed modified Shell sort, and thus I also choose it to be compared. These two sorting algorithms are in-place algorithms. For the recursive sorting algorithm, the odd-even merge sort and the bitonic sort are chosen, which are the standard oblivious recursive sorting algorithms. These two recursive algorithms are also used in [31] to compare the sorting algorithm for homomorphic encryption.

As for the odd-even merge sort and the bitonic sort, the length of the input array is originally assumed to be a power of two. However, I cannot generally choose the array length, and thus I perform the simulation with a more general type of numbers rather than the power-of-two array length. Since the length of the input array in the simulation is not a power-of-two integer, I add dummy data in the end of the array to make the input array power-of-two, and these dummy data is assumed to be larger than the data in the input array. Since these dummy data will not be moved in the sorting, I ignore the comparison if dummy data is homomorphically compared to other data, in order to erase the effect of the addition of dummy data. Thus, I can fairly compare the running time of each sorting algorithm in the case of more general array lengths other than the power-of-two length.

Table 6.3 shows the running time of several sorting algorithms required to sort an array of encrypted data of length 500. The running time of the modified Shell sort with Ciura's gap sequence is far lower than the insertion sort, which is the basic in-place sorting algorithm. The use of the modified Shell sort is proved to be efficient not only in the asymptotic sense but also practical sense. Also, although the randomized Shell sort [39] is asymptotically better than the proposed algorithm, the performance of the proposed algorithm is better than that of the randomized Shell sort for an array of length 500.

When I compare the efficient and recursive sorting algorithms, bitonic sort and odd-even merge sort, the running time of the modified Shell sort is yet larger, but it is closer than the original insertion sort. This running time performance will depend on the situation, especially in the IoT device. Since these recursive sorting algorithms make many function calls recursively and the memory of the input array is quite big, the transmission time can be a serious problem when the memory bus bandwidth is not large enough. In this situation, the modified Shell sort will be useful in that it uses no function calls or almost no additional memories. Even though the running times of bitonic sort and odd-even merge sort are smaller than that of the modified Shell sort, the numbers of memory and function calls of the bitonic sort and odd-even merge sort increases to 5121 and 9729, respectively.

Table 6.4 shows the performance of the modified Shell sort, the insertion sort, and the bitonic sort for an array of various lengths less than 500. While the running time of the insertion sort increases fast as the array length increases, the running time of the modified Shell sort with Ciura's gap sequence increases not very fast as the array length increases. This rate is somewhat similar to the rate of the Bitonic sort, whose running time complexity is better than the proposed modified Shell sort.

Table 6.3: Comparison of the running time of several sorting algorithms for array of length 500

Sorting algorithm	Running time (sec)	Non-recursive/ Recursive	Number of function calls
Insertion sort	97889	Non-recursive	1
Rand Shell sort [39]	58583	Non-recursive	1
Modified Shell sort	**28576**	**Non-recursive**	**1**
Odd-even merge sort	13085	Recursive	5121
Bitonic sort	8753	Recursive	9729

Table 6.4: Comparison of the running time in seconds of several sorting algorithms for array of various lengths

Array length	Modified Shell sort (in-place)	Insertion sort (in-place)	Bitonic sort (recursive)
100	**2521**	3899	1038
200	**7281**	15636	2745
300	**13352**	34988	5066
400	**20793**	62739	6812

Chapter 7

CONCLUSION AND FUTURE WORKS

High Precision Bootstrapping for RNS-CKKS FHE Scheme I proposed the algorithm for obtaining the optimal minimax approximate polynomial for any continuous function on the union of the finite set, including the scaled cosine function on separate approximation regions. Then I analyzed the message precision of the bootstrapping with the improved multi-interval Remez algorithm in RNS-CKKS, and its maximum message precision is measured in the `SEAL` library. I proposed the composite function method with inverse sine function to improve the message precision of the bootstrapping significantly, and thus the improved message precision bootstrapping has the precision higher than the precision of the 32-bit fixed-point number system, even when lots of slots are used. Thus, the large-depth operations in advanced applications, such as training a convolutional neural network for encrypted data, are needed to be implemented by the RNS-CKKS scheme with the improved message precision bootstrapping.

Privacy-Preserving Deep Neural Network I constructed an efficient privacy-preserving VDSCNN model on the RNS-CKKS scheme. First, I minimized the bootstrapping runtime via multiplexed packing and proposed multiplexed parallel convolution algorithm that works for multiplexed input tensor, which also supports strided convolutions. Further, I addressed the catastrophic divergence problem of VDSCNNs on the RNS-CKKS scheme and resolved it by the proposed imaginary-removing bootstrapping. By carefully integrating computations, I effectively reduced the level consumption. The simulation results reported $4.67\times$ lower latency and $134\times$ lower amortized runtime for ResNet-20 inference compared to the implementation without proposed techniques while achieving the 128-bit security. I also successfully implemented ResNet-32/44/56/110 on the RNS-CKKS scheme for the first time.

Hierarchical Galois Key Generation For the privacy-preserving AIaaS with FHE in the client-server model, I proposed a hierarchical Galois key generation method for the first time to significantly reduce the computational and communication costs of the client and to make the Galois key management with the reduced memory in the server more efficient. It allows the server to generate the Galois keys for the required cyclic shifts using the Galois keys in the higher key level without a secret key or any help from the clients, and it helps the server to use its memory for storing Galois keys efficiently. With this method, I showed a simple protocol between a client and a server using the simplest two-level hierarchical Galois key system and a more general protocol capable of efficient Galois key management reflecting the activity of the clients using a multi-level hierarchical Galois key system.

I constructed this hierarchical Galois key generation system for the BFV and CKKS schemes using a key-switching operation applied to the public key. Also, I proposed a more efficient Galois key generation method by using Edmonds' algorithm and Prim's algorithm when a bundle of Galois keys has to be generated at once. Finally, I suggested a concrete example of Galois key management protocols by putting them together.

This proposed system will make the FHE system more flexible in various applications, especially when efficient memory use is highly required in the cloud computing system. It can be an important future work that designs a systematic method to perform complex services with limited memory by using the proposed method more efficiently. Designing a fast algorithm for generating a sequence of Galois keys closer to the optimal solution in the proposed situation is also an important future work.

Modified Shell Sort for FHE In this dissertation, I proposed a modified Shell sort with an additional parameter α in the FHE setting, and for a gap sequence of powers of two, I derived the running time complexity $O(n^{3/2}\sqrt{\alpha + \log\log n})$, considering a trade-off with the SFP $2^{-\alpha}$. I also established that the running time complexity of the proposed algorithm is almost the same as the average-case running time complexity of the original Shell sort, while the SFP is maintained to be minimal. I then obtained the near-optimal window length of each gap by numerically solving a convex optimization problem. I believe that this study plays a significant role in the foundation of the analysis of the Shell sort in the FHE settings. Using the TFHE encryption scheme, the running time of the proposed modified Shell sort with Ciura's gap sequence was compared with that of the conventional sorting algorithms, and it has the best running time performance among other in-place sorting algorithms in the FHE setting.

Bibliography

[1] Lattigo v3. Online: `https://github.com/tuneinsight/lattigo` (Apr 2022), ePFL-LDS, Tune Insight SA

[2] Badawi, A.A., Chao, J., Lin, J., Mun, C.F., Jie, S.J., Tan, B.H.M., Nan, X., Khin, A.M.M., Chandrasekhar, V.: Towards the Alexnet moment for homomorphic encryption: HCNN, the first homomorphic CNN on encrypted data with GPUs. IEEE Transactions on Emerging Topics in Computing (2020)

[3] Bajard, J.C., Eynard, J., Hasan, M.A., Zucca, V.: A full rns variant of fv like somewhat homomorphic encryption schemes. In: International Conference on Selected Areas in Cryptography. pp. 423–442. Springer (2016)

[4] Blatt, M., Gusev, A., Polyakov, Y., Rohloff, K., Vaikuntanathan, V.: Optimized homomorphic encryption solution for secure genome-wide association studies. BMC Medical Genomics **13**(7), 1–13 (2020)

[5] Boemer, F., Cammarota, R., Demmler, D., Schneider, T., Yalame, H.: MP2ML: A mixed-protocol machine learning framework for private inference. In: Proceedings of the 15th International Conference on Availability, Reliability and Security. pp. 1–10 (2020)

[6] Boemer, F., Costache, A., Cammarota, R., Wierzynski, C.: nGraph-HE2: A high-throughput framework for neural network inference on encrypted data. In: *Pro-*

ceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography. pp. 45–56 (2019)

[7] Boemer, F., Lao, Y., Cammarota, R., Wierzynski, C.: nGraph-HE: a graph compiler for deep learning on homomorphically encrypted data. In: *Proceedings of the 16th ACM International Conference on Computing Frontiers*. pp. 3–13 (2019)

[8] Bossuat, J.P., Mouchet, C., Troncoso-Pastoriza, J., Hubaux, J.P.: Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In: EUROCRYPT 2021. pp. 587–617. Springer (2021)

[9] Bossuat, J.P., Troncoso-Pastoriza, J.R., Hubaux, J.P.: Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. Cryptology ePrint Archive (2022)

[10] Boyd, S., Vandenberghe, L.: *Convex optimization*. Cambridge university press (2004)

[11] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory* **6**(3), 13 (2014)

[12] Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: *CRYPTO 2012*. pp. 868–886. Springer (2012)

[13] Chatterjee, A., Kaushal, M., Sengupta, I.: Accelerating sorting of fully homomorphic encrypted data. In: *Int. Conf. Cryptology in India*. pp. 262–273. Springer (2013)

[14] Chatterjee, A., Sengupta, I.: Windowing technique for lazy sorting of encrypted data. In: *IEEE Conf. Communications and Network security*. pp. 633–637. IEEE (2015)

[15] Chatterjee, A., SenGupta, I.: Sorting of fully homomorphic encrypted cloud data: Can partitioning be effective? *IEEE Trans. Services Computing* (2017)

[16] Chen, H., Chillotti, I., Song, Y.: Improved bootstrapping for approximate homomorphic encryption. In: *Advances in Cryptology-EUROCRYPT 2019*. pp. 34–54. Springer (2019)

[17] Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., Li, L., Chen, T., Xu, Z., Sun, N., et al.: Dadiannao: A machine-learning supercomputer. In: *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. pp. 609–622. IEEE (2014)

[18] Cheney, E.: *Introduction to approximation theory*. McGraw-Hill (1966)

[19] Cheon, J., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: *Advances in Cryptology-ASIACRYPT 2017*. pp. 409–437. Springer (2017)

[20] Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: EUROCRYPT 2018. pp. 360–384 (2018)

[21] Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: A full RNS variant of approximate homomorphic encryption. In: *Proceedings of International Conference on Selected Areas in Cryptography*. pp. 347–368 (2018)

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

[23] Cheon, J.H., Kim, D., Kim, D., Lee, H.H., Lee, K.: Numerical method for comparison on homomorphically encrypted numbers. In: ASIACRYPT 2019. pp. 415–445 (2019)

[24] chillotti, i., gama, n., georgieva, m., izabachène, m.: TFHE: fast fully homomorphic encryption library (august 2016), https://tfhe.github.io/tfhe/

[25] Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology* **33**(1), 34–91 (2020)

[26] Chou, E., Beal, J., Levy, D., Yeung, S., Haque, A., Fei-Fei, L.: Faster cryptonets: Leveraging sparsity for real-world encrypted inference. arXiv preprint, abs/1811.09953 (2018), http://arxiv.org/abs/1811.09953

[27] Ciura, M.: Best increments for the average case of shellsort. In: *Int. Symp. Fundamentals of Computation Theory*. pp. 106–117. Springer (2001)

[28] Dathathri, R., Saarikivi, O., Chen, H., Laine, K., Lauter, K., Maleki, S., Musuvathi, M., Mytkowicz, T.: Chet: an optimizing compiler for fully-homomorphic neural-network inferencing. In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. pp. 142–156 (2019)

[29] Edmonds, J.: Optimum branchings. Journal of Research of the national Bureau of Standards B **71**(4), 233–240 (1967)

[30] van Elsloo, T., Patrini, G., Ivey-Law, H.: SEALion: A framework for neural network inference on encrypted data. arXiv preprint, abs/1904.12840 (2019), http://arxiv.org/abs/1904.12840

[31] Emmadi, N., Gauravaram, P., Narumanchi, H., Syed, H.: Updates on sorting of fully homomorphic encrypted data. In: *Int. Conf. Cloud Computing Research and Innovation (ICCCRI)*. pp. 19–24. IEEE (2015)

[32] Espelid, T.O.: Analysis of a shellsort algorithm. *BIT Numerical Mathematics* **13**(4), 394–400 (1973)

[33] Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2020), https://eprint.iacr.org/2012/144

[34] Filip, S.: A robust and scalable implementation of the Parks-McClellan algorithm for designing FIR filters. *ACM Transactions on Mathematical Software* **43**(1), 1–24 (2016)

[35] Gentry, C.: Fully homomorphic encryption using ideal lattices. In: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*. pp. 169–178 (2009)

[36] Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In: *Advances in Cryptology-CRYPTO 2013*. pp. 75–92. Springer (2013)

[37] Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: Proceedings of International Conference on Machine Learning (ICML). pp. 201–210 (2016)

[38] Goldschmidt, R.E.: Applications of division by convergence. Ph.D. thesis, Massachusetts Institute of Technology (1964)

[39] Goodrich, M.T.: Randomized shellsort: A simple data-oblivious sorting algorithm. *J. ACM* **58**(6), 27 (2011)

[40] Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P.: Deep learning with limited numerical precision. In: *International Conference on Machine Learning*. pp. 1737–1746 (2015)

[41] Gysel, P., Motamedi, M., Ghiasi, S.: Hardware-oriented approximation of convolutional neural networks. In: *International Conference on Learning Representation* (2016)

[42] Halevi, S., Shoup, V.: Algorithms in HElib. In: Annual Cryptology Conference. pp. 554–571. Springer (2014)

[43] Halevi, S., Shoup, V.: Faster homomorphic linear transformations in helib. In: CRYPTO 2018. pp. 93–120. Springer (2018)

[44] Han, K., Ki, D.: Better bootstrapping for approximate homomorphic encryption. In: *Cryptographers' Track at the RSA Conference*. pp. 364–390. Springer (2020)

[45] Hesamifard, E., Takabi, H., Ghasemi, M.: Cryptodl: Deep neural networks over encrypted data. arXiv preprint arXiv:1711.05189 (2017)

[46] Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning. pp. 448–456. PMLR (2015)

[47] Jiang, X., Kim, M., Lauter, K., Song, Y.: Secure outsourced matrix computation and application to neural networks. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. pp. 1209–1222 (2018)

[48] Jung, W., Kim, S., Ahn, J.H., Cheon, J.H., Lee, Y.: Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with GPUs. IACR Transactions on Cryptographic Hardware and Embedded Systems **2021**(4), 114–148 (Aug 2021)

[49] Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: GAZELLE: A low latency framework for secure neural network inference. In: *Proceedings of the 27th USENIX Security Symposium*. pp. 1651–1669 (2018)

[50] Kim, A., Papadimitriou, A., Polyakov, Y.: Approximate homomorphic encryption with reduced approximation error. *Cryptol. ePrint Arch., Tech. Rep. 2020/1118* **2020** (2020)

[51] Kim, M., Jiang, X., Lauter, K., Ismayilzada, E., Shams, S.: HEAR: Human action recognition via neural networks on homomorphically encrypted data. arXiv preprint arXiv:2104.09164 (2021)

[52] Kim, S., Kim, J., Kim, M.J., Jung, W., Rhu, M., Kim, J., Ahn, J.H.: BTS: An accelerator for bootstrappable fully homomorphic encryption. arXiv preprint arXiv:2112.15479 (2021)

[53] Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. CiteSeerX Technical Report, University of Toronto (2009)

[54] Lee, E., Lee, J.W., No, J.S., Kim, Y.S.: Minimax approximation of sign function by composite polynomial for homomorphic comparison. *IEEE Transactions on Dependable and Secure Computing*, accepted for publication (2021)

[55] Lee, J.W., Lee, E., Lee, Y., Kim, Y.S., No, J.S.: Optimal minimax polynomial approximation of modular reduction for bootstrapping of approximate homomorphic encryption. Cryptology ePrint Archive, Report 2020/552, Second Version (2020), https://eprint.iacr.org/2020/552/20200803:084202

[56] Lee, J.W., Lee, E., Lee, Y., Kim, Y.S., No, J.S.: High-precision bootstrapping of RNS-CKKS homomorphic encryption using optimal minimax polynomial approximation and inverse sine function. In: *EUROCRYPT 2021* (2021)

[57] Lee, J., Lee, E., Lee, J.W., Kim, Y., Kim, Y.S., No, J.S.: Precise approximation of convolutional neural networks for homomorphically encrypted data. arXiv preprint, abs/2105.10879 (2021), http://arxiv.org/abs/2105.10879

[58] Lee, Y., Lee, J.W., Kim, Y.S., Kang, H., No, J.S.: High-precision and low-complexity approximate homomorphic encryption by error variance minimization. Cryptology ePrint Archive (2020), accepted to *EUROCRYPT* 2022

[59] Lindell, Y.: How to simulate it–a tutorial on the simulation proof technique. Tutorials on the Foundations of Cryptography pp. 277–346 (2017)

[60] Lou, Q., Jiang, L.: SHE: A fast and accurate deep neural network for encrypted data. Advances in Neural Information Processing Systems **32**, 10035–10043 (2019)

[61] McClellan, J., Parks, T.: A personal history of the Parks-McClellan algorithm. *IEEE Signal Processing Magazine* **22**(2), 82–86 (2005)

[62] Powell, M.: *Approximation theory and methods*. Cambridge University Press (1981)

[63] Prim, R.C.: Shortest connection networks and some generalizations. The Bell System Technical Journal **36**(6), 1389–1401 (1957)

[64] Reagen, B., Choi, W.S., Ko, Y., Lee, V.T., Lee, H.H.S., Wei, G.Y., Brooks, D.: Cheetah: Optimizing and accelerating homomorphic encryption for private inference. In: 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA). pp. 26–39. IEEE (2021)

[65] Remez, E.: Sur la détermination des polynômes d'approximation de degré donnée. *Communications of the Kharkov Mathematical Society* **10**(196), 41–63 (1934)

[66] Rudin, W.: *Principles of mathematical analysis*, vol. 3. McGraw-Hill New York (1964)

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

[68] Microsoft SEAL (release 3.6). `https://github.com/Microsoft/SEAL` (Nov 2020), microsoft Research, Redmond, WA.

[69] Sedgewick, R.: Analysis of Shellsort and related algorithms. In: *European Symp. Algorithms*. pp. 1–11. Springer (1996)

[70] Shell, D.L.: A high-speed sorting procedure. *Comm. ACM* **2**(7), 30–32 (1959)

[71] Van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: *Advances in Cryptology-EUROCRYPT 2010*. pp. 24–43. Springer (2010)

초 록

완전 동형 암호는 서버가 클라이언트의 암호화된 데이터를 복호화하지 않고 임의의 연산을 할 수 있도록 하는 암호화 알고리즘이다. 이를 통해 클라이언트는 데이터에 대한 정보를 유출하지 않고 중요한 데이터를 신뢰하기 어려운 서버에 가공을 위탁할 수 있다. 2009년에 Gentry가 완전동형암호의 첫 번째 설계를 성공한 이후 여러 종류의 완전동형암호 알고리즘이 제안되고 다양한 측면에서 개선되어 왔지만, 현재의 완전동형암호 알고리즘은 실생활에 적용하기에 효율적이지 않다. 본 연구에서는 완전동형암호 알고리즘은 실제 적용에 적합하게 만들기 위해 완전동형암호 알고리즘에 대한 네 가지 주요 핵심 주제인 근사 동형 암호의 부트스트래핑, 갈루아 키 관리, 정보 보호 심층 신경망, FHE 상에서의 대한 정렬 개선을 다룬다.

첫째, 본 연구에서는 RNS-CKKS 알고리즘의 부트스트래핑 연산의 메시지 정밀도를 향상시킨다. 동형 나머지 연산 과정은 부트스트래핑의 정밀도를 결정하는 가장 중요한 단계 중 하나이기 때문에, 우리는 동형 나머지 연산 과정에 초점을 맞춘다. 우리는 먼저 함수의 최적 근사 다항식과 스케일링된 사인/코사인 함수를 얻는 *개선된 다중 간격 Remez 알고리즘*을 제안한다. 다음으로, 우리는 부트스트래핑에 사용되는 스케일링 상수와 기본 스케일링 상수의 차이를 줄이기 위해 역사인 함수를 사용하는 *함수 합성 방법*을 제안한다. 이러한 방법을 사용하여 각 매개 변수 설정에 대해 RNS-CKKS 동형암호의 부트스트래핑 근사 오류를 1/1176∼1/42(5.4∼10.2 비트 정밀도 향상) 만큼 줄일 수 있음을 보인다. 함수 합성을 사용하지 않는 부트스트래핑의 정확도는 최대 27.2∼30.3비트를 가지는 데에 반해, 함수 합성을 사용하는 부트스트래핑의 정확도는 최대 32.6∼40.5비트를 가짐을 확인한다.

둘째, 완전동형암호 알고리즘과 관련된 최첨단 기술을 사용하여 모델을 수정하

거나 재훈련 과정을 거치지 않고도 평문 상의 모델과 거의 동일한 분류 정확도를 가진 CKKS 동형암호 상의 정보 보호 ResNet-20 네트워크를 구현한다. 여기서 성능을 더욱 향상시키기 위해 여러 채널에 대한 희소 출력 데이터를 조밀하게 구성하는 *멀티플렉스 병렬 컨볼루션*을 사용하여 총 부트스트래핑의 수행 시간을 최소화한다. 우리는 또한 근사 ReLU 연산 과정 동안 심층 신경망의 극심한 발산 현상을 방지하는 *허수부 노이즈 제거 부트스트래핑*을 제안한다. 또한 레벨 소비를 최적화하고 더 가볍고 최적화된 매개 변수를 사용한다. 시뮬레이션을 통해 기술의 효율성을 보인 결과, 멀티플렉스 병렬 컨볼루션 기술이 없는 구현에 비해 ResNet-20의 추론 수행 시간이 4.67 배 적고 환산 수행 시간 (이미지당 수행 시간)이 134 배 적으며 표준 128비트 보안 수준을 달성한다는 것을 보여준다. 또한 CIFAR-10 데이터셋에 대한 ResNet-32, 44, 56, 110 모델과 CIFAR-100 데이터셋에 대한 ResNet-32를 성공적으로 구현했으며 분류 정확도는 평문 상의 정확도와 유사하다. 최대 분류 정확도는 92.9%로, 유사한 정보 보호 인공지능 모델 중 가장 높은 정확도이다.

셋째, BFV 및 CKKS 동형암호를 사용하는 클라이언트와 서버의 부담을 줄이기 위해 동형 암호 알고리즘과 관련된 새로운 개념인 *계층적 갈루아 키 생성 방법*이 제안된다. 제안된 방법의 주요 개념은 계층적 갈루아 키이며, 클라이언트가 소수의 높은 키 레벨의 갈루아 키를 생성하고 서버로 전송하면, 서버는 공개키와 소수의 높은 키 레벨의 갈루아 키 집합에서 필요한 갈루아 키를 생성할 수 있다. 이 제안된 방법은 갈루아 키 생성을 위한 클라이언트의 연산량과 갈루아 키 전송을 위한 통신 비용을 크게 줄인다. 예를 들어, CIFAR-10 데이터셋을 위한 표준 ResNet-20 네트워크와 ImageNet 데이터셋을 위한 ResNet-18 네트워크가 각각 $N = 2^{16}$와 $N = 2^{17}$를 갖는 CKKS 동형암호 상에서 사전 훈련된 매개 변수로 구현되는 경우, 서버는 각각 265개 및 617개의 갈루아 키를 필요로 하는데, 이 키의 총 용량은 105.6GB 및 197.6GB이다. 또한 3단계 계층적 갈루아 키 시스템을 사용할 경우 클라이언트에 의해 생성되고 전송되는 갈루아 키 용량을 CIFAR-10 데이터셋을 위한 ResNet-20 네트워크에 대해서는 105.6GB에서 3.4GB로 줄일 수 있고, ImageNet 데이터셋을 위한 ResNet-18 네트워크에 대해서는 197.6GB에서 3.9GB로 줄일 수 있다.

넷째, 완전동형암호로 암호화된 데이터에 셸 정렬 알고리즘을 사용할 수 있도록,

무시할만큼 작은 정렬 실패 확률을 허용하고 추가 매개 변수 α와 도입하며 수정한다. 간격 수열을 2의 승수로 구성하는 경우, 입력 배열 길이가 n인 수정된 셸 정렬은 실행 시간 복잡도 $O(n^{3/2}\sqrt{\alpha + \log\log n})$와 정렬 실패 확률 $2^{-\alpha}$의 트레이드오프를 갖는 것으로 확인된다. 그것의 시간 복잡도는 평문 상의 실행 시간 복잡도 $O(n^{3/2})$에 가깝고, 실행 시간을 약간 증가시키면서 정렬 실패 확률을 지수적으로 작게 만들 수 있다. 또한 수정된 셸 정렬의 최적 창 길이도 볼록함수 최적화를 통해 근사적으로 도출된다. 수정된 셸 정렬에 대한 수치적인 분석을 하기 위해 무작위로 생성된 배열을 사용하여 검증한다. 수치적인 분석에서는 어떠한 간격 수열에도 수정 알고리즘을 적용할 수 있으며, 실제 성능이 좋은 것으로 알려진 Ciura의 간격 수열을 사용하여 수정된 Shell 정렬을 수행했을 때 실질적으로 효과적이라는 것을 알 수 있다. 수정된 셸 정렬은 TFHE 라이브러리를 통해 FHE 상에서 수행될 수 있는 다른 정렬 알고리즘과 비교한 결과, 수정된 셸 정렬은 동형 암호 상에서 추가 메모리가 필요하지 않는 정렬 알고리즘 중 실행 시간 측면에서 최고의 성능을 갖는 것으로 나타났다.

keywords: Bootstrapping, Brakerski/Fan-Vercauteran scheme (BFV), Cheon-Kim-Kim-Song scheme (CKKS), Composite function approximation, Fully homomorhpic encryption (FHE), Fully homomorphic encryption with torus (TFHE), Hierarchical Galois key generation, Imaginary-removing bootstrapping, Improved multi-interval Remez algorithm, Modified Shell sort, Multiplexed convolution, Privacy-preserving machine learning, ResNet models, RNS-variant CKKS scheme (RNS-CKKS)
student number: 2016-24750